

ТУРБО 9

Руководство по библиотечным классам

**Москва
2008**

Турбо 9: Руководство по библиотечным классам. М.: ДИЦ, 2008.

Программное обеспечение и настоящий документ не могут быть скопированы, размножены, использованы по частям для составления других текстов, переведены на другие языки, если это не оговорено в письменной форме в договоре на поставку программного обеспечения.

Программное обеспечение, описанное в настоящем Руководстве, поставляется по лицензионному соглашению и может использоваться или копироваться только в соответствии с условиями этого соглашения.

Разработчиком и генеральным распространителем программных продуктов Турбо 9 является ЗАО "ДИЦ".

Адрес: 125057, Москва, Чапаевский пер., д. 6, стр. 1

Телефоны для справок: **(499) 157-08-20, (499) 157-04-72, (495) 956-12-50**

Телефоны для консультаций зарегистрированным пользователям:

(499) 157-03-15, (499) 157-03-64

Факс: **(495) 913-2041**

E-Mail: **tb@dic.ru** (для писем), **hotline@dic.ru** (для консультаций)

Web: **<http://www.dic.ru/>**

ЗАО "ДИЦ" 1991-2008

Встроенный объектно-ориентированный язык ТБ.Скрипт предоставляет разработчикам прикладных проектов необходимый функционал для программирования операций с журналами, бланками, картотеками и другими объектами прикладного и общесистемного плана.

В вершине [иерархии классов](#) находятся 5 базовых классов:

- [Объект|Object](#)
- [Система|System](#)
- [Математика|Mathematic](#)
- [Консоль|Console](#)
- [Бухгалтерия|Books](#)

Свойства данных классов и объектов данных классов представляют собой процедуры и функции различного назначения и могут быть унаследованы производными классами, разрабатываемыми пользователями программы. В дальнейшем процедуры и функции будут называться собирательным термином методы. Кроме того часть свойств представлена в виде полей (переменных - членов) и констант специального назначения.

К особому типу свойств относятся и [события](#), ассоциированные с некоторыми классами объектов. По сравнению с полями, процедурами и функциями события представляют собой как-бы обратный программный интерфейс: изменение или вызов любого свойства за исключением события обрабатывается самой системой, в то время как событие, наоборот, предполагает обработку изменений, произошедших внутри системы, прикладной программой. События могут возникать как в ответ на изменения состояния программы, так и в результате действия пользователя (например, по нажатию кнопки). События представлены в объектах в виде полей строкового типа, в которых записываются имена процедур или функций, вызываемых системой в момент возникновения события.

Важно отметить, что в общем случае существуют как глобальные [свойства классов](#) (статические члены), так и свойства конкретного объекта (экземпляра класса). Кроме того, программист может управлять *областью видимости свойств и методов*, разделяя их на те, что доступны из любого места исходного кода (в том числе из других классов) через полностью квалифицированное имя, включающее имя самого класса, и те, что доступны только из того же класса, где они описаны. Свойства первого рода называются публичными (public), а второго - личными (private). Все свойства объектов ([и public, и private](#)) доступны из объектов производных классов.

Свойства любого из четырех базовых классов - **Система, Математика, Консоль, Бухгалтерия** - допускается использовать в краткой форме без полного разыменования. К свойствам объектов можно обращаться только при указании конкретного родительского объекта.

Класс не может иметь два метода с одним именем, но разным списком аргументов, однако модель поддерживает полиморфизм, то есть в производных классах могут быть определены свойства, перекрывающие одноименные свойства базовых классов.

Очевидно, что объекты могут инкапсулироваться другими объектами, то есть использоваться в качестве свойств других объектов (например, объект **Шрифт** входит в состав объекта **КлеткаШаблона**).

Описание встроенных классов выполняется по одним и тем же [правилам](#) и требует хорошего знания [принципов программирования](#) классов на языке ТБ Скрипт.

В дальнейшем для изложения материала применяется единый порядок описания языковых структур. Для каждой из них указываются:

- описание (формат вызова или обращения, англоязычный синоним);
- для методов - аргументы (входные, выходные, константные) и их типы;
- назначение, причем для функции также указывается и смысл возвращаемого значения;
- пример использования.

Описание стандартных процедур и функций дается в соответствии с правилами их объявления в коде программы, то есть сначала указывается название процедуры/функции, далее в круглых скобках через точку с запятой описываются ее аргументы (в случае отсутствия аргументов круглые скобки не ставятся). После имени каждого аргумента через двоеточие следует описание его типа. Описание процедуры завершается точкой с запятой, например:

```
ОчиститьБланк(ИмяБл :Строка; ОчищатьПоля :Логическое);
```

При описании функции за списком аргументов через двоеточие следует ее тип, например:

```
ПроводАналитика(НомерПр :Целое) : Строка;
```

Язык ТБ.Скрипт разрешает указывать *русское или английское название* (синоним) стандартной процедуры/функции.

Существует несколько функций, которые принимают аргументы различных типов. В таких случаях типы перечисляются через вертикальную черту в фигурных скобках или же просто используется универсальный тип Variant, если аргумент может быть любого типа.

```
Цел(Выражение : {Целое | Число | Строка}) : Целое;
```

Пара квадратных скобок после имени аргумента означает, что он принимает [переменную-массив](#), например:

```
Альтернатива(Заголовок: Строка; Строки[ ] :Строка; Ширина :Целое) : Целое;
```

Необязательные аргументы заключаются в квадратные скобки и могут отсутствовать в обращении к процедуре/функции, например:







```
Оборот(УсловиеОтбора :Строка; Дата1 :Дата; Дата2 :Дата [;Показатель :Строка] ) : Число;  
ВыборПризнака [(Условие : Строка)] : Строка;
```

Некоторые аргументы используются для передачи значения не только в процедуру или функцию, но и в обратном направлении - из процедура или функции в вызывающий фрагмент кода. Такие аргументы помечаются ключевым словом **Var**, например:

```
func OnLookup(C :TemplateCell; Value :String; var NewValue : Пример.Накладная) : Logical;
```

В данном случае переменная **NewValue** может быть использована для передачи значения (указателя на документ типа "Накладная") из функции **OnLookup** во внешнюю процедуру или функцию.

При описании свойств и перечислимых типов используются следующие графические обозначения:

-  - поле, доступное на чтение и запись;
-  - поле, доступное только на чтение;
-  - функция;
-  - процедура;
-  - событие;
-  - перечислимый тип.

Некоторые свойства классов (например, свойство **Шрифт** клетки шаблона) доступны только на чтение. Это означает, что им нельзя присвоить новое значение. В случае, когда такое свойство является объектным (то есть представляет собой указатель на другой объект), программа, написанная на языке ТБ.Скрипт, может изменять параметры данного объекта, изменяя тем самым частичные характеристики свойства. Например:

```
Шаблон.ТекущаяКлетка.Шрифт.Размер = 12;
```

Базовые функции и процедуры

Базовые процедуры и функции относятся к классам [Система](#), [Математика](#), [Бухгалтерия](#) и [Консоль](#). По функциональному назначению их можно разделить на следующие группы:

- [Математические функции](#)
- [Строковые функции и процедуры](#)
- [Функции даты и времени](#)
- [Функции и процедуры для работы с массивами](#)
- [Функции и процедуры для работы с записями](#)
- [Системные диалоги](#)
- [Общесистемные процедуры и функции](#)
- [Функции и процедуры для работы с буфером обмена](#)
- [Функции и процедуры для работы с файловой системой](#)
- [Функции и процедуры бухгалтерского учета](#)
- [Отладочные функции и процедуры](#)
- [Процедуры и функции управления исключительными ситуациями](#)
- [Сервисные функции и процедуры](#)
- [Процедуры и функции для работы с окнами](#)
- [Процедуры и функции для работы с отчетами](#)

Класс **Система** включает следующие

- группы процедур и функций:

- [Строковые функции и процедуры](#)
- [Функции даты и времени](#)
- [Функции и процедуры для работы с массивами](#)
- [Функции и процедуры для работы с записями](#)
- [Функции и процедуры для работы с файловой системой](#)
- [Процедуры и функции управления исключительными ситуациями](#)
- [Общесистемные процедуры и функции](#)

- свойства:



[ИнфСессии / SessionInfo](#)



[ИнфБазы / BaseInfo.](#)

- перечислимые типы, указанные в теме: ["Константы"](#).

В классе **Система / System** определены следующие перечислимые типы:

[Константы для определения режимов работы с файлами](#)

[Константы режимов поиска файлов](#)

[Константы режимов синхронизации файлов](#)

[Константы типов иконок](#)

[Константы типов разыменования фильтра](#)

Константы для определения режимов работы с файлами ([РежимыФайлов / FileModes](#))

В функциях классов [DBFФайл](#), [ТекстовыйФайл](#) используются следующие константы, определенные с помощью перечислимого типа **РежимыФайлов / FileModes**:

- **рфСоздать (fmCreate)** - создать файл на запись; если файл уже существовал, все его содержимое удаляется, а длина устанавливается равной нулю;
- **рфОткрНаЧтение (fmOpenRead)** - открыть существующий файл на чтение; указатель файла устанавливается на начало; если файла нет, генерируется исключительная ситуация;
- **рфОткрНаЗапись (fmOpenWrite)** - открыть существующий файл на запись; содержимое существующего файла не удаляется, но указатель файла устанавливается на начало и при последующих операциях записи старая информация будет последовательно затираться новой; размер файла не меняется вплоть до того момента, когда объем записываемой информации превышает тот, что уже был в файле, после чего размер файла увеличивается синхронно с записью; если файла нет, генерируется исключительная ситуация;
- **рфОткрНаЧтениеЗапись (fmOpenReadWrite)** - открыть существующий файл на запись и чтение; содержимое существующего файла не удаляется; указатель файла устанавливается на начало; последующие операции чтения и записи смещают указатель по файлу; если указатель находится в пределах размера файла, записываемая информация "ложится" поверх старых данных, а в противном случае файл соответствующим образом расширяется; если файла нет, генерируется исключительная ситуация.

Перечислимый тип **РежимыПоискаФайлов / FindFileModes** используются функцией [ВзятьСписокФайлов](#) и содержит следующие константы:

ИскатьРекурсивно/RecursiveFind - поиск с подкаталогами; если данный режим не указан, поиск ведется только в указанном каталоге;

ИскатьТолькоФайлы/FindOnlyFiles - поиск только файлов;

ИскатьТолькоКаталоги/FindOnlyFolders - поиск только каталогов;

Если ни один из двух последних режимов не включен, то ведется поиск и файлов, и каталогов.

Для работы процедуры [СинхронизироватьФайл](#) используются следующие константы, задающие режимы синхронизации файлов и определенные в перечисляемом типе **РежимСинхронизацииФайлов / SynchronizeFileMode**:

- **НаСервере / ClientToServer** - локальная версия файла создаётся или обновляется на сервере;
- **НаКлиенте / ServerToClient** - серверная версия файла переносится или обновляется на локальном компьютере;
- **Автоматически / Auto** - необходимость и направление синхронизации выбирается автоматически. Значение, заданное по умолчанию.

Перечислимый тип **ТипИконки / IconType**, определенный в классе **Система**, содержит допустимые константы для описания стандартных значков (иконок), которые могут выводиться в сообщениях об ошибке.

Данный тип содержит следующие константы:

- **itInformation|тиИнформация** - системная иконка "информация" (с восклицательным знаком);
- **itQuestion|тиВопрос** - системная иконка "вопрос" (с вопросительным знаком;)
- **itWarning|тиПредупреждение** - системная иконка "предупреждение";
- **itError|тиОшибка** - системная иконка "ошибка";
- **itApplication|тиПрограмма** - иконка Студии;
- **itWinLogo|тиСистема** - системная иконка Windows.

Вышеприведенные константы используются в свойстве [УстОшибку / SetError](#).

Константы типов разыменования фильтра











Перечислимый тип **ТипыРазыменованияФильтра / FilterDesignateTypes**, определенный в классе **Система / System**, предназначен для заданий условий на фильтры с использованием макросов.

В данном типе определены следующие константы:

- **byDocID** - макрос будет раскрываться в полный DocID;
- **byForeignKey** - макрос будет раскрываться в ForeignKey;
- **byDescription** - макрос будет раскрываться в описательное поле.

Константы, определенные в данном типе, используются в методе [ВосстановитьФильтрЗаписей / ComposeRecordFilter](#) класса **System**.

К общесистемным процедурам и функциям относятся следующие процедуры и функции из класса [Система](#):

-  [Функция Версия / Version](#)
-  [Функция ВызовНаСервере / ServerCall](#)
-  [Функция НайтиКласс / FindClass](#)
-  [Функция ПроверкаЛицензии / LicenseCheck](#)
-  [Функция ТипПеременной / VarType](#)
-  [Функция СлучайноеЧисло / Random](#)
-  [Процедура СлучайноеНачало / Randomize](#)
-  [Функция РежимОтладки / DebugMode](#)
-  [Функция ВыполнитьSQL / ExecuteSQL](#)
-  [ДобКонтрольнуюСумму32 / AddCRC32](#)
-  [ДобКонтрольнуюСумму64 / AddCRC64](#)

Процедура Вставка / Insertion

Описание

Вставка (Журнал:Строка; ДатаОп: Дата; Опер: Строка [; Ч:Логическое]);
Insertion (Journal:String; DateOp: Date; Oper: String [; X:Logical]);

Замечание

Процедура пока не реализована.

Назначение

Вставляет в заданный журнал проводку или типовую операцию за указанную дату.

Описание

```
ДобКонтрольнуюСумму32(var CRC :Строка; Значение :Вариант);  
AddCRC32(var CRC :String; Value :Variant);
```

Аргументы

CRC - строковое представление контрольной суммы, к которой надо добавить контрольную сумму значения, переданного в параметре **Значение**. Если контрольная сумма ещё не была создана, в данном параметре надо передать пустую строку;

Значение - значение, CRC которого необходимо добавить к переданному в параметре CRC.

Внимание. Параметр CRC - строка шестнадцатеричного представления контрольной суммы. На выходе из метода она будет дополнена слева недостающими символами '0' (ноль) до 8 символов (для CRC32).

Назначение

Метод вычисляет 32-битную контрольную сумму значения, переданного в параметре **Значение**, и добавляет вычисленную контрольную сумму к переданной в параметре CRC.

Описание

```
ДобКонтрольнуюСумму64(var CRC :Строка; Значение :Вариант);  
AddCRC64(var CRC :String; Value :Variant);
```

Аргументы

CRC - строковое представление контрольной суммы, к которой надо добавить контрольную сумму значения, переданного в параметре **Значение**. Если контрольная сумма ещё не была создана, в данном параметре надо передать пустую строку;

Значение - значение, CRC которого необходимо добавить к переданному в параметре CRC.

Внимание. Параметр CRC - строка шестнадцатеричного представления контрольной суммы. На выходе из метода она будет дополнена слева недостающими символами '0' (ноль) до 16 символов (CRC64).

Назначение

Метод вычисляет 64-битную контрольную сумму значения, переданного в параметре **Значение**, и добавляет вычисленную контрольную сумму к переданной в параметре CRC.

Описание

```
ЗавершитьИзменениеСхемы ([Сохранить: Логическое]);  
EndChangeSchema ([Save: Logical]);
```

Аргументы

Сохранить - логическое значение, которое определяет, следует ли записать проведенные изменения в файл (TRUE) или же оставить их действовать лишь временно до конца сессии (FALSE); если параметр опущен - он подразумевается равным TRUE

Назначение

Завершает программное изменение текущей схемы доступа, начатое с помощью процедуры [НачатьИзменениеСхемы](#). Если параметр **Сохранить** равен TRUE или опущен, проведенные изменения сохраняются в долговременной памяти. В противном случае изменения действуют до конца текущей сессии.

Пример

```
class БланкЗаказа;  
  
--...  
  
proc SwitchPrinting;  
    BeginChangeSchema;  
    try  
        -- меняем право на печать из того самого класса,  
        -- который подвергается изменению  
        Access.CanPrint = not Access.CanPrint;  
        -- Trace(Access.CanPrint);  
    finally  
        -- изменения действуют лишь в текущей сессии  
        EndChangeSchema (False);  
    end;  
end;
```

Описание

НачатьИзменениеСхемы;
BeginChangeSchema;

Назначение

Разрешает начать программное изменение текущей схемы доступа во время сессии (подключенный пользователь должен иметь право на администрирование). Для того чтобы проведенные изменения вступили в силу, необходимо вызывать процедуру [ЗавершитьИзменениеСхемы](#), парную данной.

Пример

```
var R :AccessRights;  
  BeginChangeSchema;  
  try  
    R = AccessRights.Create(AccessRights.sgBlanks,'Small.Пустой');  
    -- включаем/отключаем доступность бланка  
    R.Доступен = not R.Доступен;  
    --Trace(R.Доступен);  
  finally  
    EndChangeSchema(True);  
  end;
```

Описание

```
СлучайноеНачало ( [Инициализация:Целое] );  
Randomize ( [Init:Integer] );
```

Аргументы

Инициализация - необязательный параметр, целое число, определяющее вид случайной последовательности, генерируемой при последующих вызовах функции [СлучайноеЧисло](#).

Назначение

Явным образом инициализирует генератор случайных чисел. Если необязательный параметр не задан, то инициализация проводится случайным образом на основе текущего времени.

Указание параметра инициализации позволяет обеспечить воспроизводящуюся из раза в раз последовательность случайных чисел, возвращаемую функцией [СлучайноеЧисло](#). Разные параметры инициализации приводят к генерации различных последовательностей.

Пример

```
proc P1;  
  var Дано : Целое[];  
  Randomize(1);  
  Дано[1] = Random ( 10 );  
  Дано[2] = Random ( 10 );  
  Дано[3] = Random ( 10 );  
  -- всегда получаем последовательность чисел  
  -- 0, 8, 2  
end;
```

Функция **Версия** / **Version**

Описание

Версия:Строка;
Version:String;

Назначение

Возвращает название версии программы Студия, используемой в данный момент.

Пример

```
-- выводим на экран версию Студии  
Message(Version);
```

Описание

```
ВызовНаСервере(DLL:Строка; Имя:Строка; Параметр:Вариант) : Вариант;  
ServerCall(DLL:String; Name:String; Parameter:Variant) : Variant;
```

Аргументы

DLL - путь к файлу DLL-библиотеки, функцию из которой требуется выполнить на сервере; если здесь задано только имя файла, то он ищется в каталоге Студии и по системному пути;

Имя - имя функции;

Параметр - параметр, передаваемый в функцию.

Назначение

Функция предназначена для интеграции проектов Студии с внешними программами, не поддерживающими COM-технологии.

Функция загружает на сервере указанную DLL и вызывает в ней требуемую функцию. Затем возвращает результат работы серверной функции или генерирует исключение в случае ошибки.

Замечание

Вызываемая функция должна иметь следующий прототип (в Pascal-нотации):

```
function RemoteFunction(Key, Schema, User: PChar; In: OleVariant; Out: OleVariant):  
WordBool;
```

Первые три параметра предоставляет сам сервер. В параметре **In** будет передаваться значение, заданное последним параметром функции **ВызовНаСервере**. Вызываемая на сервере функция должна записать результат своей работы в выходной параметр **Out**, который затем в качестве выходного значения возвращается функцией **ВызовНаСервере**. Если серверная функция завершилась успешно, она должна вернуть 0. Любое другое значение интерпретируется как признак ошибки. При этом в параметр **Out** должна быть занесена строка с описанием ошибки. В коде ТБ.Скрипт при этом будет сгенерировано исключение с данным описанием.

Описание

```
ВыполнитьSQL(Запрос :Строка; {Запись :Класс Запись}; {Параметры :Строка[]}; {var  
Значения :Вариант[]}; {Процедура :Логическое}) :Вариант;  
ExecuteSQL (SQL :String; {Record :Class Record}; {Parameters :String[]}; {var Values :Variant  
[]}; {Procedure :Logical}) :Variant;
```

Аргументы

Запрос - строка с произвольным текстом SQL-запроса или имя хранимой процедуры, если указан параметр **Процедура|Procedure** (см. описание ниже);

Запись - необязательный параметр, идентификатор класса документа. SQL-запрос выполняется для базы, в которой размещен данный документ.

Если он не указан, то SQL-запрос будет выполняться в базе, где размещен класс записей Kernel.Settings.User;

Параметры - необязательный параметр, строковый массив с именами параметров указанных в запросе (параметр **Запрос|SQL**);

Значения - необязательный параметр, массив значений произвольного типа для передачи и приема значений в SQL-запросе.

Если указан массив параметров (**Параметры|Parameters**), то элемент со значением будет передан в соответствующий имени параметр SQL-запроса.

Количество элементов в массиве должно быть не меньше чем количество элементов в массиве с именами параметров.

Если массив параметров не указан, то значение будет передано в соответствующий параметр SQL-запроса по порядку;

Процедура - необязательный параметр логического типа. Если он указан и равен значению True, то это значит, что будет выполняться хранимая процедура, а значит можно получить результирующее значение и новые значения в параметрах из процедуры (если это в ней предусмотрено).

В этом случае параметр **Запрос|SQL** должен содержать только имя хранимой процедуры, а имена параметров и входные значения должны размещаться в соответствующих массивах аргументов функции. Результирующее значение после выполнения хранимой процедуры, можно получить в виде результата выполнения функции.

Назначение

Функция предназначена для выполнения произвольных SQL-запросов и запуска сторонних хранимых процедур.

Функция возвращает значение в результате выполнения хранимой процедуры. При условии, что в хранимой процедуре это предусмотрено. В иных случаях значение может содержать произвольное значение произвольного типа.

Функция работает только с теми СУБД, к которым Сервер данных подключается через ADO.

Пример

```
proc Button5OnClick(Sender :Button);  
var vSQL :String;  
var vEoL :String;  
var vResult :Variant;  
var vValues[] :Variant;  
  
-- Удаляем хранимую процедуру, если она есть в базе, где размещен  
-- класс документов Пресс.Справочники.Товар  
vEoL = Chr(13) + Chr(10);  
vSQL = "if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[sp_test]'))" +  
" and OBJECTPROPERTY(id, N'IsProcedure') = 1)" + vEoL +  
"drop procedure [dbo].[sp_test]";  
  
ExecuteSQL(vSQL, Пресс.Справочники.Товар);  
  
-- Создаем хранимую процедуру, которая второй параметр
```

```

-- переприсваивает в первый с конвертацией
-- и возвращает его в виде результата

vSQL = "CREATE PROCEDURE [dbo].[sp_test]" + vEoL +
        "@prm1 int output," + vEoL +
        "@prm2 varchar(30)" + vEoL +
        "AS" + vEoL +
        "set @prm1 = Convert(int, @prm2)" + vEoL +
        "return @prm1";

ExecuteSQL(vSQL, Пресс.Справочники.Товар);

-- Выполняем хранимую процедуру, указав явно параметры
-- Возвращаемый результат не нужен

ExecuteSQL("sp_test @prm1=12, @prm2='21'", Пресс.Справочники.Товар);

-- Выполняем хранимую процедуру, имена параметров и значения указаны
-- в соответствующих аргументах
-- Возвращаемый результат не нужен

ExecuteSQL("sp_test :@prm1, :@prm2", Пресс.Справочники.Товар, ['@prm1',
'@prm2'], [12, '21']);

-- Выполняем хранимую процедуру, имена параметров опущены,
-- Указанные значения будут переданы по-порядку
-- Возвращаемый результат не нужен

ExecuteSQL("sp_test :@prm1, :@prm2", Пресс.Справочники.Товар, [,12, '21']);

-- Выполняем хранимую процедуру, как процедуру, получаем результат

vValues = [12, '21'];
vResult = ExecuteSQL("sp_test", Пресс.Справочники.Товар, ['@prm1', '@prm2'],
vValues, True);

Trace('Result = ' + Str(vResult) + ', Values = ' + Str(vValues));
-- Result = 21, Values =[21, '21']
end;

```

Описание

```
НайтиКласс (Имя :Строка):Класс;  
FindClass (Name :String) :Class;
```

Аргументы

Имя - имя класса, который необходимо найти.

Назначение

С помощью данной функции можно провести проверку на наличие какого-либо класса в текущем проекте.

В случае успеха возвращает искомый класс, иначе **nil**.

Пример

```
var ОбъектКласса :Variant;  
  
ОбъектКласса = НайтиКласс("редПриходныйОрдер");  
If (ОбъектКласса = nil) Then  
    Message("Класс не найден.");  
    Return;  
End;  
ОбъектКласса = ОбъектКласса.Create;  
...
```


Описание

```
ПроверкаЛицензии(Номер:Целое):Целое;  
LicenseCheck(Number:Integer):Integer;
```

Аргументы

Номер - номер лицензии на проект.

Назначение

Проверяет допустимость кода лицензии и возвращает количество лицензий. Если код лицензии неверный, то возвращает 0. Применяется для защиты от копирования проектов, написанных с помощью Студии.

Лицензии регистрируются на сервере администратором после установки электронного ключа, в который "защиты" коды.

Пример

```
if ПроверкаЛицензии(123) = 0 then  
    Message("Незарегистрированная версия");  
    CloseApp;  
fi;
```

Описание

РежимОтладки : Логическое;
DebugMode : Logical;

Назначение

Возвращает TRUE, если проект выполняется в отладочной сессии, и FALSE - в противном случае.

Пример

```
B: Button;  
...  
if DebugMode = TRUE then  
    B.Visible = TRUE;  
end;
```

Описание

```
СлучайноеЧисло(Диапазон:Целое): Целое;  
Random(Range:Integer): Integer;
```

Аргументы

Диапазон - целое число или числовое выражение целого типа, задающее диапазон генерируемых случайных чисел.

Назначение

При каждом вызове возвращает случайное число из диапазона от 0 до значения **Диапазон**-1. Если через аргумент **Диапазон** было передано отрицательное значение, функция генерирует случайные числа во всей области значений типа Integer, в том числе и отрицательные.

По умолчанию, при каждом запуске проекта генерируется уникальная последовательность случайных чисел, то есть она меняется от одного запуска к другому. Программист может в целях отладки инициализировать генератор случайных чисел с помощью процедуры [СлучайноеНачало](#) - в этом случае последовательность генерируемых чисел будет воспроизводиться из раза в раз.

Пример

```
proc P1;  
  var Дано : Целое;  
  Дано = Random ( 10 ); -- случайное число от 0 до 9 включительно  
end;
```

Описание

```
ТипПеременной(Выражение :Вариант): Целое;  
VarType(Expression :Variant): Integer;
```

Аргументы

Выражение - выражение произвольного неизвестного типа.

Назначение

Позволяет узнать тип выражения (переменной), которое передается в функцию. Возвращает одну из следующих констант, определенных в проекте СИС2:


















```
varUnknown :Integer = 0; -- неизвестный тип  
varString  :Integer = 1; -- строка  
varInt      :Integer = 2; -- 4-байтовое целое  
varNumeric :Integer = 3; -- Число с плавающей точкой  
varLogical  :Integer = 4; -- Логическое (булево) значение  
varDate     :Integer = 5; -- ДатаВремя в формате TDateTime  
varObject   :Integer = 6; -- Объект  
varVariant  :Integer = 7; -- Вариант  
varArray    :Integer = 8; -- Массив  
varClass    :Integer = 9; -- Указатель на класс  
varNull     :Integer = 10; -- Пусто  
  
varUnit     :Integer = 17; -- Измеритель  
varUser     :Integer = 50; -- Пользовательский тип, в частности, этот тип  
                        -- имеют параметры процедур и функций типа Массив
```

Для использования этих констант необходимо подключить проект СИС2 в качестве подпроекта.

Пример

```
if VarType(Умолчание) = СИС2.Константы.varString then  
    Значение = Dbf.GetField(ИмяПоля);  
else  
    Значение = Str(Dbf.GetField(ИмяПоля));  
fi;
```

Для работы с датами и временем используются следующие процедуры и функции класса [Система](#):

-  [Функция Время / Time](#)
-  [Функция Год / Year](#)
-  [Функция День / Day](#)
-  [Функция ДобавитьМесяц / AddMonth](#)
-  [Функция ДобавитьСек / AddSec](#)
-  [Функция ИнтервалСек / IntervalSec](#)
-  [Функция Мес / Mon](#)
-  [Функция Минута / Minute](#)
-  [Функция ПериодСек / PeriodSec](#)
-  [Функция Сегодня / Today](#)
-  [Функция Сейчас / Now](#)
-  [Функция Секунда / Second](#)
-  [Функция СоздатьДату / CreateDate](#)
-  [Функция Час / Hour](#)
-  [Процедура Задержка / Sleep](#)
-  [Функция ДеньНедели / DayOfWeek](#)
-  [Функция СтрокаВДату / StringToDate](#)

Процедура Задержка / Sleep

Описание

```
Задержка(Секунды :Число);  
sleep(Seconds :Numeric);
```

Аргументы

Секунды - количество секунд. Для задания долей секунд используется вещественная часть аргумента.

Назначение

Вызов метода задерживает исполнение программы на количество секунд, заданных в параметре Секунды.

Пример

```
proc P1;  
  var X : Число;  
  X = Задержка(12.5);  
end;
```

Функция Время / Time

Описание

Время (CСервера :Логическое) :Дата;
Time (FromServer :Logical) :Date;

Аргументы

CСервера - признак, определяющий, следует ли узнать дату на сервере или на клиентском компьютере.

Назначение

Возвращает текущее время (часы, минуты, секунды). День, месяц и год обнулены.

Если параметр *CСервера* отсутствует или равен FALSE, возвращаемое значение соответствует системным настройкам клиентской машины, в противном случае – возвращается дата и время, установленные на компьютере сервера.

Необходимо помнить, что при попытке получить системное время серверной машины, из-за запроса этой информации с удаленного компьютера, потребуется некоторое дополнительное время.

Пример

```
proc Example;  
  var X : Date;  
  X = Time;  
  Message("Текущее время:" + Str(X));  
end;
```

В результате выполнения приведенного кода будет выдано сообщение вида "Текущее время:15:32:34".

Функция Год / Year

Описание

```
Год (День :Дата) :Целое;  
Year (Day :Date) :Integer;
```

Аргументы

День - задает произвольную дату, из которой необходимо выделить год.

Назначение

По введенной дате возвращает четырехзначный номер года в формате целого числа.

Пример

```
X : Date = 01.12.99;  
  
proc Example;  
  var Y : Integer;  
  Y = Year(X); -- получили Y = 1999  
end;
```


Функция День / Day

Описание

```
День(День:Дата):Целое;  
Day(ADay:Date):Integer;
```

Аргументы

День - задает произвольную дату, из которой необходимо выделить число (день месяца).

Назначение

Возвращает по введенной дате порядковый номер ее дня в месяце.

Пример

```
X : Date = 01.12.99;  
proc Example;  
  var Y : Integer;  
  Y = Day(X); -- получили Y = 1
```

Описание

ДеньНедели (День:Дата):Целое;
DayOfWeek (ADay:Date):Integer;

Аргументы

День - задает произвольную дату, для которых необходимо определить, какой это день недели.

Назначение

Возвращает порядковый номер дня недели, то есть число от 1 до 7 включительно, для указанной даты.

Пример

```
X : Date = 01.12.99; -- 1 декабря 1999
proc Example;
  var X : Integer;
  X = DayOfWeek(X);
  -- X = 3, то есть 1 декабря 1999 - это среда
end;
```

Описание

```
ДобавитьМесяц(День:Дата [; Добавить:Целое]):Дата;  
AddMonth(ADay:Date [; Add:Integer]):Date;
```

Аргументы

День - задает произвольную дату, к которой необходимо прибавить месяц или несколько месяцев, число которых определяется аргументом **Добавить**.

Назначение

Прибавляет к некоторой дате, заданной первым параметром, один месяц, если второй параметр опущен, или же то количество месяцев, что указано во втором параметре. Второй параметр может быть как положительным, так и отрицательным. Если в исходной дате были определены не только день, месяц, год, но и время, то время отсекается, и в возвращаемом значении даты время равно нулю.

Пример

```
proc Example;  
  var X : Date;  
  X = AddMonth(Now,3); -- через три месяца от сего дня  
end;
```

Описание

```
ДобавитьСек(Время:Дата ; Добавить:Число):Дата;  
AddSec(TimeStamp:Date ; Add:Numeric):Date;
```

Аргументы

Время - задает дату и время, к которому необходимо прибавить указанное число секунд;

Добавить - задает желаемый временной сдвиг в секундах (может быть отрицательным).

Назначение

Прибавляет к указанному в первом параметре времени число секунд, переданное во втором параметре. Возвращает новое значение типа Дата.

Пример

```
func Interval: Date;  
  var X : Date;  
  X = AddSec(Now,180); -- интервал три минуты  
  return X;  
end;
```

Описание

ИнтервалСек :Число;
IntervalSec :Numeric;

Назначение

Функция возвращает время от начала работы Windows в секундах. Точность результата зависит от длительности непрерывной работы Windows. Интервал в пределах одних суток измеряется с точностью не хуже миллионной доли секунды. С увеличением интервала точность падает.

Равные по длительности интервалы представляются с разной точностью на компьютерах с разной производительностью: чем выше производительность, тем выше точность результата.

Пример

```
-- процедура измеряет время, необходимое для
-- выполнения заданного выражения
proc BenchMark(Expression :String);
  var X1, X2 :Numeric;
  X1 = ИнтервалСек;
  self.Evaluate(Expression);
  X2 = ИнтервалСек;
  trace("Вызов '" + Expression + "' занял " + Str(X2-X1) + " секунд");
end;
```

Функция Мес / Mon

Описание

```
Мес(День:Дата):Целое;  
Mon(ADay:Date):Integer;
```

Аргументы

День - задает произвольную дату, из которой необходимо выделить месяц.

Назначение

Для введенной даты возвращает номер месяца, то есть целое число от 1 до 12 включительно.

Пример

```
X : Date = 01.12.99;  
proc Example;  
  var Y : Integer;  
  Y = Mon(X); -- получили Y = 12  
end;
```

Функция Минута / Minute

Описание

Минута(ДеньиВремя:Дата):Целое;
Minute(DayTime:Date):Integer;

Аргументы

ДеньиВремя - задает произвольную дату (со временем), из которой необходимо выделить минуты.

Назначение

Возвращает для указанной даты (со временем) минутную составляющую.

Пример

```
proc Example;  
  var X : Date;  
  var Y : Integer;  
  X = Time; -- определили, сколько сейчас времени, например, 15:30:10  
  Y = Minute(X); -- вычленили минуты, то есть Y = 30  
end;
```

Описание

```
ПериодСек(ДеньВремя1:Дата; ДеньВремя2:Дата):Число;  
PeriodSec(DayTime1:Date; DayTime2:Date):Numeric;
```

Аргументы

ДеньВремя1 и **ДеньВремя2** - задают две произвольных даты (со временем), для которых необходимо получить разницу во времени с точностью до секунды.

Назначение

Возвращает число секунд между моментом времени ДеньВремя1 и моментом ДеньВремя2 (то есть, ДеньВремя1-ДеньВремя2). Возвращаемое значение больше нуля, если параметр ДеньВремя1 содержит более поздний момент времени, нежели ДеньВремя2, или отрицательное значение в противном случае.

Пример

```
proc Example;  
  var X : Numeric;  
  X = PeriodSec(X2,X1);  
  -- определили, сколько секунд в сутках, 86400  
end;
```


Функция Сегодня / Today

Описание

Сегодня (ССервера : Логическое) :Дата;
Today (FromServer : Logical) :Date;

Аргументы

ССервера - признак, определяющий, следует ли узнать дату на сервере или на клиентском компьютере.

Назначение

Функция **Сегодня** по разному работает в коде классов (бланков, картотек и пр.) и типовых операций. В классах она возвращает текущую дату. В типовой операции она выдает значение текущей даты из контекста журнала типовых операций (дату операции).

В обоих случаях возвращаемое значение содержит только число, месяц и год, а время (часы, минуты и секунды) рано нулю.

При использовании в ТБ.Скрипт, в зависимости от значения параметра, возвращается либо дата клиентской машины (**ССервера** равно FALSE или опущен), либо серверной (**ССервера** равно TRUE).

Необходимо помнить, что при попытке получить системную дату серверной машины, из-за запроса этой информации с удаленного компьютера, потребуется некоторое дополнительное время.

Пример

```
proc Example;  
  var X : Date;  
  X = Today;  
  Message("Сегодня у нас:" + Str(X));  
end;
```

В результате выполнения приведенного кода будет выдано сообщение вида «Сегодня у нас:26.06.1999».

Функция Сейчас / Now

Описание

Сейчас (ССервера : Логическое) :Дата;
Now (FromServer : Logical) :Date;

Аргументы

ССервера - признак, определяющий, следует ли узнать время на сервере или на клиентском компьютере.

Назначение

Возвращает текущую дату и время с точностью до секунды.

Если параметр **ССервера** отсутствует или равен FALSE, возвращаемое значение соответствует системным настройкам клиентской машины, в противном случае – возвращается дата и время, установленные на компьютере сервера.

Необходимо помнить, что при попытке получить системное время серверной машины, из-за запроса этой информации с удаленного компьютера, потребуется некоторое дополнительное время.

Пример

```
proc Example;  
  var X : Date;  
  X = Now;  
  Message("Сейчас у нас:" + Str(X));  
end;
```

В результате выполнения приведенного кода будет выдано сообщение вида "Сейчас у нас:26.06.1999 12:36:54". Забегая вперед, скажем, что функция Str преобразует внутреннее (машинное) представления даты в строковую константу, формат которой в общем случае зависит от настроек Студии (**Сервис|Настройки программы|Формат|Формат даты**).

Функция Секунда / Second

Описание

Секунда(ДеньиВремя:Дата):Число;
Second(DayTime:Date):Numeric;

Аргументы

ДеньиВремя - задает произвольную дату (со временем), из которой необходимо выделить секунды.

Назначение

Возвращает для указанной даты (со временем) секундную составляющую.

Пример

```
proc Example;  
  var X : Date;  
  var Y : Numeric;  
  X = Time; -- определили, сколько сейчас времени, например, 15:30:10  
  Y = Секунда(X); -- вычленили секунды, то есть Y = 10  
end;
```

Описание

```
СоздатьДату(День:Ц; Месяц:Ц; Год:Ц [;Часы:Ц] [;Мин:Ц] [;Сек:Ц]):Дата;  
CreateDate(dd:I; mo:I; yy:I [;hh:I] [;mn:I] [;sec:I]):Date;
```

Аргументы

День, Месяц и Год - задают соответственно число, месяц и год требуемой даты.

Необязательные аргументы **Часы, Мин и Сек** дают возможность дополнить значение типа "дата" конкретным временем с точностью до секунды. Если они не указаны, время считается равным нулю. Допустимо указать только часы или только часы и минуты, тогда опущенные аргументы принимаются равными нулю. Если младшие единицы времени указаны в отсутствии старших (например, секунды без минут или минуты без часов), то позиции старших единиц следует обозначить запятыми (см. пример).

Назначение

Возвращает значение типа дата, сконструированное на основе переданных значений целого типа, задающих день, месяц, год, часы, минуты, секунды.

Пример

```
proc Example;  
  var X : Date;  
  X = CreateDate(29,2,2000,,,44);  
  Message("Время Ч:" + Str(X));  
end;
```

В результате выполнения приведенного кода будет выдано сообщение вида "Время Ч:29.02.2000 00:00:44", то есть 29 февраля 2000 года и 44 секунды. Часы и минуты опущены.

Описание

```
СтрокаВДату(День:Строка):Дата;  
StringToDate(ADay:String):Date;
```

Аргументы

День - задает строковое представление произвольной даты, которое необходимо преобразовать в значение типа "дата".

Назначение

Преобразует строку в дату, предварительно проверяя строку на соответствие синтаксису записи значений типа даты ("DD.ММ.YY" или "DD.ММ.YYYY"). Если строка неправильная - на стадии выполнения кода выдается сообщение об ошибке.

Пример

```
proc Example;  
  var X : Date;  
  X = StringToDate("01.13.1999"); -- получим сообщение об ошибке  
end;
```

Функция Час / Hour

Описание

```
Час (ДеньиВремя:Дата):Целое;  
Hour (DayTime:Date):Integer;
```

Аргументы

ДеньиВремя - задает произвольную дату (со временем), из которой необходимо выделить час.

Назначение

Возвращает для указанной даты (со временем) часовую составляющую.

Пример

```
proc Example;  
  var X : Date;  
  var Y : Integer;  
  X = Time; -- определили, сколько сейчас времени, например, 15:30:10  
  Y = Hour(X); -- получили текущий час, то есть Y = 15  
end;
```

Функции и процедуры для работы с записями

Для работы с записями используются следующие процедуры и функции из класса [Система](#):

-  [Функция ВосстановитьФильтрЗаписей / ComposeRecordFilter](#)
-  [Процедура ЗавершитьИзоляцию / EndIsolation](#)
-  [Процедура ЗавершитьТранзакцию / EndTransaction](#)
-  [Функция ЗаписьПоИмениОбщегоФайла / GetRecordByCommonFileName](#)
-  [Процедура ИмпортЗаписей / RecordsImport](#)
-  [Функция ИмяОбщегоФайлаПоЗаписи / GetCommonFileNameByRecord](#)
-  [Процедура НачатьИзоляцию / BeginIsolation](#)
-  [Процедура НачатьТранзакцию / BeginTransaction](#)
-  [Процедура ОтменитьТранзакцию / AbortTransaction](#)
-  [Функция РазложитьФильтрЗаписей / DecomposeRecordFilter](#)
-  [Процедура УстановитьСообщениеПриБлокировке / SetLockMessage](#)
-  [Процедура ЭкспортЗаписей / RecordsExport](#)

Описание

```
ЗаписьПоИмениОбщегоФайла(ИмяОбщегоФайла :Строка) :Запись;  
GetRecordByCommonFileName (CommonFileName :String) :Record;
```

Аргументы

ИмяОбщегоФайла - строковая переменная, содержащая имя файла.

Назначение

По заданному имени файла функция возвращает запись, принадлежащую картотеке [Kernel.Settings.CommonFile](#), в которой хранится содержание данного файла. Функция возбудит исключение, если имя файла не задано или указывает на файл/каталог, который отсутствует в папке, путь к которой определяется с помощью свойства [КаталогОбщихФайловИБ](#).

Пример

```
var ИмяФайла :Строка;  
var ВЗаписи :Kernel.Settings.CommonFile;  
ИмяФайла =  
  
ВЗаписи = ЗаписьПоИмениОбщегоФайла(ИмяФайла);  
If (ВЗаписи <> nil) Then  
    Message("Размер файла: " + Str(ВЗаписи.Size) + " байт");  
End;
```

См. также функцию [ИмяОбщегоФайлаПоЗаписи](#).

Описание

ИмяОбщегоФайлаПоЗаписи(Запись :Запись) :Строка;
GetCommonFileNameByRecord (Record :Record) :String;

Аргументы

Запись - запись картотеки [Kernel.Settings.CommonFile](#), в которой хранится содержимое файла.

Назначение

Возвращает строковое имя общего файла, записанное в поле **Name** записи картотеки **Kernel.Settings.CommonFile**. Если запись не принадлежит указанной картотеке или параметр Запись = nil, функция возбудит исключение.

Внимание. Функция вернёт пустую строку, если Запись<>nil, но в картотеке **Kernel.Settings.CommonFile** помечена как удалённая (Deleted).

Пример

```
var i                      :Integer;
var ВсегоРазмерВВбайтах   :Integer;
var Q                     :Query;
var ТекущаяЗапись        :Kernel.Settings.CommonFile;
var ПолноеИмяФайла       :String;

Q = Query.Create([Kernel.Settings.CommonFile]);
Q.Filter = 'IsGroup = 0';
Q.Select;

For i = 1..Q.Count do
    ТекущаяЗапись = Q.Current;
    ПолноеИмяФайла = ИмяОбщегоФайлаПоЗаписи(ТекущаяЗапись);
    Trace("Имя файла: " + ПолноеИмяФайла + ", размер: " +
        Стр(ТекущаяЗапись.Size) + " байт.")
    ВсегоРазмерВВбайтах = ВсегоРазмерВВбайтах + ТекущаяЗапись.Size;
    Q.Next;
End;

Message("Всего файлов: " + Стр(i) +
    ", Всего байт: " + Стр(ВсегоРазмерВВбайтах));
```

См. также функцию [ЗаписьПоИмениОбщегоФайла](#).

Если при обработке документа есть вероятность, что он не изменится (что зависит от внутренних условий в цепочке выполнения инструкций алгоритма), то рекомендуется не вызывать метод **Edit** (объекта класса **Запись** или производного класса) заранее. Это нужно делать лишь непосредственно перед модификацией (в той ветке алгоритма, где необходимость редактирования документа стала очевидной, а не вероятной). Каждый вызов Edit приводит к обращению на сервер, что замедляет работу.

Пример:

```
-- КАК НЕ НАДО ДЕЛАТЬ
PROC ОбработатьДокумент(D:Документы.Движение_Ресурсов_И_Обязательств);
  Var ТипСоб:Integer;

  ТипСоб = D.ТипСоб;
  IF ((ТипСоб=4 ИЛИ ТипСоб=16) И D.МыИ И D.Источник<>D.Получатель) THEN
    -- Отгрузка контрагенту
    D.Номер = 10;
  ELSIF ТипСоб=11 THEN
    -- Списание,
    -- изменений не нужно
  FI;
END;

PROC ОбработатьДокументы(Q:Запрос);
  While НЕ Q.EOF do
    D=Q.Current;
    D.Edit;
    ОбработатьДокумент(D);
    IF D.Modified:
      D.Post;
    ELSE
      D.Cancel;
    FI;
    Q.Next;
  End;
end;
```

-- А ТЕПЕРЬ, КАК НАДО ДЕЛАТЬ

```
PROC ДокументEdit(D:Документ);
  IF D.State = 0 THEN
    D.Edit;
  END;
END;

PROC ОбработатьДокумент(D:Документы.Движение_Ресурсов_И_Обязательств);
  Var ТипСоб:Integer;

  ТипСоб = D.ТипСоб;
  IF ((ТипСоб=4 ИЛИ ТипСоб=16) И D.МыИ И D.Источник<>D.Получатель) THEN
    -- Отгрузка контрагенту
    ДокументEdit(D);
    D.Номер = 10;
  ELSIF ТипСоб=11 THEN
    -- Списание,
    -- изменений не нужно
  FI;
END;

PROC ОбработатьДокументы(Q:Запрос);
  While НЕ Q.EOF do
    D=Q.Current;
    ОбработатьДокумент(D);

    IF (D.State = 1) OR (D.State = 2) THEN
```

```

        D.Post;
    FI;
    Q.Next;
End;
end;

```

В случае, если происходит последовательная обработка нескольких строк подтаблицы записи или нескольких полей одной строчки подтаблицы, то нужно завести локальную переменную соответствующего типа и работать через нее (или использовать оператор **with**).

Пример:

-- КАК НЕ НАДО ДЕЛАТЬ

```

IF НЕ D.Позиции[j].АвтоРасчет:
    D.Позиции[j].АвтоРасчет = Истина;
FI;
IF D.Позиции[j].КоличПоДок<>ДолжноБытьКол:
    D.Позиции[j].КоличПоДок = ДолжноБытьКол;
FI;
IF D.Позиции[j].СуммаУч<>ДолжноБытьСум:
    D.Позиции[j].СуммаУч = ДолжноБытьСум;
FI;

```

-- А ТЕПЕРЬ, КАК НАДО ДЕЛАТЬ

```

var vStruct :Structure;

vStruct = D.Позиции[j];
IF НЕ vStruct.АвтоРасчет:
    vStruct.АвтоРасчет = Истина;
FI;
IF vStruct.КоличПоДок<>ДолжноБытьКол:
    vStruct.КоличПоДок = ДолжноБытьКол;
FI;
IF vStruct.СуммаУч<>ДолжноБытьСум:
    .СуммаУч = ДолжноБытьСум;
FI;

```

Если модифицируется более одного поля документа, то необходимо до модификации вызвать **Edit**, а после **Post**.

Если необходимо модифицировать более одного документа, то необходимо заключить обработку в транзакции с помощью вызовов **BeginTransaction...EndTransaction**.

Если при анализе группы документов необходима непротиворечивость и устойчивость состояния (например, отчет на определенный момент по данным, с которыми в сетевом многопользовательском режиме работает несколько человек), необходимо пользоваться изоляциями.

При доступе к группе документов через объект класса **Запрос**, необходимо предварительно проанализировать, какие поля документов реально используются, и заполнить соответствующим образом свойства запроса **LoadingFieldsMode** и **LoadingFields** перед построением запроса.

Если в открытом запросе (объекте класса **Запрос**.) нужно изменить сразу несколько настроек, например, индекс, фильтр, то лучше сначала закрыть запрос, а после установок его открыть. Реализована возможность устанавливать Query.Current до построения запроса (Query.Select). Таким образом, если нужно открыть запрос и позиционировать его на определенную запись, то оптимальнее будет сначала позиционировать, а после открыть. Если данной записи нет, то запрос становится на ближайшую по значениям индексных полей.

Описание

```
ЗавершитьИзоляцию;  
EndIsolation;
```

Назначение

Процедура завершает внешнюю изоляцию, начатую ранее с помощью [BeginIsolation](#). Если вызовов **BeginIsolation** было несколько, **EndIsolation** завершает изоляцию по самому последнему из них. При этом автоматически определяются классы документов, по которым была начата данная изоляция.

Все внешние изменения, проведенные с документами после вызова **BeginIsolation**, попадают на локальное рабочее место только после вызова **EndIsolation**.

Пример

```
proc P1;  
  
    EndIsolation;  
    -- ошибка "нет активной внешней изоляции",  
    -- так как ранее отсутствуют вызовы BeginIsolation  
  
end;
```

Описание

```
ЗавершитьТранзакцию;  
EndTransaction;
```

Назначение

Процедура завершает транзакцию, начатую ранее с помощью [BeginTransaction](#). Если вызовов **BeginTransaction** было несколько, **EndTransaction** завершает транзакцию по самому последнему из них. При этом автоматически определяются классы документов, по которым была начата данная транзакция.

Все изменения, проведенные с документами после вызова **BeginTransaction**, отсылаются на сервер и попадают в информационную базу только после вызова **EndTransaction**.

Пример

```
proc P1;  
  
  BeginTransaction ([Накладная, Счет]);  
  try  
    -- здесь изменяем накладные и счета  
    -- ...  
    -- закончили изменять накладные и счета  
    EndTransaction; -- записали изменения в ИБ  
  except  
    AbortTransaction;  
    Raise;  
  end;  
  
end;
```

Описание

```
ИмпортЗаписей (Имя :Строка; {Режим : Импортер.ТипыРежимов}; {Опции : Импортер.Опции\[\]});  
  
RecordsImport (Name :String; {Mode : Importer.ConflictModeType}; {Options : Importer.Option  
\[\]});
```

Аргументы

Имя - имя файла, из которого осуществляется чтение;

Режим - одна из predefined констант, определяющая способ разрешения конфликтов.

Опция - необязательный параметр, массив [опций импорта](#). Если он не указан, то по умолчанию содержится один элемент [Importer.AutoImport] и открывается мастер импорта.

Если параметр задан и содержит один элемент [Importer.ShowWizard], то сразу запускается импорт. При задании двух элементов [Importer.ShowWizard, Importer.AutoImport] открывается мастер импорта и сразу запускается импорт. После успешного завершения импорта, мастер автоматически закрывается.

Назначение

Импортирует записи из предоставленных файлов TBC, TBD, XML.

Пример

```
ИмпортЗаписей(SystemPath(spProject, 'TSS')+'DEF_DAT\' +ИмяФайла, Импортер.Прерывать);  
-- Импорт прерывается  
  
RecordsImport(SystemPath(spProject, 'TSS') + 'DEF_DAT\' + ИмяФайла,  
Importer.Replace, [Importer.ShowWizard, Importer.AutoImport]);  
-- Вызов мастера и автоматический импорт.
```

См. также

[Класс Экспортер](#).

Описание

```
НачатьИзоляцию[(Записи[:Класс Документ]);  
BeginIsolation[(Records[:Class Document]);
```

Аргументы

Записи - это массив с перечислением классов документов (таблиц информационной базы), по которым начинается изоляция.

Назначение

Процедура начинает изоляцию указанных в массиве **Записи** классов документов. В массиве должны быть перечислены именно классы документов, а не ссылки на сами документы. Если массив опущен, изоляция открывается на все классы документов всех проектов, для которых была создана текущая информационная база.

После вызова данной процедуры все изменения, проведенные с документами указанных классов другими пользователями, не будут поступать на данное клиентское место вплоть до завершения изоляции с помощью процедуры [EndIsolation](#).

Пример

```
proc P1;  
  
  BeginTransaction([Накладная, Счет]);  
  try  
    BeginIsolation ([Счет]);  
    try  
      -- анализируем счета  
      -- ...  
    finally  
      EndIsolation;  
    end;  
    -- здесь изменяем накладные и счета  
    -- ...  
  EndTransaction; -- записали изменения в ИБ  
except  
  AbortTransaction;  
  Raise;  
end;  
  
end;
```

Описание

```
НачатьТранзакцию[ (Записи[]:Класс Документ)];  
BeginTransaction[ (Records[]:Class Document)];
```

Аргументы

Записи -это массив с перечислением классов документов (таблиц информационной базы), для работы с которыми начинается транзакция.

Назначение

Процедура начинает транзакцию для указанных в массиве **Записи** классов документов. В массиве должны быть перечислены именно классы документов, а не ссылки на сами документы. Если массив опущен, транзакция открывается на все классы документов всех проектов, для которых была создана текущая информационная база.

Все изменения, проведенные с документами указанных классов после вызова данной процедуры, не будут отосланы на сервер и не попадут в информационную базу до тех пор, пока не вызвана процедура [EndTransaction](#).

Пример

```
proc P1;  
var DD[] : Class Document;  
    DD[1] = Накладная;  
    DD[2] = Счет;  
  
    BeginTransaction(DD);  
    try  
        -- здесь изменяем накладные и счета  
        -- ...  
        -- закончили изменять накладные и счета  
        EndTransaction; -- записали изменения в ИБ  
    except  
        AbortTransaction;  
        Raise;  
    end;  
  
end;
```

ТБ.Скрипт допускает укороченную запись массива без явного описания. Поэтому вышеприведенный вызов BeginTransaction можно переписать следующим образом:

```
BeginTransaction([Накладная, Счет]);
```

а массив (в данном случае, DD) не описывать.

Описание

ОтменитьТранзакцию;
AbortTransaction;

Назначение

Процедура отменяет транзакцию, начатую ранее с помощью [BeginTransaction](#). Если вызовов **BeginTransaction** было несколько, **AbortTransaction** завершает транзакцию по самому последнему из них. При этом автоматически определяются классы документов, по которым была начата данная транзакция.

Все изменения, проведенные с документами после вызова **BeginTransaction**, отменяются, и локальная копия данных приводится в соответствие с информационной базой сервера.

Пример

```
proc P1;

  BeginTransaction([Накладная, Счет]);
  try
    -- ...
    EndTransaction;
  except -- была какая-то ошибка
    -- "откатываемся" до начального состояния
    AbortTransaction;
    Raise;
  end;

end;
```

Описание

```
УстановитьСообщениеПриБлокировке(СтрокаСообщения :Строка);  
SetLockMessage (MessageString :String);
```

Аргументы

СтрокаСообщения - параметр типа **Строка**, содержащий текст сообщения.

Назначение

С помощью данной процедуры можно вместо стандартного сообщения выводимое при неудачной блокировке записи, вывести другое.

Сообщение устанавливается (т.е. процедура выполняется) перед попыткой изменения записи.

Описание

```
ЭкспортЗаписей (Имя:Строка [; З:Запрос] [; Р:Логическое]);  
RecordsExport (Name:String [; Q:Query] [; R:Logical]);
```

Аргументы

Имя - имя файла, в который будет осуществляться запись;
З - выполненный запрос, содержащий набор экспортируемых записей;
Р - логическое выражение, определяющее, нужно ли проводить рекурсивный экспорт записей.

Назначение

Экспортирует записи, удовлетворяющие запросу, в указанный файл формата TBC, TBD или XML. Если задан рекурсивный экспорт (последний параметр равен TRUE), то экспортируются не только те записи, что попали в результат запроса, но и те, на которые есть ссылки из записей запроса. Например, если экспортируются накладные, в которых есть ссылки на карточки контрагентов, то рекурсивный экспорт будет сопровождаться сохранением не только накладных, но и связанных с ними данных о контрагентах. Уровень ссылок не ограничен. То есть, если бы в карточке контрагента были ссылки на карточки сотрудников, то сведения о сотрудниках также были бы экспортированы.

См. также

[Класс Экспортер](#).

Назначение и использование механизма транзакций

В целях сохранения целостности данных при их модификации ТБ.Скрипт поддерживает механизм транзакций. Реализуется он с помощью процедур [BeginTransaction](#), [EndTransaction](#) и [AbortTransaction](#) класса **Система** или же с помощью методов и свойств класса [Транзакция / Transaction](#). В соответствии со стандартным принципом работы транзакций все изменения, проведенные с записями заданных типов (классов) после начала транзакции (**BeginTransaction, Transaction.Create**), не будут отосланы на Сервер данных и не попадут в информационную базу до тех пор, пока не вызвана процедура, завершающая транзакцию (**EndTransaction, Transaction.Apply**). Если эта процедура по каким-либо причинам не вызвана вообще, изменения никогда не вступят в действие. Иными словами, транзакция определяется парой вызовов **BeginTransaction/EndTransaction** (или состоянием объекта класса Transaction) и включает в себе единый, логически неделимый набор операций редактирования, которые либо происходят все вместе, либо не происходят вообще (в случае непредвиденных ошибок).

Транзакции могут быть вложенными, то есть после первого вызова **BeginTransaction** может последовать второй, третий и т.д. (скорее всего, для других классов документов, но допускается - и для тех же), и только затем соответствующее число вызовов **EndTransaction**.

Кроме процедуры EndTransaction в ТБ.Скрипт имеется еще одна процедура, использующаяся в сочетании с BeginTransaction - это **AbortTransaction**, отменяющая произведенные с начала транзакции изменения. Точно также в дополнение к **Transaction.Apply** класс Transaction содержит процедуру **Cancel** для отмены изменений.

Для каждого класса документов (типов записей) система поддерживает счетчик начатых транзакций, который увеличивается на 1 при вызове BeginTransaction и уменьшается при вызове EndTransaction или AbortTransaction. Изменения принимаются только при обнулении этого счетчика. То же самое относится и к вызовам методов класса Transaction.

Транзакции рекомендуется применять в случаях, когда существует вероятность нарушения целостности данных за счет рассинхронизации информации в нескольких документах одного класса или документах разных классов. Например, если не использовать транзакции, то ошибки в данных возможны при нарушении связи с сервером или перебоев с питанием.

Назначение и использование механизма изоляций

Изоляция по смыслу противоположна транзакции. Транзакция откладывает передачу изменений, сделанных пользователем, на сервер. Изоляция откладывает прием изменений с сервера, то есть изменений, сделанных остальными пользователями.

Для работы с изоляциями используются две парные процедуры: [BeginIsolation](#) и [EndIsolation](#) или класс [Изоляция / Isolation](#). С момента начала изоляции и вплоть до ее завершения, информационная база "видится" с данного компьютера в том виде, в каком она была до изоляции. Это позволяет проводить анализ "моментального снимка" данных и ускоряет их обработку. *Если EndIsolation по каким-либо причинам не вызвана, клиент никогда не получит обновленные версии изолированных классов документов.*

Для изоляций, по аналогии с транзакциями, ведется стек вызовов пар процедур **BeginIsolation/EndIsolation** и счетчик изоляций по каждому классу документов. Счетчик увеличивается на 1 при вызове BeginIsolation и уменьшается при вызове EndIsolation. Изоляция конкретного класса документов завершается только при обнулении соответствующего счетчика.

Транзакции и изоляции можно применять одновременно.

Все процедуры по работе с транзакциями и изоляциями необходимо использовать внутри [блоков обработки исключительных ситуаций или блоков защиты ресурсов](#).

Описание

ВосстановитьФильтрЗаписей(МакроФильтр :Строка; Записи :Запись[] {; Разыменование : [Система.ТипыРазыменованияФильтра](#)}) :Строка;

ComposeRecordFilter(MacroFilter :String; Records:Record[] {; Designate : [System.FilterDesignateTypes](#)}) :String;

Аргументы

МакроФильтр - строковый параметр, содержащий условие макро-фильтра.

Записи - массив записей, связанных с макро-фильтром записи.

Разыменование - необязательный параметр, управляющий тем, как будет раскрываться макрос. Параметр может принимать одно из следующих значений (по умолчанию он равен byDocID):

byDocID - условием фильтра является значение из поля **DocID**;

byForeignKey - условием фильтра является значение из поля **ForeignKey**;

byDescription - условием фильтра является значение из описательного поля **RecordDescription**.

Назначение

Функция раскладывает макро-фильтр на строковый фильтр и делает обратное преобразование относительно функции [DecomposeRecordFilter](#).

Функция возвращает строку фильтра и раскрывает макрос в полный [DocID](#) (по умолчанию, если третий параметр не задан), в [ForeignKey](#) или в [описательное поле](#).

Пример

```
var МакроФильтр :String;  
var Записи      :Record[];  
var Фильтр      :String;
```

```
МакроФильтр = "Управление.Данные.Ресурс.Изготовитель = {%2}";  
Записи[1] = {Управление.Данные.Субъект:5};  
Записи[2] = {Управление.Данные.Субъект:10};  
Фильтр = ВосстановитьФильтрЗаписей(МакроФильтр, Записи, byDescription);
```

В результате условие фильтра принимает вид:

```
Фильтр = 'Управление.Данные.Ресурс.Изготовитель = АО "Изготовитель"'
```

Значение "{%2}" в переменной **МакроФильтр**, означает 2-й элемент массива **Записи**.

byDescription означает, что условие будет брать значение из поля **Имя**.

См. также [Функция РазложитьФильтрЗаписей](#).

Описание

```
РазложитьФильтрЗаписей (Фильтр :Строка; var Записи :Запись[]) :Строка;  
DecomposeRecordFilter(Filter :String; var Records :Record[]) :String;
```

Аргументы

Фильтр - условие фильтра, параметр типа **Строка**;
Записи - массив записей.

Назначение

Возвращает строку с макросами плюс и заполняет массив для разрешения этих макросов. Т.е. все ссылки на документы заменяются на макросы, а массив заполняется мягкими ссылками на них.

Пример

```
var Фильтр           :String;  
var МакроФильтр      :String;  
var СвязанныеЗаписи  :Record[];
```

```
Фильтр = "Управление.Данные.Ресурс.Изготовитель = {Управление.Данные.Субъект:5}";  
МакроФильтр = РазложитьФильтрЗаписей(Фильтр, СвязанныеЗаписи);
```











В результате получится:

```
МакроФильтр = "Управление.Данные.Ресурс.Изготовитель = {%1}"  
СвязанныеЗаписи[1] = {Управление.Данные.Субъект:5};
```

Значение "{%1}" в переменной **МакроФильтр**, означает 1-й элемент массива **СвязанныеЗаписи**.

См. также [Функция ВосстановитьФильтрЗаписей](#).

Для работы с массивами применяются следующие процедуры и функции из класса [Система](#):

-  [Процедура ВставитьВМассив / InsertInArray](#)
-  [Функция ДлинаМассива / LengthOfArray](#)
-  [Функция Макс / Max](#)
-  [Функция Мин / Min](#)
-  [Функция ПоискБлижайшегоВМассиве / SearchNearestInArray](#)
-  [Функция ПоискВМассиве / SearchInArray](#)
-  [Функция СледующийЭлементМассива / NextArrItem](#)
-  [Функция СуммаМассива / SumOfArray](#)
-  [Процедура УдалитьИзМассива / DeleteFromArray](#)
-  [Процедура УпорядочитьМассив / SortArray](#)

Описание

ВставитьВМассив (перем Массив[]:Вариант; Индекс: Целое; Элемент:Вариант);
InsertInArray(var Array[]:Variant; Index: Integer; Element:Variant);

Аргументы

Массив - одномерный массив значений произвольного типа (при вставке в многомерный массив размерности N необходимо определить N-1 индексов, определяющих тем самым одномерный подмассив).

Индекс - определяет позицию, куда нужно вставить элемент.

Элемент - непосредственно значение вставляемого элемента (в том числе, допускается использование подмассива с размерностью на 1 меньше чем у **Массива**).

Назначение

Вставляет указанный элемент в требуемую позицию в массиве.

Внимание! Индекс всегда указывается по первой размерности массива. Это означает, что если, например, массив двумерный, то вставить в него новую строку можно одним вызовом **ВставитьВМассив**, однако для вставки нового столбца потребуются несколько вызовов этой процедуры по числу строк в массиве. Иными словами, столбец вставляется путем вставки скалярных элементов в одну и ту же позицию каждой из строк.

Пример

```
Цены : Число[2]; -- двумерный массив цен, например
-- Цены[1,1] - закупочная цена товара 1
-- Цены[1,2] - стандартная цена товара 1
-- Цены[1,3] - оптовая цена товара 1
-- Цены[2,1] - закупочная цена товара 2
-- и так далее

-- вставляем новый товар в таблицу цен
ЦеныНаНовыйТовар : Число[]; -- одномерный массив

proc P1(Номер:Целое);
  -- по индексу Номер в массив Цены вставляется
  -- массив ЦеныНаНовыйТовар
  ВставитьВМассив (Цены,Номер,ЦеныНаНовыйТовар);
end;

-- вставляем новую цену для всех товаров в таблице цен
МелкооптоваяЦена : Число[]; -- для всех товаров

proc P2(Номер:Целое);
var i:Integer;
  for i=1..КоличествоТоваров do
    -- по индексу Номер во все подмассивы Цены[i]
    -- вставляется соответствующая МелкооптоваяЦена[i]
    ВставитьВМассив (Цены[i],Номер,МелкооптоваяЦена[i]);
  end;
end;
```


Описание

УдалитьИзМассива (перем Элементы[:Вариант; Индекс: Целое);
DeleteFromArray(var Elements[:Variant; Index: Integer);

Аргументы

Элементы - одномерный массив значений произвольного типа.
Индекс -номер элемента, который требуется удалить.

Назначение

Удаляет элемент находящийся на позиции **Индекс**, из массива **Элементы**.

Пример

```
Массив : Строка[]; -- одномерный массив строк
proc P1(НомерУдаляемогоЭлемента:Целое);
  if Массив[НомерУдаляемогоЭлемента] <> "резерв" then
    УдалитьИзМассива(Массив, НомерУдаляемогоЭлемента);
  fi;
end;
```

Описание

```
УпорядочитьМассив(перем Элементы[:Вариант; [:Индексы[: Целое]);  
SortArray(var Elements[:Variant; [: Index[: Integer]);
```

Аргументы

Элементы - массив значений произвольного типа.

Индексы - необязательный массив, используется при сортировке элементов многомерного массива. В этом случае данный параметр содержит индексы элементов по всем размерностям, кроме одной сортируемой (тем самым определяется виртуальный одномерный подмассив).

Назначение

Процедура сортирует массив. Если элементами массива являются n-мерные массивы (а сам он, соответственно, - (n+1)-мерный массив), то в параметре **Индексы** должны быть заданы n индексов, определяющие, по какому элементу идет сравнение.

Пример

```
proc P1;  
  var M[] : Integer;  
  M[1] = 3;  
  M[2] = 2;  
  M[5] = -1;  
  -- элементы 3 и 4 не инициализированы, что эквивалентно 0  
  SortArray ( M );  
  -- получили массив M[1]=-1; M[2]=0; M[3]=0; M[4]=2; M[5]=3  
end;
```

Описание

ДлинаМассива(Элементы[]:Вариант): Целое;
LengthOfArray(Elements[]:Variant): Integer;

Аргументы

Элементы - одномерный массив значений произвольного типа.

Назначение

Возвращает количество элементов в массиве. Поскольку массивы в языке ТБ.Скрипт могут быть разреженными, фактически возвращается не число проинициализированных элементов, а разница между максимальным и минимальным индексом проинициализированных элементов, увеличенная на 1.

В случае многомерного [массива](#) функцию **LengthOfArray** следует использовать с учетом специфики внутреннего хранения массивов в языке ТБ.Скрипт. Пусть имеется трехмерный массив А. Тогда вызов:

- LengthOfArray(A) - возвращает длину массива по первой размерности;
- LengthOfArray(A[i]) - длину по второй размерности при выбранном i-том индексе по первой размерности (для каждого i длина по второй размерности может быть разной);
- LengthOfArray(A[i,j]) - длину по третьей размерности при выбранном i-том индексе по первой размерности и j-том - по второй.

Пример

```
proc P1;  
  var Дано[] : String;  
  var Y : Integer;  
  Дано[1] = "3";  
  Дано[3] = "2";  
  Дано[10] = "11";  
  Message( LengthOfArray ( Дано ) );  
  -- выведет 10  
end;
```

Описание

Макс(Элементы[:Вариант]): Вариант;
Max(Elements[:Variant): Variant;

Аргументы

Элементы - одномерный массив значений произвольного типа, из которых требуется выбрать максимальное.

Назначение

Возвращает максимальный из элементов массива. Правила сравнения конкретных значений определяет тип массива. В случае, когда используется массив Variant, все его элементы должны иметь один и тот же базовый тип. Иначе функция генерирует исключение.

Пример

```
proc P1;
  var Дано[] : Variant;
  var X : Numeric;
  var S : String;
  var Y : Integer;
  var Строки[] : String;
  Дано[1] = 3;
  Дано[2] = 2;
  Дано[3] = 11;
  X = Max ( Дано ); -- X = 11
  Дано[1] = "3";
  Дано[2] = "2";
  Дано[3] = "11";
  S = Max( Дано ); -- S = "3"
  Строки[1] = "3";
  Строки[2] = "2";
  Строки[3] = "11";
  S = Max( Строки ); -- S = "3"
  Дано[1] = 3;
  Дано[2] = "2";
  Дано[3] = 11;
  Y = Max( Дано ); -- ошибка на ошибке
end;
```

В конце данного примера присутствуют две неточности, способные вызвать ошибки. Во-первых, в массив элементов типа Variant были записаны значения разных типов (их нельзя сравнить). Во-вторых, возвращаемое значение присваивается переменной типа Integer, хотя должно - переменной того же типа, что и массив.

Описание

```
Мин(Элементы[:Вариант): Вариант;  
Min(Elements[:Variant): Variant;
```

Аргументы

Элементы - одномерный массив значений произвольного типа, из которых требуется выбрать минимальное.

Назначение

Возвращает минимальный из элементов массива. Правила сравнения конкретных значений определяет тип массива. В случае, когда используется массив Variant, все его элементы должны иметь один и тот же базовый тип. Иначе функция генерирует исключение.

Пример

```
proc P1;  
  var Дано[] : Variant;  
  var X : Numeric;  
  var S : String;  
  var Y : Integer;  
  var Строки[] : String;  
  Дано[1] = 3;  
  Дано[2] = 2;  
  Дано[3] = 11;  
  X = Min ( Дано ); -- X = 2  
  Дано[1] = "3";  
  Дано[2] = "2";  
  Дано[3] = "11";  
  S = Min( Дано ); -- S = "11"  
  Строки[1] = "3";  
  Строки[2] = "2";  
  Строки[3] = "11";  
  S = Min( Строки ); -- S = "11"  
  Дано[1] = 3;  
  Дано[2] = "2";  
  Дано[3] = 11;  
  Y = Min( Дано ); -- ошибка на ошибке  
end;
```

В конце данного примера присутствуют две неточности, способные вызвать ошибки. Во-первых, в массив элементов типа Variant были записаны значения разных типов (их нельзя сравнить). Во-вторых, возвращаемое значение присваивается переменной типа Integer, хотя должно - переменной того же типа, что и массив.

Описание

```
ПоискБлижайшегоВМассиве(Элементы :Вариант[]; Искомое :Вариант; перем Позиция :Integer
[;Индексы :Целое[]]) :Логическое;
SearchNearestInArray(Elements :Variant[]; Wanted :Variant; var Position :Integer [;
Index :Integer[]]) :Logical;
```

Аргументы

Элементы - отсортированный по возрастанию одномерный массив значений произвольного типа, среди которых требуется найти значение **Искомое**.

Искомое - искомое значение, тип параметра **Искомое** и элементов массива должен совпадать. Иначе функция генерирует исключение.

Позиция - функция заполняет переменную **Позиция** номером найденного элемента (если искомое найдено) или номером позиции, куда необходимо вставить ненайденное значение, чтобы массив остался отсортированным.

Индексы - необязательный массив, используется при поиске в многомерном массиве. В этом случае данный параметр содержит индексы элементов по дополнительным размерностям (не заданным через первый параметр вместе с именем массива), кроме одной, по которой проводится поиск (тем самым определяется виртуальный одномерный подмассив). Более подробно об индексации элементов в многомерных массивах см. раздел [ПоискВМассиве](#).

Назначение

Возвращает TRUE, если искомое было найдено. При этом номер найденного элемента помещается в переменную **Позиция**. Возвращает FALSE, если искомое не было найдено. При этом в переменную **Позиция** записывается позиция, в которую ненайденное значение должно быть вставлено, чтобы массив остался отсортированным. Правила сравнения конкретных значений определяет их тип. Если в массиве более одного элемента имеют искомое значение, используется позиция первого из них.

Следует иметь в виду, что для двумерных массивов первый индекс задает строку, а второй - столбец.

Пример

```
proc InsertIntoSortedArray;
  var Val, i : Integer;
  var Массив : Integer[2]; -- двумерный массив
  --   1  8
  --   2 16
  --   4 32
  Массив = [[1, 2, 4],[8, 16, 32]];
  Val = 3;
  -- поиск во 2-ом столбце
  if SearchNearestInArray (Массив[2], Val, i)
  then -- уже есть такой элемент
    return;
  else
    InsertInArray(Массив[2],i,Val);
  end;
  trace(Str(Массив));
  -- получили массив:
  --   1  3
  --   2  8
  --   4 16
  --   32
end;
```

Описание

```
ПоискВМассиве(Элементы :Вариант[]; Искомое :Вариант [;Индексы :Целое[]]  
[;Бинарный:Логическое]) :Целое;  
SearchInArray(Elements :Variant[]; Wanted :Variant [; Index :Integer[]] [;  
Binary:Logical]) :Integer;
```

Аргументы

Элементы - одномерный массив значений произвольного типа, среди которых требуется найти значение **Искомое**.

Искомое - искомое значение, тип параметра **Искомое** и элементов массива должен совпадать. Иначе функция генерирует исключение.

Индексы - необязательный массив, используется при поиске в многомерном массиве. В этом случае данный параметр содержит индексы элементов по дополнительным размерностям, кроме одной, по которой проводится поиск (тем самым определяется виртуальный одномерный подмассив).

Бинарный - определяет способ сортировки: если передается значение TRUE - используется быстрый поиск (массив предварительно должен быть отсортирован), в случае FALSE - последовательный перебор.

Если поиск осуществляется по третьему измерению четырехмерного массива, то индексы двух первых измерений указываются в квадратных скобках непосредственно при имени массива, а индекс по четвертому измерению - через параметр **Индексы**. При поиске по четвертому измерению, вместе с именем массива (в первом параметре функции) указываются индексы по первым трем измерениям, а параметр **Индексы** не используется. Вообще, параметр **Индексы** опускается при поиске по последнему измерению сколь угодно мерного массива.

Назначение

Возвращает индекс элемента, равного искомому значению. Если совпадения нет, возвращает -1. Правила сравнения конкретных значений определяет их тип.

Следует иметь в виду, что для двумерных массивов первый индекс задает строку, а второй - столбец.

Пример

```
proc P1;  
  var Дано[] : String; -- одномерный массив  
  var Y : Integer;  
  Дано[1] = "3";  
  Дано[2] = "2";  
  Дано[3] = "11";  
  Y = SearchInArray ( Дано, "11" ); -- Y = 3  
end;  
  
proc P2;  
  var Val, i : Integer;  
  var Массив : Integer[2]; -- двумерный массив  
  --      1 4  
  --      2 5  
  --      3 6  
  Массив = [[1, 2, 3],[4, 5, 6]];  
  -- поиск во 2-ом столбце  
  trace ('Ищем по Массив[2,?] - SearchInArray (Массив[2], Val)');  
  for Val = 1..6 do  
    i = SearchInArray (Массив[2], Val);  
    if i<>-1 then  
      trace('Найдено '+Str(Val)+  
        ' в элементе '+Str(i)+  
        ' - проверка '+Str(Массив[2,i]));  
    end;  
  end;  
  trace ('');  
  -- поиск во 2-ой строке  
  trace ('Ищем по Массив[?,2] - SearchInArray (Массив, Val, [2])');
```

```

for Val = 1..6 do
  i = SearchInArray (Массив, Val, [2]);
  if i<>-1 then
    trace('Найдено '+Str(Val)+
      ' в элементе '+Str(i)+
      ' - проверка '+Str(Массив[i,2]));
  end;
end;
end;

proc P3;
  var Val, i : Integer;
  var Массив : Integer[3]; -- трехмерный массив

  -- массив 3*2*2
  -- 3 элемента по X, т.е. в строке;
  -- 2 элемента по Y, т.е. в столбце;
  -- 2 элемента по Z, т.е. в глубину

  -- 1      2      3
  -- 7      8      9
  --
  -- 4      5      6
  -- 10     11     12

  Массив = [ [[1,7],[4,10]], [[2,8],[5,11]], [[3,9],[6,12]] ];

  -- поиск "в глубину" на пересечении 2-ой строки и 2-го столбца
  trace ('Ищем по Массив[2,2,?] - SearchInArray (Массив[2, 2], Val)');
  for Val = 1..12 do
    i = SearchInArray (Массив[2, 2], Val);
    if i<>-1 then
      trace('Найдено ' +Str(Val)+
        ' в элементе '+Str(i)+
        ' - проверка '+Str(Массив[2,2,i]));
    end;
  end;
  trace ('');
  -- поиск по 2-му столбцу "на глубине" 2
  trace ('Ищем по Массив[2,?,2] - SearchInArray (Массив[2], Val, [2])');
  for Val = 1..12 do
    i = SearchInArray (Массив[2], Val, [2]);
    if i<>-1 then
      trace('Найдено '+Str(Val)+
        ' в элементе '+Str(i)+
        ' - проверка '+Str(Массив[2,i,2]) );
    end;
  end;
  trace ('');
  -- поиск по 2-ой строке "на глубине" 2
  trace ('Ищем по Массив[?,2,2] - SearchInArray (Массив, Val, [2,2])');
  for Val = 1..12 do
    i = SearchInArray (Массив, Val, [2,2]);
    if i<>-1 then
      trace('Найдено '+Str(Val)+
        ' в элементе '+Str(i)+
        ' - проверка '+Str(Массив[i,2,2]) );
    end;
  end;
end;
end;

```


Описание

СледующийЭлементМассива(Массив :Вариант[]; Индекс :Целое) :Целое;
NextArrItem(Array :Variant[]; Index :Integer) :Integer;

Аргументы

Массив - массив с элементами произвольного типа;

Индекс - индекс элемента, после которого необходимо найти в массиве проинициализированный элемент.

Назначение

Функция позволяет последовательно выявить все проинициализированные элементы любого массива. Массивы хранятся в системе разреженно - в виде списков, в которых перечислены только те элементы массива, в которые были записаны данные. Остальные элементы при обращении к ним на чтение возвращают значение **nil (пусто)** . Быстрый доступ к следующему заполненному элементу с помощью данной функции исключает необходимость прямого перебора всех элементов массива.

Если через **Индекс** передается значение 0, функция возвращает индекс первого непустого элемента массива. Если после элемента с номером **Индекс** больше нет проинициализированных элементов, функция возвращает -1. Допустимо, если элемента с номером **Индекс** также нет в массиве.

Пример

```
proc TrackArray;
  var Arr1 : integer[];
  Arr1[1] = 1;
  Arr1[2] = 4;
  Arr1[5] = 25;
  trace( NextArrItem (Arr1, 0)); -- выводит 1
  trace( NextArrItem (Arr1, 1)); -- выводит 2
  trace( NextArrItem (Arr1, 2)); -- выводит 5
  trace( NextArrItem (Arr1, 3)); -- выводит 5
  trace( NextArrItem (Arr1, 4)); -- выводит 5
  trace( NextArrItem (Arr1, 5)); -- выводит -1
  trace( NextArrItem (Arr1, 100)); -- выводит -1
end;
```

Описание

```
СуммаМассива(Элементы:Вариант[] {; Размер:Целое} {; Индексы:Целое[]}): Число;  
SumOfArray(Elements:Variant[] {; Size:Integer} {; Indices:Integer[]): Numeric;
```

Аргументы

Элементы -одномерный массив числовых значений, которые необходимо просуммировать.

Размер - количество элементов массива, подлежащих суммированию. Если он опущен, суммируются все элементы массива.

Индексы - массив дополнительных индексов, который используется в том случае, если необходимо просуммировать часть элементов многомерного массива: с помощью этих индексов из многомерного массива выделяется виртуальный одномерный подмассив (см. примеры). Так, для трехмерного массива **SumOfArray**(Arr3Dim, , [J, K]) без учета сложностей, связанных с тем, что массивы ТБ.Скрипт являются разреженными (хранятся в виде списков), будет делать следующее:

```
for I = 1..LengthOfArray(Arr3Dim) do  
    Result = Result + Arr3Dim[I, J, K];  
end;
```

Назначение

Суммирует значения заданного числа начальных элементов переменной-массива и возвращает полученную сумму. Если тип параметра-массива не соответствует допустимому типу аргумента (целые или вещественные числа), функция генерирует исключение.

Пример

```
proc P1;  
    var Дано[] : Число;  
    Дано[1] = 3;  
    Дано[2] = 2;  
    Дано[3] = 1;  
    Дано = СуммаМассива ( Дано ); -- 6  
    Дано = СуммаМассива ( Дано, 2 ); -- 5  
    Дано = СуммаМассива ( Дано, 1 ); -- 3  
end;  
  
proc P2;  
    var Val : Numeric;  
    var Массив : Integer[2]; -- двумерный массив  
    --      1 2 3  
    --      4 5 6  
    Массив = [ [1,4], [2,5], [3,6] ];  
  
    -- сумма элементов 2-ой строки, т.е. Массив[2,i]  
    Val = SumOfArray(Массив[2]);  
    trace(Val); -- выведет 7 (2+5)  
  
    -- сумма элементов 2-ого столбца, т.е. Массив[i,2]  
    Val = SumOfArray(Массив, ,[2]);  
    trace(Val); -- выведет 15 (4+5+6)  
  
end;  
  
proc P3;  
    var Val : Numeric;  
    var Массив : Integer[3]; -- трехмерный массив  
  
    -- массив 3*2*2  
    -- 3 элемента по X, т.е. в строке;  
    -- 2 элемента по Y, т.е. в столбце;  
    -- 2 элемента по Z, т.е. в глубину
```

```

-- 1      2      3
-- 7      8      9
--
-- 4      5      6
-- 10     11     12

Массив = [ [[1,7],[4,10]], [[2,8],[5,11]], [[3,9],[6,12]] ];

-- сумма элементов колонки, уходящей "в глубину" на
-- пересечении 2-ой строки и 2-го столбца, т.е. Массив[2,2,i]
Val = SumOfArray(Массив[2,2]);
trace(Val); -- выведет 16 (5+11)














-- сумма элементов 2-го столбца во 2-ой плоскости "по глубине"
-- т.е. Массив[2,i,2]
Val = SumOfArray(Массив[2], , [2]);
trace(Val); -- выведет 19 (8+11)

-- сумма элементов 2-ой строки во 2-ой плоскости "на глубине"
-- т.е. Массив[i,2,2]
Val = SumOfArray(Массив, , [2,2]);
trace(Val); -- выведет 33 (10+11+12)

end;

```

Для работы с файловой системой используются следующие процедуры и функции из класса Система:

-  [Процедура ВзятьИнформациюОФайле / GetFileInfo](#)
-  [Функция ВзятьСписокФайлов / GetFileList](#)
-  [Функция ДатаИзмененияФайла / FileModifyDate](#)
-  [Функция ЕстьПапка / ExistFolder](#)
-  [Функция ЕстьФайл / ExistFile](#)
-  [Процедура КопироватьФайл / CopyFile](#)
-  [Процедура ПереместитьФайл / MoveFile](#)
-  [Процедура СинхронизироватьФайл / SynchronizeFile](#)
-  [Процедура СоздатьПапку / CreateFolder](#)
-  [Процедура УдалитьПапку / RemoveFolder](#)
-  [Процедура УдалитьФайл / RemoveFile](#)
-  [Процедура УстановитьАтрибутФайла / SetFileAttr](#)
-  [Процедура УстановитьДатуФайла / SetFileDate](#)

Описание

```
ПереместитьФайл(ИмяФайлаОткуда :Строка; ИмяФайлаКуда :Строка);  
MoveFile (SourceFileName :String; DestFileName :String);
```

Аргументы

ИмяФайлаОткуда - переменная содержащая исходное имя (путь) файла.

ИмяФайлаКуда - переменная содержащая новое имя (путь) файла.

Назначение

С помощью данной процедуры можно переместить и/или переименовать файл.

Пример

```
var ИмяФайлаОткуда :Строка;  
var ИмяФайлаКуда :Строка;
```

```
ИмяФайлаОткуда = 'c:\СтараяПапка\СтароеИмяФайла.txt'  
ИмяФайлаКуда = 'd:\НоваяПапка\НовоеИмяФайла.doc'  
ПереместитьФайл(ИмяФайлаОткуда, ИмяФайлаКуда);
```

В данном примере производятся две операции над файлом:

- 1 - Переименование файла;
- 2 - Перемещение файла.

Описание

ВзятьИнформациюОФайле (ИмяФайла :Строка; перем Размер :Целое; перем Дата :Дата; перем Атрибуты :Целое);
GetFileInfo (FileName :String; var Size :Integer; var ADate :Date; var Attributes :Integer);

Аргументы

ИмяФайла - строка с именем файла;
Размер - переменная, получающая значение размера файла;
Дата - переменная, получающая дату и время файла;
Атрибуты - переменная, получающая атрибуты файла.

Назначение

Позволяет получить сведения о размере, времени последней модификации и атрибутах указанного файла. Если имя файла задано без пути, программа просматривает локальный каталог информационной базы, которую использует текущий проект.

Атрибуты файла кодируются различными битами в параметре **Атрибуты**. Основными из них являются:

- 1 - только чтение;
- 2 - скрытый;
- 4 - системный;
- 16 - каталог;
- 32 - архивный.

На выходе из процедуры переменная **Атрибуты** может содержать произвольную суперпозицию этих величин.

Полный список атрибутов можно узнать из справки по Windows API.

Пример

```
proc Test(FileName :String);  
var Size :Integer;  
var ADate :Date;  
var Attr: Integer;  
  GetFileInfo(FileName, Size, ADate, Attr);  
  Message(FileName+' : '+Str(Size)+' ; '+Str(ADate)+' ; '+Str(Attr));  
end;
```

Описание

```
КопироватьФайл(ИмяФайлаОткуда :Строка; ИмяФайлаКуда :Строка);  
CopyFile(FileNameFrom :String; FileNameTo :String);
```

Аргументы

ИмяФайлаОткуда - строка с именем файла-источника;
ИмяФайлаКуда - строка с именем файла назначения.

Назначение

Копирует файл из одного места в другое. Если имя файла задано без пути, программа предполагает, что он находится в локальном каталоге информационной базы, которую использует текущий проект. Если файл назначения существует, то он перезаписывается указанным файлом-источником.

Пример

```
-- создаем резервную копию файла  
proc BackUp(FileName :String);  
var position :Integer;  
var BackupFileName :String;  
    position = Pos(".", FileName);  
    if position > 0 then  
        BackupFileName = SubStr(FileName, 1, position);  
        BackupFileName = BackupFileName + "bak";  
        CopyFile(FileName, BackupFileName);  
    end;  
end;
```

Описание

```
СоздатьПапку(Название:Строка);  
CreateFolder(Name:String);
```

Аргументы

Название - название папки, которую требуется создать. Можно указывать как полное имя, включая путь, так и краткое название папки. Когда имя папки задано без пути, программа создает ее в своем рабочем каталоге Bin, где расположены исполняемые модули (*.exe и *.dll).

Назначение

Создает указанную папку.

Пример

```
proc Pl;  
  if ( ЕстьПапка ("Дополнения") ) = false then  
    -- если папка не существует, создаем ее  
    СоздатьПапку ("Дополнения ");  
  end;  
end;
```


Описание

```
УдалитьФайл(ИмяФайла :Строка);  
RemoveFile(FileName :String);
```

Аргументы

ИмяФайла - строка с именем файла, подлежащего удалению.

Назначение

Удаляет указанный файл. Если имя файла задано без пути, программа предполагает, что он находится в локальном каталоге информационной базы, которую использует текущий проект.

Пример

```
-- Эмулируем операцию перемещения файла  
proc MoveFile(FileNameFrom :String; FileNameTo :String);  
  try  
    CopyFile(FileNameFrom, FileNameTo);  
    RemoveFile(FileNameFrom);  
  except  
  end;  
end;
```

Описание

```
УстановитьДатуФайла (Название :Строка; Дата :Дата);  
SetFileDate(Name :String; ADate :Date);
```

Аргументы

Название - строка с именем файла;

Дата - дата и время, которые необходимо установить для указанного файла.

Назначение

Изменяет дату последнего изменения указанного файла. Если имя файла задано без пути, программа просматривает локальный каталог информационной базы, которую использует текущий проект.

Пример

```
УстановитьДатуФайла ("Журнал.txt", Сегодня);
```

Описание

```
СинхронизироватьФайл(Имя :Строка; {Режим : Система.РежимСинхронизацииФайлов});  
SynchronizeFile (Name :String; {Mode : System.SynchronizeFileMode});
```

Аргументы

Имя - имя синхронизируемого файла.

Режим - параметр, который может принимать одно из следующих значений:

НаСервере - указанный файл обновляется на сервере данных;

НаКлиенте - указанный файл обновляется на клиенте, т.е. на локальном компьютере;

Автоматически - указанный файл обновляется автоматически, т.е. старый файл заменяется новым.

Назначение

Данная процедура синхронизирует файл в трех режимах.

Пример

```
var ИмяФайла : Строка;  
ИмяФайла =
```

```
СинхронизироватьФайл(ИмяФайла, НаСервере);
```

В данном примере, файл на сервере данных заменяется копией с локальной машины.

Описание

```
УдалитьПапку(ИмяПапки :Строка; Рекурсивно :Логическое);  
RemoveFolder (FolderName :String; Recursive :Logical);
```

Аргументы

ИмяПапки - переменная содержащая имя папки для удаления.

Рекурсивно - переменная содержащая значение логического типа. При передаче значения **False**, папка не удалится, если она содержит файлы или подпапки, и в таком случае генерируется исключение. При передаче значения **True**, папка удалится со всеми вложенными файлами и папками.

Назначение

Данная процедура позволяет удалить папку (директорию).

Пример

```
var УдаляемаяПапка :Строка;  
  
УдаляемаяПапка = 'd:\ПапкаСМусором';  
УдалитьПапку(УдаляемаяПапка, True);
```

Описание

```
УстановитьАтрибутФайла(ИмяФайла :Строка; Атрибут :Целое);  
SetFileAttr (FileName :String; Attrib :Integer);
```

Аргументы

ИмяФайла - имя файла, у которого необходимо изменить атрибуты;

Атрибут - атрибут файла. Ниже приведены возможные варианты:

- 0** - сбрасывает (обнуляет) все атрибуты файла;
- 1** - устанавливает атрибут "только чтение";
- 2** - устанавливает атрибут "скрытый";
- 3** - устанавливает атрибуты "только чтение" и "скрытый";
- 4** - устанавливает атрибут "системный";
- 5** - устанавливает атрибуты "только чтение" и "системный".

Назначение

Процедура устанавливает атрибут файла.

Пример

```
var ИмяФайла :Строка;  
  
ИмяФайла = 'С:\Папка\МойДокумент.doc';  
УстановитьАтрибутФайла(ИмяФайла, 0);  
УстановитьАтрибутФайла(ИмяФайла, 1);
```

В данном примере сначала все атрибуты файла сбрасываются, а затем устанавливается атрибут "только чтение".

Описание

```
ВзятьСписокФайлов(ИмяПапки :Строка; Маска :Строка [; Режимы :Система.РежимыПоискаФайлов  
[] ]) :Строка[];  
GetFileList(FolderName :String; Mask :String [; Mode :System.FindFileModes[] ]) :String[];
```

Аргументы

ИмяПапки - строка с именем папки, в которой осуществляется поиск файлов;

Режимы - строка с подстановочными символами '*' (произвольное количество любых символов, в том числе и ноль символов) и '?' (один любой символ); например, "*.txt" - для поиска всех текстовых файлов или "отчет??.*" - для поиска всех файлов с произвольными расширениями и с именем, начинающимся со строки "отчет", после которой обязательно следует еще 2 символа;

Маска - массив с предопределенными константами, задающими режим поиска; если явным образом не задана ни одна из констант режимов, то поиск ведётся рекурсивно с подкаталогами, причем в результаты поиска попадают и файлы, и папки; если указана одна из констант **ИскатьТолькоФайлы** или **ИскатьТолькоКаталоги**, то в результаты попадают, соответственно, либо только файлы, либо только каталоги; эти два флага взаимно исключают друг друга.

Назначение

Функция выполняет поиск заданных файлов и/или папок в указанной папке и возвращает имена найденных объектов файловой системы в массиве строк.

Пример

```
var FilesSection :TemplateSection;  
var Files :String[];  
  
-- выводим список всех файлов из каталога ИБ в секцию  
proc EnumFiles(FolderName :String);  
  Files = GetFileList(SystemPath(spInfBase), "*.*", [FindOnlyFiles]);  
  FilesSection.Count = LengthOfArray(Files);  
end;
```

Описание

```
ДатаИзмененияФайла (Название:Строка): Дата;  
FileModifyDate(Name:String): Date;
```

Аргументы

Название - название файла, дату последней модификации которого требуется узнать. Можно указывать как полное имя, включая путь, так и только название файла.

Назначение

Возвращает дату последнего изменения файла. Когда имя файла задано без пути, программа просматривает рабочий каталог информационной базы, которую использует текущий проект.

См. также процедуру [ВзятьИнформациюОФайле](#).

Пример

```
proc P1;  
var ДФ : Дата;  
  if (ЕстьФайл ("Testing.idb") ) then  
    ДФ = ДатаИзмененияФайла ("Testing.idb");  
  end;  
end;
```

Функция ЕстьПапка / ExistFolder

Описание

ЕстьПапка(Название:Строка): Логическое;
ExistFolder(Name:String): Logical;

Аргументы

Название - название папки, наличие которой требуется проверить. Можно указывать как полное имя, включая путь, так и только название папки.

Назначение

Возвращает значение True, если папка существует, и False, если - нет. Когда имя папки задано без пути, программа просматривает каталог Bin, где расположены исполняемые модули программы (*.exe и *.dll).

Пример

```
proc P1;  
  if ( ЕстьПапка ("Дополнения") ) = false then  
    -- если папка не существует, создаем ее  
    СоздатьПапку ("Дополнения ");  
  end;  
end;
```


Описание

```
ЕстьФайл(Название:Строка): Логическое;  
ExistFile(Name:String): Logical;
```

Аргументы

Название - название файла, наличие которого требуется проверить. Можно указывать как полное имя, включая путь, так и только название файла.

Назначение

Возвращает значение ИСТИНА, если файл существует, и ЛОЖЬ, если - нет. Когда имя файла задано без пути, программа просматривает рабочий каталог информационной базы, которую использует текущий проект.

Пример

```
proc P1;  
  if ( ЕстьФайл ("Testing.idb") ) then  
    -- если файл существует, открыть его на редактирование  
    ОткрытьРедактор ("Testing.idb");  
  end;  
end;
```

Для операций над строками используются следующие процедуры и функции из класса [Система](#):

-  [Функция Бол / Up](#)
-  [Функция Вставить / Insert](#)
-  [Функция ВСтр / ToStr](#)
-  [Функция ВыделитьЗначение / ExtractValue](#)
-  [Функция ВыделитьЗначения / ExtractValues](#)
-  [Функция ВыделитьСлова / ExtractWords](#)
-  [Функция ВыделитьСлово / ExtractWord](#)
-  [Функция ВыровнятьЛ / PadL](#)
-  [Функция ВыровнятьП / PadR](#)
-  [Функция ВыровнятьЦ / PadC](#)
-  [Функция Длина / Length](#)
-  [Функция Заменить / Replace](#)
-  [Процедура ИзСтр / FromStr](#)
-  [Функция Код / Ord](#)
-  [Функция КоличествоЗначений / ValuesCount](#)
-  [Функция КоличествоСлов / WordsCount](#)
-  [Функция Мал / Lo](#)
-  [Функция МаскироватьСимвол / MaskChar](#)
-  [Функция Отрезать / Trim](#)
-  [Функция ОтрезатьЛ / TrimL](#)
-  [Функция ОтрезатьП / TrimR](#)
-  [Функция Повтор / RepStr](#)
-  [Функция ПодСтр / SubStr](#)
-  [Функция Поз / Pos](#)
-  [Процедура РазбитьСтроку / SplitString](#)
-  [Функция РазмаскироватьСимвол / UnmaskChar](#)
-  [Функция Символ / Chr](#)
-  [Функция Сло / Words](#)
-  [Функция СловоВСтроке / WordInString](#)
-  [Функция Соотв / Match](#)
-  [Функция Стр / Str](#)
-  [Функция Удалить / Delete](#)

Описание

```
СБольшойБуквы(ИсходнаяСтрока :String) :String;  
Загл(ИсходнаяСтрока :String) :String;
```

Аргументы

ИсходнаяСтрока - произвольная строка, в которой первая буква должна быть заменена на заглавную букву.

Назначение

Функция возвращает исходную строку, в которой первая буква будет заглавной буквой.

Пример

```
proc Example;  
  var S : String;  
  S = "Баланс"  
  S = Загл(S); -- получили "Баланс"  
end;
```

Описание

```
ИзСтр(var Переменная :Вариант; Выражение :Строка);  
FromStr (var Variable :Variant; String :String);
```

Аргументы

Переменная - параметр передаваемый по ссылке, т.е. данной переменной будет присвоено значение после конвертации. Конвертируемое значение примет тип данного параметра.

Выражение - переменная типа **Строка** содержащая значение для конвертации.

Назначение

Данная процедура конвертирует значение из одного типа в другой.

Пример

```
var ЗначениеЦелое :Целое;  
var ЗначениеЧисло :Число;  
var ЗначениеДата :Дата;  
  
FromStr(ЗначениеЦелое, "123.52");  
-- ЗначениеЦелое = 123  
ИзСтр(ЗначениеЧисло, "123.52");  
-- ЗначениеЧисло = 123.52  
ИзСтр(ЗначениеДата, "18.11.2005");  
-- ЗначениеДата = 18.11.2005
```

См. также функцию [Стр](#).

Описание

```
МаскироватьСимвол(Строка :Строка; Символ :Строка) :Строка;  
MaskChar (String :String; Symbol :String) :String;
```

Аргументы

Строка - исходная строка, которую требуется замаскировать;
Символ - символ, который необходимо добавить в начало и конец исходной строки.

Назначение

Функция добавляет указанный символ в начало и конец строки, если в строке встречается заданный символ, то он дублируется.

Функция возвращает замаскированную строку. Если потребуется получить исходную строку, то можно воспользоваться функцией [РазмаскироватьСимвол](#).

Пример

```
var ИсхСтрока :Строка;  
var СтрокаСМаской :Строка;  
  
ИсхСтрока = 'ООО "Лютики"';  
СтрокаСМаской = MaskChar(ИсхСтрока, ' ');  
-- ' "ООО " "Лютики" " " '
```

Функция "Падеж"

Описание функции:

```
func Падеж synonym ВернутьСловоформу(  
    аОписаниеСловоформ :String;  
    аПадежЧисло :Integer = фсИменительныйЕдин;  
    аСБольшойБуквы :Logical = True;  
    аОписаниеСловоформПоУмолчанию :String = ''  
) :String;
```

Функция выделяет из строки в формате описания словоформ (строка с разделителями, в которой перечислены различные падежные формы одного существительного), передаваемой в параметре **аОписаниеСловоформ**, конкретную словоформу, заданную падежом и числом (единственным или множественным). Для указания требуемой словоформы в качестве параметра **аПадежЧисло** можно использовать следующие константы:

```
фсИменительныйЕдин  
фсИменительныйМнож  
фсРодительныйЕдин  
фсРодительныйМнож  
фсДательныйЕдин  
фсДательныйМнож  
фсВинительныйЕдин  
фсВинительныйМнож  
фсТворительныйЕдин  
фсТворительныйМнож  
фсПредложныйЕдин  
фсПредложныйМнож
```

Параметр **аСБольшойБуквы** указывает, следует ли делать первую букву возвращаемой словоформы заглавной (по умолчанию - следует). А в параметре **аОписаниеСловоформПоУмолчанию** в функцию можно передать дополнительное, "страховочное" описание словоформ, которое будет использовано, если в основном описании нет словоформы для требуемых падежа и числа.

Пример использования функции "Падеж" в формулах реквизитов:

```
Падеж(%1.Менеджер.СловоФормаДолжность, фсДательныйЕдин)
```

где %1 - ссылка на конкретный реквизит (например, "ПроцессОткуда").

Описание

```
РазбитьСтроку(Текст :Строка; {Разделители :Строка}; var Строки :Строка[]);  
SplitString (Text :String; {Delimiters :String}; var Strings :String[]);
```

Аргументы

Текст - исходный текст, который необходимо разбить на несколько строк (подстрок).

Разделители - передаваемые символы, которые делят текст на подстроки. Данный параметр может быть опущен, в этом случае в качестве разделителя используются такие символы, как "." (точка); ",", (запятая); ";" (точка с запятой); "**пробел**".

Строки - передаваемый параметр по ссылке типа **Строка[]**, который будет заполнен подстроками.

Назначение

Данная процедура разбивает переданную строку на несколько строк и помещает их в массив.

Пример

```
var РазбиваемаяСтрока :Строка;  
var Разделители       :Строка;  
var Строки             :Строка[ ];
```

```
РазбиваемаяСтрока = "Яблоко;Хурма.Апельсин_Лимон Груша"  
Разделители = " ;._"  
РазбитьСтроку( РазбиваемаяСтрока , Разделители , Строки );
```

В результате массив Строки[] будет заполнен следующими 5 элементами:

```
Строки[1]      -- Яблоко  
Строки[2]      -- Хурма  
Строки[3]      -- Апельсин  
Строки[4]      -- Лимон  
Строки[5]      -- Груша
```

Функция Бол / Ур

Описание

```
Бол(Текст:Строка):Строка;  
Ур(Text:String):String;
```

Аргументы

Текст - произвольная строка, которую требуется преобразовать к верхнему регистру.

Назначение

Переводит все символы в заданной строке в верхний регистр (заменяет все строчные буквы на прописные).

Пример

```
proc Example;  
  var S : String;  
  S = "Баланс"  
  S = Ур(S); -- получили "БАЛАНС"  
end;
```


Функция Вставить / Insert

Описание

```
Вставить(Текст1:Строка; Текст2:Строка; Где:Целое):Строка;  
Insert(Text1:String; Text2:String; Pos:Integer): String;
```

Аргументы

Текст1 - строка, которая вставляется в строку **Текст2**.

Текст2 - исходная строка, в которую вставляется строка **Текст1**.

Где - позиция, начиная с которой вставляется строка **Текст1**.

Назначение

Возвращает строку, созданную на базе исходной строки Текст2, в которую, начиная с позиции Где, вставлена строка из Текст1.

Пример

```
S1 : String = "двадцать один";  
proc Example;  
  Message(Insert(" плюс", S1, 9));  
  -- выдаст сообщение "двадцать плюс один"  
end;
```

Функция ВСтр / ToStr

Описание

```
ВСтр(Значение:Вариант):Строка;  
ToStr (Value :Variant) :String;
```

Аргументы

Значение - значение любого типа, которое требуется преобразовать в строку.

Назначение

Функция преобразует значение любого типа в строку. В отличие от функции [Стр / Str](#) она использует те же правила, по которым переменные сохраняются в DBT файле (например, строковые значения берутся в кавычки). Строка, возвращаемая этой функцией может использоваться для обратного преобразования в значение с помощью функции [ИзСтр / FromStr](#).

Пример

```
proc P1(Sender :Button);  
...  
  Message( "Сообщ3" + ToStr(SessionInfo.UserRecord));  
...  
end;
```

Функция ВыделитьЗначение / ExtractValue

Описание

ВыделитьЗначение(Текст:Строка; Номер:Целое [; Разделители: Строка]): Строка;
ExtractValue(Text:String ; Num:Integer [; Delim: String]): String;

Аргументы

Текст - строка, из которой необходимо выделить значение.

Номер - номер значения в тексте. Значения в тексте нумеруются, начиная с 1 и до общего их числа (**КоличествоЗначений**).

Разделители - необязательный аргумент, позволяет указать нестандартные разделители.

Назначение

Функция возвращает строку, содержащую фрагмент исходного текста - так называемое значение. Порядковый номер фрагмента соответствует значению второго параметра функции. Если указанный номер значения превышает количество значения в тексте, функция возвращает пустую строку.

Фрагментом считается непрерывная последовательность символов, отличных от символов-разделителей. В том числе, фрагментом считается и нулевая последовательность между двумя следующими друг за другом символами-разделителями. Если символ-разделитель встречается в самом начале текста, то перед ним также подразумевается одно пустое значение. По аналогии, если символ-разделитель встречается в самом конце текста, то после него подразумевается одно пустое значение.

По умолчанию разделителем считается любой символ, отличный от буквы, цифры и знака подчеркивания.

Пример

```
proc Example;
  var S1 : String;
  S1 = ExtractValue("раз, два три",3);
  -- S1 содержит "два", так как в тексте находятся
  -- такие значения: "раз", "", "два", "три"
  -- второе пустой значение лежит между символами ",", " и пробел
end;
```

Описание

```
ВыделитьЗначения(Текст:Строка; N1:Целое; КолВо:Целое [; Разделители: Строка]): Строка;  
ExtractValues(Text:String; N1:Integer; Count:Integer [; Delim: String]): String;
```

Аргументы

Текст - строка, из которой необходимо выделить **КолВо** значений , начиная со значения номер **N1**.

N1 - порядковый номер первого выделяемого значения.

КолВо - количество значений.

Разделители - необязательный аргумент, позволяет указать нестандартные разделители.

Назначение

Функция возвращает строку, содержащую несколько значений из исходного текста.

Значением считается непрерывная последовательность символов, отличных от символов-разделителей. В том числе, значением считается и нулевая последовательность между двумя следующими друг за другом символами-разделителями. Если символ-разделитель встречается в самом начале текста, то перед ним также подразумевается одно пустое значение. По аналогии, если символ-разделитель встречается в самом конце текста, то после него подразумевается одно пустое значение.

По умолчанию разделителем считается любой символ, отличный от буквы, цифры и знака подчеркивания.

Пример

```
proc Example;  
  var S1 : String;  
  S1 = ExtractValues("раз,,два,три,четыре",2,3);  
  -- S1 содержит "раз,,два", так как первое значение  
  -- подразумевается перед первой запятой, а третье  
  -- значение - между двумя смежными запятыми  
end;
```

Описание

```
ВыделитьСлова(Текст:Строка; N1:Целое; N2:Целое [; P: Строка]): Строка;  
ExtractWords(Text:String; N1:Integer; N2:Integer [; Delim: String]): String;
```

Аргументы

Текст - строка, из которой необходимо выделить несколько слов, начиная со слова номер N1 и длиной N2 слов.

N1 - порядковый номер слова, с которого начинается выделение слов.

N2 - количество выделяемых слов в тексте.

P - необязательный аргумент, позволяет указать нестандартные разделители слов.

Назначение

Функция возвращает строку, содержащую несколько слов из исходного текста, причем порядковый номер первого выделяемого слова соответствует значению параметра N1, а количество слов равно значению N2.

Словом считается непрерывная последовательность букв, цифр и знаков подчеркивания, отделенная от соседней такой последовательности символом-разделителем.

По умолчанию, если последний параметр не передается, разделителем считается любой символ, отличный от вышеупомянутых.

Пример

```
proc Example;  
  var S1 : String;  
  S1 = ExtractWords("раз два три четыре",2,2);  
  -- S1 содержит "два три"  
end;
```

Описание

```
ВыделитьСлово(Текст:Строка; Номер:Целое [; Р: Строка]): Строка;  
ExtractWord(Text:String ; Num:Integer [; Delim: String]): String;
```

Аргументы

Текст - строка, из которой необходимо выделить слово.

Номер - номер слова в тексте. Слова в строке нумеруются, начиная с 1.

Р - необязательный аргумент, позволяет указать нестандартные разделители слов.

Назначение

Функция возвращает строку, содержащую слово из исходного текста, причем порядковый номер этого слова соответствует значению второго параметра функции. Словом считается непрерывная последовательность букв, цифр и знаков подчеркивания, отделенная от соседней такой последовательности символом-разделителем (или несколькими разделителями).

По умолчанию, если последний параметр не передается, разделителем считается любой символ, отличный от вышеупомянутых. Если указанный номер слова превышает количество слов в исходной строке, функция возвращает пустую строку.

Пример

```
proc Example;  
  var S1 : String;  
  S1 = ExtractWord("раз, два три",3);  
  -- S1 содержит "три"  
end;
```

Описание

```
ВыровнятьЛ(Текст1:Строка; Ч: Целое; Текст2:Строка):Строка;  
PadL(Text1:String; Size:Integer; Text2:String): String;
```

Аргументы

Текст1 - произвольная строка, которую необходимо расширить до длины **Ч** символов, с выравниванием исходного текста по правому краю. Последний аргумент не используется.

Назначение

Возвращает строку, созданную на базе исходной строки **Текст1** путем добавления в ее начало (левую часть) пробелов, дополняющих ее длину до заданного значения Ч.

Пример

```
Message(PadL("слово", 11, " "));  
-- выдаст сообщение "      слово"
```

Описание

```
ВыровнятьП(Текст1:Строка; Ч: Целое; Текст2:Строка):Строка;  
PadR(Text1:String; Size:Integer; Text2:String): String;
```

Аргументы

Текст1 - произвольная строка, которую необходимо расширить до длины **Ч** символов, с выравниванием исходного текста по левому краю. Последний аргумент не используется.

Назначение

Возвращает строку, созданную на базе исходной строки **Текст1** путем добавления в ее конец (правую часть) пробелов, дополняющих ее длину до заданного значения **Ч**.

Пример

```
Message(PadR("слово", 11, " "));  
-- выдаст сообщение "слово      "
```


Описание

```
ВыровнятьЦ(Текст1:Строка; Ч: Целое; Текст2:Строка):Строка;  
PadC(Text1:String; Size:Integer; Text2:String): String;
```

Аргументы

Текст1 - произвольная строка, которую необходимо расширить до длины **Ч** символов, путем центровки. Последний аргумент не используется.

Назначение

Возвращает строку, созданную на базе исходной строки **Текст1** путем добавления в ее начало и конец пробелов, дополняющих ее длину до заданного значения **Ч**.

Пример

```
Message(PadC("слово", 11, " "));  
-- выдаст сообщение "    слово    "
```

Функция Длина / Length

Описание

```
Длина(Текст:Строка): Целое;  
Length(Text:String): Integer;
```

Аргументы

Текст - произвольная строка, длину которой требуется определить.

Назначение

Возвращает длину заданной строки.

Пример

```
S2 : String = "Взаиморасчет";  
proc Example;  
  var X : Integer;  
  X = Length(S2); -- X = 12  
end;
```

Функция Заменить / Replace

Описание

```
Заменить(Текст1:Строка; Текст2:Строка; Где:Целое):Строка;  
Replace(Text1:String; Text2:String; Pos:Integer): String;
```

Аргументы

Текст1 - строка, которая будет вставлена в исходную строку (**Текст2**), начиная с позиции **Где**.

Назначение

Возвращает строку, созданную на базе исходной строки **Текст2**, в которой все символы, начиная с позиции **Где**, заменены на строку из **Текст1**.

Если строка **Текст1** короче, чем длина фрагмента строки **Текст2** от позиции **Где** до ее конца, то результирующая строка обрезается. Если позиция **Где** превышает длину строки **Текст2**, то строка **Текст1** просто присоединяется в конец **Текст2**.

Пример

```
Message(Replace("два","двадцать один",10));  
-- выдаст сообщение "двадцать два"
```

Функция Код / Ord

Описание

Код(Символ:Строка): Целое;
Ord(Symbol:String): Integer;

Аргументы

Символ - произвольная строка. Обработаться будет только первый символ. Если строка нулевой длины, возвращается -1.

Назначение

Возвращает код первого символа в строке по кодовой таблице Windows.

Пример

```
proc Example;  
  var X : Integer;  
  X = Ord("я"); -- X = 255  
end;
```

Описание

```
КоличествоЗначений(Текст:Строка [; Разделители: Строка]): Целое;  
ValuesCount(Text:String [; Delimiters: String]): Integer;
```

Аргументы

Текст - строка, в которой необходимо подсчитать значения, т.е. пустые и непустые фрагменты строки (см. ниже), разделенные разделителями.

Разделители - необязательный аргумент, в котором можно указать символы-разделители, отличные от используемых по умолчанию.

Назначение

Функция подсчитывает количество значений в заданном тексте. Значением считается любая непрерывная последовательность символов (в том числе - последовательность нулевой длины), отличных от символов-разделителей. Если два символа разделителя встречаются в тексте один за другим, то считается, что между ними расположено пустое значение. Если символ-разделитель встречается в самом начале текста, то перед ним также подразумевается одно пустое значение. По аналогии, если символ-разделитель встречается в самом конце текста, то после него подразумевается одно пустое значение.

По умолчанию, когда второй параметр не передается, разделителем считается любой символ, отличный от буквы, цифры и знака подчеркивания.

Пример

```
proc Example;  
  var X : Integer;  
  X = ValuesCount(",800,600,"); -- X равно 4  
end;
```

Описание

КоличествоСлов(Текст:Строка [; Разделители: Строка]): Целое;
WordsCount(Text:String [; Delimiters: String]): Integer;

Аргументы

Текст - строка, в которой необходимо подсчитать слова.

Разделители - необязательный аргумент, в котором можно указать символы-разделители, отличные от используемых по умолчанию.

Назначение

Функция подсчитывает количество слов в заданном тексте. Словом считается непрерывная последовательность букв, цифр и знаков подчеркивания, отделенная от соседней такой последовательности символом-разделителем. По умолчанию, когда второй параметр не передается, разделителем считается любой символ, отличный от вышеупомянутых.

Пример

```
Пример
proc Example;
  var N : Integer;
  N = WordsCount("Налоговая декларация", " ");
  -- N равно 2
end;
```

Описание

```
Мал(Текст:Строка):Строка;  
Lo(Text:String):String;
```

Аргументы

Текст - произвольная строка, которую требуется преобразовать к нижнему регистру.

Назначение

Переводит все символы в заданной строке в нижний регистр (заменяет все прописные буквы на строчные).

Пример

```
proc Example;  
  var S : String;  
  S = "Баланс"  
  S = Lo(S); -- получили "баланс"  
end;
```

Функция Отрезать / Trim

Описание

```
Отрезать(Текст:Строка): Строка;  
Trim(Text:String): String;
```

Аргументы

Текст - произвольная строка, из которой необходимо убрать лидирующие и замыкающие пробелы и знаки табуляции.

Назначение

Возвращает строку, содержащую текст из исходной строки, но без лидирующих и замыкающих пробелов и знаков табуляции, если такие были.

Пример

```
S1 : String = "  двадцать один  ";  
proc Example;  
  Message(Trim(S1)); -- выдаст сообщение "двадцать один"  
end;
```


Описание

ОтрезатьЛ(Текст:Строка): Строка;
TrimL(Text:String): String;

Аргументы

Текст - произвольная строка, из которой необходимо убрать лидирующие пробелы и знаки табуляции.

Назначение

Возвращает строку, содержащую текст из исходной строки, но без лидирующих пробелов и знаков табуляции, если такие были.

Пример

```
S1 : String = " двадцать один ";  
proc Example;  
  Message(TrimL(S1)); -- выдаст сообщение "двадцать один "  
end;
```

Описание

```
ОтрезатьП(Текст:Строка): Строка;  
TrimR(Text:String): String;
```

Аргументы

Текст - произвольная строка, из которой необходимо убрать замыкающие пробелы и знаки табуляции.

Назначение

Возвращает строку, содержащую текст из исходной строки, но без замыкающих пробелов и знаков табуляции, если такие были.

Пример

```
S1 : String = " двадцать один ";  
proc Example;  
  Message(TrimR(S1)); -- выдаст сообщение " двадцать один"  
end;
```

Функция Повтор / RepStr

Описание

Повтор(Текст:Строка; ЧислоПовт: Целое): Строка;
RepStr(Text:String; Repeat: Integer): String;

Аргументы

Текст - произвольная строка.

ЧислоПовт - определяет, сколько раз требуется повторить эту строку.

Назначение

Возвращает новую строку, созданную из исходной путем ее повторения указанное число раз. Когда **ЧислоПовт** неположительно, возвращается пустая строка. Когда **ЧислоПовт** равно 1, результирующая строка является копией исходной.

Пример

```
S2 : String = "Привет,";  
proc Example;  
  Message(RepStr(S2,2));  
  -- выведет сообщение "Привет,Привет,"  
end;
```

Описание

ПодСтр(Текст:Строка; Начало:Целое; Сколько:Целое):Строка;
SubStr(Text:String; From:Integer; Num:Integer):String;

Аргументы

Текст - строка, из которой необходимо выделить подстроку.

Начало и **Сколько** определяют, соответственно, начиная с какого символа и сколько символов следует выделить в подстроку. Символы в строке нумеруются, начиная с 1.

Назначение

Возвращает строку, выделенную из исходной строки, начиная с указанного символа и определенной длины. Символы строки нумеруются, начиная с 1.

Пример

```
Пример
S2 : String = "Взаиморасчет";
proc Example;
  var S1 : String;
  S1 = SubStr(S2,7,6); -- S1 = "расчет"
end;
```

Описание

```
Поз(Текст1:Строка; Текст2:Строка):Целое;  
Pos(Text1:String; Text2:String):Integer;
```

Аргументы

Текст1 - подстрока, которую необходимо найти в строке Текст2.

Текст2 - исходная строка.

Назначение

Функция возвращает позицию подстроки Текст1 от начала строки Текст2, если подстрока содержится в строке Текст2. В противном случае возвращается ноль. Позиции нумеруются, начиная с 1. Строки сравниваются с учетом регистра.

Пример

```
S1 : String = "расчет";  
S2 : String = "Взаиморасчет";  
proc Example;  
  X : Integer;  
  X = Pos(S1,S2); -- X = 7, S1 содержится в S2, начиная с 7 знака  
  X = Pos(S2,S1); -- X = 0, S2 не содержится в S1  
end;
```

Описание

```
РазмаскироватьСимвол(Строка :Строка; Символ :Строка) :Строка;  
UnmaskChar (String :String; Symbol :String) :String;
```

Аргументы

Строка - замаскированная строка, т.е. строка, сформированная функцией [МаскироватьСимвол](#), из которой необходимо удалить первый и последний символ;

Символ - символ, который должен быть удален с начала и конца замаскированной строки.

Назначение

Функция возвращает строку, из которой удален первый и последний символ, при условии, что они совпадают с указанным символом. Если символы различаются, возвращается пустая строка.

Функция выполняет действие, обратное функции [МаскироватьСимвол](#).

Пример

```
var СтрокаСМаской :Строка;  
var СтрокаБезМаски :Строка;  
  
СтрокаСМаской=' 'ООО "Лютики" ' '  
СтрокаБезМаской = UnmaskChar(СтрокаСМаской, ' ');  
-- 'ООО "Лютики" '
```

Функция Символ / Chr

Описание

Символ(Код:Целое): Строка;
Chr(Code:Integer): String;

Аргументы

Код - код символа (в кодировке Windows).

Назначение

Возвращает символ (строку единичной длины) по заданному коду.

Пример

```
proc Example;  
  var S1 : String;  
  S1 = Chr(255); -- строка "я"  
end;
```

Описание

Сло(Выражение:Число [, ЕдИзм1 :Строка] [, ЕдИзм2 :Строка] [, Точность :Целое]): Строка;
Words(Expression:Numeric [, Unit1 :String] [, Unit2 :String] [, Accuracy:Integer]): String;

Аргументы

Выражение - числовое выражение, значение которого необходимо представить в виде суммы прописью.
ЕдИзм1 - содержит заданное в специальном формате имя единицы измерения в трех формах для правильного составления словесного описания целой части числа.

ЕдИзм2 - содержит заданное в специальном формате имя единицы измерения в трех формах для правильного составления словесного описания дробной части числа.

Точность - задает количество знаков в дробной части числа, которые будут интерпретированы при формировании суммы прописью. Иными словами, данный параметр косвенно определяет, сколько разменных единиц измерения (например, копеек) содержится в основной единице измерения (например, рубле). Если данный параметр опущен, он считается равным 2.

Если **ЕдИзм1** и **ЕдИзм2** не заданы, то по умолчанию сумма считается заданной в рублях (целая часть числа) и копейках (дробная часть числа);

Назначение

Функция интерпретирует заданное числовое выражение как сумму в рублях (по умолчанию) или в указанных с помощью необязательных параметров единицах измерения и возвращает строку, в которой эта сумма написана прописью (по умолчанию, на русском языке).

Формат записи параметров **ЕдИзм1** и **ЕдИзм2** следующий:

<основа>|<окончание1>|<окончание2>|<окончание3>:<род>

Род задается с помощью одного из следующих обозначений:

- **мужской** - 1, или М (русское), или M (английское);
- **женский** - 2, или Ж, или F;
- **средний** - 3, или С (русское), или N;

При составлении строки с суммой прописью программа берет основу названия единицы измерения и пристыковывает к ней одно из окончаний в зависимости от значения младшего разряда числительного:

- для 1 - это <окончание1>,
- для значений от 2 до 4 - <окончание2>,
- в остальных случаях - <окончание3>.

Любое из окончаний при необходимости может быть пустым.

Например:

```
"рубл|ь|я|ей:1"  
"копе|йка|йки|ек:2"  
"доллар|а|ов:1"
```

Подобные описания позволяют программе сгенерировать прописи, такие как:

```
"1 копейка", "2 копейки", "5 копеек"
```

или

```
"один доллар", "три доллара", "пятнадцать долларов"
```

Дробная часть числа записывается не прописью, а в виде целого, округленного (или дополненного нулями) до указанного (с помощью параметра **Точность**) числа знаков.

Если параметр **ЕдИзм2** опущен, то значение выражения округляется до целого и уже в таком виде преобразуется в сумму прописью.

Пример

```
прос P1;  
var Сумма : Строка;  
Сумма = Сло(25.211);  
-- Сумма = "Двадцать пять рублей 21 копейка"
```



```
Сло(123.55, "доллар||а|ов:1");  
-- Сумма = "сто двадцать четыре доллара",  
-- произошло округление с точностью до доллара  
Сло(123.55, "доллар||а|ов:1", "цент||а|ов:1");  
-- Сумма = "сто двадцать три доллара 55 центов"  
Сло(123.55, "килограмм||а|ов:1", "грамм||а|ов:1", 3);  
-- Сумма = "сто двадцать три килограмма 550 граммов"  
end;
```

Описание

```
СловоВСтроке(Слово:Строка; Текст:Строка [; Р: Строка]): Логическое;  
WordInString(Word:String; Text:String [; Delim: String]): Logical;
```

Аргументы

Слово - строка, содержащая слово.

Текст - строка, в которой необходимо провести поиск указанного слова.

- необязательный аргумент, позволяет указать нестандартные разделители слов.

Назначение

Функция возвращает значение ИСТИНА, если слово присутствует в тексте, и ЛОЖЬ в противном случае.

Словом считается непрерывная последовательность букв, цифр и знаков подчеркивания, отделенная от соседней такой последовательности символом-разделителем. По умолчанию, если последний параметр не передается, разделителем считается любой символ, отличный от вышеупомянутых.

Пример

```
proc Example;  
  var log : Logical;  
  log = WordInString("89346", "65436*67526 89346*67834", "*");  
  -- log = ЛОЖЬ, так как с учетом разделителя '*'  
  -- строка не содержит слова "89346"  
end;
```

Описание

Соотв(Текст: Строка; Маска: Строка): Логическое;
Match(Text: String; Mask: String): Logical;

Аргументы

Текст - произвольная строка, которую необходимо проверить на соответствие маске, заданной во втором аргументе Маска. В маске могут использоваться специальные подстановочные символы:

- ? - произвольный символ;
- * - последовательность любых символов произвольной длины, в том числе нулевой;
- ! - пустая подстрока либо подстрока, соответствующая маске ".*".

Проверить наличие символов '?', '*', '!' в строке не представляется возможным.

В том случае, если в качестве маски в функцию была передана строка без подстановочных символов, то функция выполняет строгое сравнение двух переданных ей строк (на полное соответствие) и возвращает ИСТИНА, если они одинаковы.

Назначение

Возвращает логическое значение ИСТИНА, если строка удовлетворяет заданной маске, и ЛОЖЬ - в противном случае.

Пример

```
Message(Str(Match("68.НДС", "68!"))); -- выдаст сообщение "Истина"
Match("Налог", "Налог*"); -- равно ИСТИНА
Match("Налоговые отчисления", "Налог*"); -- равно ИСТИНА
Match("Налог", "Налог?"); -- равно ЛОЖЬ, так как в соответствии
--с маской требуется еще 1 символ после строки "Налог"
Match("Налог", "На"); -- равно ЛОЖЬ, так как строки "Налог" и "На" разные
```

Описание

```
Стр(Что:Вариант [; Как:Вариант]):Строка;  
Str(What:Variant [; How: Variant]):String;
```

Аргументы

Что - значение любого типа, которое требуется преобразовать в строку.

Как - аргумент, зависящий от типа первого параметра.

Если через Что передается целое или действительное число, то второй параметр может быть целым числом, указывающим количество знаков после запятой. Если второй параметр положителен, то он задает количество знаков в дробной части результирующего числа после десятичной точки. При отрицательном значении параметра округляются разряды целой части числа. Во всех остальных случаях, т.е. когда преобразуемое значение имеет тип, отличный от целого или вещественного, через второй аргумент должна передаваться строка [форматного преобразования](#). При вызове функции второй параметр можно вообще не указывать, тогда преобразование производится в соответствии с умолчаниями, действующими для каждого типа данных.

Функция позволяет обрабатывать значения произвольных типов, в том числе вариантных и объектных. В частности, если функции передан объект класса, производного от класса **Запись**, то результатом будет строка вида "{<Имя класса записи>:}", что совпадает со значением свойства [DocIDStr](#) класса **Запись**.

Назначение

Преобразует значение произвольного типа в строку.

Пример

```
Num : Numeric = 6743.23456;  
  
func Example: String;  
    var S : String;  
    S = Str(Num, 2); -- получили "6743.23"  
    Return S;  
end;
```

Функция Удалить / Delete

Описание

Удалить(Текст:Строка; Позиция:Целое; Сколько:Целое):Строка;
Delete(Text:String; Pos:Integer; Num: Integer): String;

Аргументы

Текст - произвольная строка, из которой необходимо убрать несколько символов.

Позиция - номер первого удаляемого символа.

Сколько - число удаляемых символов.

Назначение

Возвращает строку, созданную на базе исходной, но без последовательности символов, начиная с позиции номер Позиция и длиной в Сколько знаков.

Пример




```
S1 : String = "двадцать один";  
proc Example;  
  Message(Delete(S1,9,1)); -- выдаст сообщение "двадцатьодин"  
end;
```

Замечание

Когда данная функция вызывается из кода бланка-редактора картотеки, необходимо полностью указывать ее имя System.Удалить или System.Delete, так как объект Запись (Record) имеет одноименную процедуру и перекрывает глобальное описание.

Процедуры и функции управления исключительными ситуациями

Для управления исключительными ситуациями применяются следующие процедуры и функции из класса [Система](#):

-  [Процедура Откат / Abort](#)
-  [Функция УстОшибку / SetLastError](#)
-  [Функция Код ошибки / ErrorCode](#)

Описание

Откат;
Abort;

Назначение

Генерирует "мягкое" исключение, не выдающее сообщение об ошибке. По сути дела, такая исключительная ситуация не является исключением в полном смысле, а представляет собой стандартный способ передать управление в вызывающий фрагмент кода. Данной возможностью удобно пользоваться, когда в соответствии с логикой программы необходимо прервать исполнение текущей части алгоритма.

В результате вызова процедуры управление передается в вышестоящие обработчики исключительных ситуаций.

Пример

```
if ЕстьФайл(СистемныйПуть(спИнфБаза)+ "новый.sig") = 0 then  
  Abort;  
fi;
```

Описание

```
УстОшибку(Номер:Целое {; Текст:Строка} {; Заголовок:Строка} {; ТипИконки :  
Система.ТипИконки});  
SetError(Number:Integer {; Text:String} {; Caption:String} {; IconType: System.IconType});
```

Аргументы

Номер - определяет код ошибки;

Текст - произвольная строка, которая будет выведена в диалоге сообщения об ошибке. Если данный параметр опущен - формируется стандартное сообщение с указанием кода ошибки;

Заголовок - произвольная строка, которая будет выведена в качестве заголовка диалога. Если данный параметр опущен, используется заголовок "Ошибка приложения";

ТипИконки - необязательный параметр, который определяет [тип иконки](#) при выводе сообщения с информацией об ошибке. По умолчанию ТипИконки = тиОшибки.

Назначение

Генерирует исключение с заданным кодом **Номер**. Если исключение не "гасится" в последующих обработчиках исключительных ситуаций, на экран выводится сообщение об ошибке. Когда второй аргумент не задан, сообщение содержит номер ошибки, в противном случае - указанный текст. Код исключения можно узнать внутри блока [TRY](#) с помощью функции [ErrorCode](#).

Пример

```
if ЕстьФайл(СистемныйПуть(спИнфБаза)+ "новый.sig") = 0 then  
  Веер; -- сигнализируем ошибку  
  SetError(1001,"Требуется обновление версии");  
fi;
```


Описание

КодОшибки: Целое;
ErrorCode :Integer;

Прототип функции ErrorText

ТекстОшибки: Строка;
ErrorText :String;

Назначение

Функции эти предназначены для получения информации о текущей ошибке.

Замечание

Данные функции могут использоваться только в разделе ИСКЛЮЧЕНИЕ..КОНЕЦ [блока обработки исключительных ситуаций](#). Важно, что функция КодОшибки может выдавать 0 для ошибок, не имеющих номера или контекста помощи.

Ошибки могут быть сгенерированы программным способом с помощью процедуры [УстОшибку / SetError](#) класса Система.

Описание

ИнфБазы :ИнфБазы;
BaseInfo :BaseInfo;

Назначение

Свойство возвращает ссылку на объект класса [ИнфБазы/BaseInfo](#).

Поле доступно только на чтение.

Пример

```
proc P1(Sender :Button);  
...  
  Message( "Имя инф.базы= " + BaseInfo.Name );  
...  
end;
```

Описание

```
ИнфСессии :ИнфСессии;  
SessionInfo :SessionInfo;
```

Назначение

Свойство возвращает ссылку на объект класса [ИнфСессии/SessionInfo](#).

Поле доступно только на чтение.

Пример




```
proc P1(Sender :Button);  
  ...  
  Message(SessionInfo.SharedFolder);  
  -- в сообщении выдается полный путь к каталогу Shared  
  ....  
end;
```

В класс **Консоль / Console** входят следующие группы процедур и функций:





- [Системные диалоги](#)
- [Работа с окнами](#)
- [Работы с отчетами](#)
- [Работа с буфером обмена](#)
- [Отладочные процедуры](#)
- [Прочие процедуры](#)

и перечислимые типы, описанные в теме: ["Константы"](#).

В классе **Консоль / Console** определены следующие перечислимые типы

-  [ВидыПравовойПоддержки / LegalSupportKinds](#)
-  [ТипыСобытийЖурнала/LogEventTypes](#)
-  [ТипПомощи/HelpType](#)

и константы:

-  [Константа кмдВерно / cmOk](#)
-  [Константа кмдДа / cmYes](#)
-  [Константа кмдНет / cmNo](#)
-  [Константа кмдОтказ / cmCancel](#)

Константы видов правовой поддержки

Для работы функции [ТипПравовойПоддержки](#) используются следующие константы, определенные в перечисляемом типе **ВидыПравовойПоддержки / LegalSupportKinds**:

- **ппРеферент / IsReferent** - используется информационная правовая система "Референт"
- **ппГарант / IsGarant** - используется информационная правовая система "Гарант"

Для работы процедуры [КонтекстнаяСправка](#) используются константы, определенные в перечисляемом типе **ТипПомощи / HelpType**. Они описывают суть действия, выполняемого справочной системой Windows. Большинство таких команд требуют указания дополнительного параметра – его смысл приводится в следующей таблице, вместе с назначением констант.

- **HELP_CONTEXT** - вызов окна справочной системы с темой, идентификатор которой задан параметром. В качестве параметра используется идентификатор (неотрицательное целое) темы, которую следует открыть;
- **HELP_QUIT** - закрывает окно справочной системы. Параметр игнорируется;
- **HELP_CONTENTS** - вызов окна справочной системы с главной темой (обычно это содержание.) Параметр игнорируется, так как идентификатор темы с содержанием записан в самом hlp-файле.
- **HELP_HELPONHELP** - вызов справки по пользованию справочной системой Windows. Параметр игнорируется;
- **HELP_SETCONTENTS** - временно изменяет идентификатор темы, используемой как содержание справки; эта тема выводится по нажатию кнопки Содержание в окне справочной системы. В качестве параметра используется идентификатор темы (неотрицательное целое). тема с данным идентификатором должна быть определена в hlp-файле;
- **HELP_CONTEXTPOPUP** - вызывает всплывающее окно с темой, идентификатор которой задан параметром; всплывающее окно автоматически закрывается по щелчку мыши вне его. В качестве параметра используется идентификатор (неотрицательное целое) темы, которую следует открыть;
- **HELP_FORCEFILE** - проверяет, открыт ли заданный hlp-файл в справочной системе; если нет, то открывает его. Параметр игнорируется.
- **HELP_FINDER** - открывает диалог справочной системы с оглавлением тем. Работает только в том случае, если вместе hlp-файлом поставляется одноименный cnt-файл; Параметр игнорируется.
- **HELP_KEY** - открывает тему, в которой указано ключевое слово, заданное с помощью параметра; если в hlp-файле несколько тем, в которых указано это слово, открывается диалог со списком найденных тем, из которого можно выбрать конкретную тему для просмотра. В качестве параметра используется строка, содержащая искомое ключевое слово. При сравнении строки с ключевыми словами ищется строгое соответствие;
- **HELP_COMMAND** - выполняет макрокоманду справочной системы. В качестве параметра используется строка с именем одной или более макрокоманд. Если макрокоманд несколько, их имена должны быть разделены точкой с запятой;
- **HELP_PARTIALKEY** - открывает тему, в которой указано ключевое слово, заданное с помощью параметра; если в hlp-файле несколько тем, в которых указано это слово, открывается диалог со списком найденных тем, из которого можно выбрать конкретную тему для просмотра. В качестве параметра используется строка, содержащая искомое ключевое слово. При сравнении строки с ключевыми словами ищется нестрогое соответствие. Если переданная строка пуста, выводится диалог справочной системы с полным перечнем ключевых слов.

Константы типов событий журнала (ТипыСобытийЖурнала/LogEventTypes)

Константы **ТипыСобытийЖурнала/LogEventTypes** используются в процедуре [ЗаписатьСобытие / LogEvent](#) для описания уровня важности добавляемого события:

- **тсИнформация / leInformation** - общая информация для справки;
- **тсПредупреждение / leWarning** - предупреждение о некритическом событии;
- **тсОшибка / leError** - запись о критическом событии.

Описание

кмдВерно ;
стOk ;

Назначение

Возвращает предопределенную константу **кмдВерно / стOk**, если была нажата кнопка **Выбор**, т.е. операция была проведена удачно.

Константа **кмдВерно / стOk** из класса **Консоль** используется в большинстве [функций](#) для интерактивного взаимодействия с пользователем.

Поле **кмдДа** / **смYes**

Описание

кмдДа;
смYes;

Назначение

Возвращает предопределенную константу **кмдДа** / **смYes**, если была нажата кнопка **Да**, т.е. операция была проведена удачно. Поле доступно только на чтение.

Константа **кмдДа** / **смYes** из класса **Консоль** используется в большинстве [функций](#) для интерактивного взаимодействия с пользователем.

Поле **кmdНет** / **cmNo**

Описание

кmdНет ;
cmNo ;

Назначение

Возвращает предопределенную константу **кmdНет** / **cmNo**, если была нажата кнопка **Нет**, т.е. операция не была проведена. Поле доступно только на чтение.

Константа **кmdНет** / **cmNo** из класса **Консоль** используется в большинстве [функций](#) для интерактивного взаимодействия с пользователем.

Описание

кмдОтказ ;
cmCancel ;

Назначение





Возвращает предопределенную константу **кмдОтказ / cmCancel**, если была нажата кнопка **Отмена** (операция не проведена). Поле доступно только на чтение.

Константа **кмдОтказ / cmCancel** из класса **Консоль** используется в большинстве [функций](#) для интерактивного взаимодействия с пользователем.

Пример

Отладочные функции и процедуры

Для проведения отладки используются следующие процедуры и функции из класса [Консоль](#):

-  [Функция TraceRef](#)
-  [ЗаписатьСобытие / LogEvent](#)
-  [ОчиститьТрассировку / ClearTrace](#)
-  [Процедура Трассировка / Trace](#)

Описание

ЗаписатьСобытие (Тип : [Консоль.ТипыСобытийЖурнала](#); Сообщение :Строка);
LogEvent (Type : [Console.LogEventTypes](#); Message :String);

Аргументы

Тип - тип добавляемого события;

Сообщение - строка с текстом, который будет записан в соответствующем параметре записи журнала событий (кроме того, среди параметров будут имя текущего пользователя, ip-адрес, компьютер и имя информационной базы).

Назначение

Данная процедура добавляет заданное строковое сообщение в качестве события указанного типа в системный журнал событий (доступный администратору).

Пример

```
LogEvent(leWarning, "Инициализация проекта проведена неполностью");
```

Описание

```
ОчиститьТрассировку;  
ClearTrace;
```

Назначение

Очищает окно сообщений (не следует путать его с диалогом, выдающим сообщение). В частности, в окне сообщений выводятся все сообщения (предупреждения и ошибки) компилятора ТБ.Скрипт.

Пример

```
proc P1;  
    ОчиститьТрассировку;  
end;
```

Описание

```
Трассировка(Информация:Вариант);  
Trace(Information:Variant);
```

Аргументы

Информация - значение произвольного типа, которое преобразуется в строковый тип и выводится в окне сообщений.

Назначение

Выводит указанную информацию в окне сообщений (не следует путать его с диалогом, выдающим сообщение). В частности, в окно сообщений попадают все сообщения (предупреждения и ошибки) компилятора ТБ.Скрипт. Процедура предназначена для отладочных целей, так как позволяет отследить последовательность выполнения участков кода.

Пример

```
проц P1;  
  Трассировка("Вход в процедуру P1, полет нормальный");  
  -- здесь собственно обработка данных  
  -- ...  
  Трассировка("Выход из процедуры P1, полет нормальный");
```


Функция TraceRef

Описание

```
TraceRef( {Класс :Класс} ) :Логическое;
```


Аргументы

Класс - необязательный параметр типа **Класс**, который необходимо трассировать.

Назначение

Данная функция трассирует список всех объектов определенного класса с указанием мест в соd-файлах, где они были созданы и возвращает значение логического типа.

К сервисным процедурам и функциям, определенным в классе [Консоль](#), относятся следующие:

-  Поле [Принтер / Printer](#)
-  Поле [Команды / Commands](#)
-  Поле [КоличествоПараметров / ParamCount](#)
-  Функ [СтрокаПараметров / ParamStr](#)
-  Проц [ВыполнитьКоманду / ExecuteCommand](#)
-  Функ [ВыполнитьПрограмму / ExecuteProgram](#)
-  Проц [ДобавитьИнфОСессии / AddSessionInfo](#)
-  Проц [ЗаккрытьПрог / CloseApp](#)
-  Проц [Звук / Веер](#)
-  Проц [КонтекстнаяСправка / HelpContext](#)
-  Функ [КнопкаНажата / KeyPressed](#)
-  Проц [Подсказка / Hint](#)
-  Проц [ПравоваяПоддержка / LegalSupport](#)
-  Проц [РазложитьЦвет / SplitColor](#)
-  Функ [ТипПравовойПоддержки / LegalSupportType](#)
-  Функ [ЧислоБитНаПиксел / BitsPerPixel](#)
-  Функ [ШиринаЭкрана / ScreenWidth](#)
-  Функ [ВысотаЭкрана / ScreenHeight](#)
-  Функ [СоздатьЖурнал / CreateJournal](#)
-  Функ [КоррекцияСсылок / CorrectRefs](#)
-  Проц [ПоказатьИсториюЗаписи / ShowRecordHistory](#)
-  Проц [УстановитьСообщениеПриБлокировке / SetLockMessage](#)

Коды клавиш, используемые в функции [КнопкаНажата](#).

Обозначение	Код	Клавиша/кнопка мыши
VK_LBUTTON	01	левая кнопка мыши
VK_RBUTTON	02	правая кнопка мыши
VK_MBUTTON	04	средняя кнопка мыши
VK_BACK 08	BACKSPACE	
VK_TAB 09	TAB	
VK_CLEAR	12	CLEAR
VK_RETURN	13	ENTER
VK_SHIFT	16	SHIFT
VK_CONTROL	17	CTRL
VK_MENU 18	ALT	
VK_PAUSE	19	PAUSE
VK_CAPITAL	20	CAPS LOCK
VK_ESCAPE	27	ESC
VK_SPACE	32	пробел
VK_PRIOR	33	PAGE UP
VK_NEXT 34	PAGE DOWN	
VK_END 35	END	
VK_HOME 36	HOME	
VK_LEFT 37	стрелка	влево
VK_UP 38	стрелка	вверх
VK_RIGHT	39	стрелка вправо
VK_DOWN 40	стрелка	вниз
VK_SNAPSHOT	44	PRINT SCREEN
VK_INSERT	45	INS
VK_DELETE	46	DEL
VK_HELP 47	HELP	
VK_0 48	0	
VK_1 49	1	
VK_2 50	2	
VK_3 51	3	
VK_4 52	4	
VK_5 53	5	
VK_6 54	6	
VK_7 55	7	
VK_8 56	8	
VK_9 57	9	
VK_A 65	A	
VK_B 66	B	
VK_C 67	C	
VK_D 68	D	
VK_E 69	E	
VK_F 70	F	
VK_G 71	G	
VK_H 72	H	
VK_I 73	I	
VK_J 74	J	
VK_K 75	K	
VK_L 76	L	
VK_M 77	M	
VK_N 78	N	
VK_O 79	O	
VK_P 80	P	
VK_Q 81	Q	
VK_R 82	R	
VK_S 83	S	
VK_T 84	T	
VK_U 85	U	
VK_V 86	V	
VK_W 87	W	

VK_X	88	X	
VK_Y	89	Y	
VK_Z	90	Z	
VK_LWIN	91	клавиша	Windows левая
VK_RWIN	92	клавиша	Windows правая
VK_APPS	93	клавиша	Приложений
VK_NUMPAD0	96		0 на доп. панели
VK_NUMPAD1	97		1 на доп. панели
VK_NUMPAD2	98		2 на доп. панели
VK_NUMPAD3	99		3 на доп. панели
VK_NUMPAD4	100		4 на доп. панели
VK_NUMPAD5	101		5 на доп. панели
VK_NUMPAD6	102		6 на доп. панели
VK_NUMPAD7	103		7 на доп. панели
VK_NUMPAD8	104		8 на доп. панели
VK_NUMPAD9	105		9 на доп. панели
VK_MULTIPLY	106		*
VK_ADD	107	+	
VK_SEPARATOR	108		Separator key (?)
VK_SUBTRACT	109	-	
VK_DECIMAL	110	.	
VK_DIVIDE	111	/	
VK_F1	112	F1	
VK_F2	113	F2	
VK_F3	114	F3	
VK_F4	115	F4	
VK_F5	116	F5	
VK_F6	117	F6	
VK_F7	118	F7	
VK_F8	119	F8	
VK_F9	120	F9	
VK_F10	121	F10	
VK_F11	122	F11	
VK_F12	123	F12	
VK_F13	124	F13	
VK_F14	125	F14	
VK_F15	126	F15	
VK_F16	127	F16	
VK_F17	128	F17	
VK_F18	129	F18	
VK_F19	130	F19	
VK_F20	131	F20	
VK_F21	132	F21	
VK_F22	133	F22	
VK_F23	134	F23	
VK_F24	135	F24	
VK_NUMLOCK	144		NUM LOCK
VK_SCROLL	145		SCROLL LOCK

Описание

КоличествоПараметров ;
ParamCount ;

Назначение

Свойство позволяет получить сведения о количестве параметров, заданных в командной строке запуска программы. Общий формат строки запуска программы и назначение параметров, используемых в командной строке, приведен в теме: ["Параметры запуска программы"](#).

Поле доступно только на чтение.

См. также [Функция СтрокаПараметров / ParamStr](#).

Поле Команды / **Commands**

Описание

Команды :ГруппаКоманд;
Commands :CommandGroup;

Назначение

Свойство позволяет получить ссылку на корневую группу команд. Поле доступно только на чтение.

Пример

```
Comm = Commands.CommandByName[ "Kernel.Account.Blanks" ] ;
```

Поле Принтер / Printer

Описание

```
поле Принтер : Принтер;  
field Printer : Printer;
```

Назначение

Возвращает ссылку на объект класса [Printer](#), который используется для управления печатью.

Свойство доступно только для чтения.

Пример

[Печать множества выделенных документов](#)

Описание

```
ВыполнитьКоманду(ИмяКоманды:Строка [; Отложено:Логическое]);  
ExecuteCommand(CommandName:String [; Delayed:Logical]);
```

Аргументы

ИмяКоманды - полное иерархическое имя команды, которое можно скопировать из поля **Команда** [на странице "Команды"](#) диалога "Настройки интерфейса". Для встроенных команд редактировать это поле запрещено. Диалог открывается командой [Сервис|Настройка интерфейса](#);

Отложено - необязательный параметр, определяет, следует ли выполнять команду немедленно в процессе выполнения данной процедуры или же поместить команду в очередь на выполнение. Немедленное выполнение обозначается через FALSE, отложенное - через TRUE. По умолчанию, если параметр **Отложено** опущен, предполагается немедленное выполнение команды, что эквивалентно значению FALSE.

Предупреждение. В полном имени команды в качестве имени группы первого уровня всегда указывается **имя проекта**. Для встроенных команд в качестве имени проекта используется ключевое слово **Kernel**, для пользовательских команд - **Custom**.

Назначение

Выполняет заданную команду немедленно или в отложенном режиме. Если указанная группа команд или команда не найдены среди существующих, генерируется исключение.

Отложенное выполнение необходимо в тех случаях, когда действие команды затрагивает выполняющийся класс (код). Например, нельзя немедленно выполнить команду **Заккрыть все**, поскольку она уничтожит все классы, в том числе и тот, что вызывает процедуру **ВыполнитьКоманду**.

Пример

```
proc ButtonPreviewClick (Sender :Button);  
  -- переключаем текущий бланк в режим предварительного просмотра  
  -- перед печатью  
  ExecuteCommand("Kernel.File.PrintPreview");  
end;
```


Описание

```
ДобавитьИнфоСессии (Страница :Строка; Сообщение :Строка);  
AddSessionInfo (Page :String; Message :String);
```

Аргументы

Страница - имя добавляемой страницы (закладки) в окне информации о сессии;
Сообщение - строка с текстом, который следует вывести на указанной странице.

Назначение

Данная процедура предоставляет возможность программно расширять диалог "Информация о сессии" собственными страницами с информацией о проекте. Для этого в классе **Profile** проекта надо ввести процедуру GetSessionInfo (без параметров), которая будет вызываться ядром системы при открытии окна информации о сессии (по команде [Учет|Информация](#)). Внутри процедуры GetSessionInfo и следует вызывать AddSessionInfo. AddSessionInfo имеет смысл только в теле GetSessionInfo.

Если страница с названием **Страница** уже существует, то строка с сообщением будет в неё добавлена.

Строка с текстом сообщения может содержать [макросы](#) оформления.

Пример

```
InClass Public  
  
proc GetSessionInfo;  
    AddSessionInfo("Версия проекта","Copyright (с) Вася Пупкин, 2002");  
    AddSessionInfo("Версия проекта", "Версия:"+ProjectInfo(piVersion,"ВасинПроект"));  
end;
```

Описание

```
ЗагрузитьИнтерфейснуюСхему(ИмяФайла :Строка);  
LoadInterfaceSchema(FileName :String);
```

Аргументы

ИмяФайла - имя файла интерфейсной схемы.

Назначение

Данная процедура загружает интерфейсную схему.

Пример

```
var ИнтерфейснаяСхема    :String;  
  
ИнтерфейснаяСхема = "ИнтерфейсныеСхемы\РядовойПользователь.shi"  
ЗагрузитьИнтерфейснуюСхему(ИнтерфейснаяСхема) ;  
...
```

Описание

```
ЗакрытьПрог;  
CloseApp;
```

Назначение

Выдает программе Студия команду на закрытие. В зависимости от текущего состояния программы (например, если есть открытые окна с отредактированным и несохраненным текстом), на экран могут выводиться запросы, адресованные пользователю, относительно того, нужно ли сохранять изменения. После этого, вне зависимости от того, какой вариант выбрал пользователь, Студия (вместе с выполняющимися проектами) будет закрыта.

В отладочной сессии вызов процедуры закрывает сессию, а не всю программу целиком.

В результате вызова **ЗакрытьПрог** пользователю никогда не выводится запрос на подтверждение закрытия программы, как это может быть сделано в ответ на попытку самого пользователя закрыть программу (при включении соответствующего флага в настройках программы).

Пример

```
-- стандартная обработка события "закрытие бланка"  
проц ПриЗакрытии(Режим:Logical);  
    Message("До новых встреч!");  
    CloseApp;  
end;
```

Процедура Звук / Веер

Описание

```
Звук([Индекс:Целое]);  
Веер([Index:Integer]);
```

Аргументы

Индекс - необязательный аргумент определяет, какой звук необходимо выдать. По умолчанию выдается стандартный звук Windows.

Назначение

Воспроизводит звук. Если задан **Индекс**, отличный от 0, компьютер должен быть оснащен соответствующим аппаратным и программным обеспечением (звуковая карта и/или драйвер).

Пример

```
if ЕстьФайл(СистемныйПуть(спИнфБаза)+ "новый.sig") = 0 then  
  Веер; -- сигнализируем ошибку  
  Message("Требуется обновление версии");  
fi;
```

Описание

КонтекстнаяСправка - (ИмяФайла:Строка; Команда:Консоль.ТипПомощи; Параметр:Вариант);
HelpContext - (FileName:String; Command:Console.HelpType; Parameter:Variant);

Аргументы

ИмяФайла - имя файла справки;

Команда - одна из [предопределенных констант](#), определяющих тип запрашиваемой справки;

Параметр - дополнительный параметр, содержащий информацию, необходимую для выполнения команды.

Назначение

Открывает указанный файл справки с помощью Справочной системы Windows и высвечивает требуемую тему или окно, в соответствии с командой и параметром.

Пример

```
InClass
HelpFileName: String = "d:...\hlp";
InObject
proc ShowHelpContext (ContextID :Integer);
var SubString :String;
    if HelpFileName = "" then
        Сообщение ("Нет справочной информации по данному контексту");
        exit;
    end
    --Удаляем кавычки, если они есть
    SubString = Подстр(HelpFileName, 1, 1);
    if (SubString = ' " ') or (SubString = " ' ") then
        HelpFileName = Подстр(HelpFileName, 2, Length(HelpFileName)-2);
    end;
    -- Открываем тему справки по её контекстному номеру
    КонтекстнаяСправка(HelpFileName, HELP_CONTEXT, ContextID);
end; -- ShowHelpContext
```

Описание

```
Подсказка(Информация:Вариант [; Процент:Целое] [; Всего:Целое]);  
Hint(Information:Variant [; Percent:Integer] [; Total:Integer]);
```

Аргументы

Информация - значение произвольного типа, которое преобразуется в строковый тип и выводится в строке состояния.

Процент -необязательный параметр, используется для того, чтобы отобразить графический индикатор прогресса (с заданным процентом выполнения) в правом углу строки состояния. Если параметр не используется, индикатор не выводится.

Всего - определяет величину, соответствующую выполнению всего процесса целиком. Если аргумент опущен, он полагается равным 100.

Таким образом, параметр **Процент** действительно содержит процент выполнения задачи только в том случае, если параметр **Всего** опущен или равен 100.

Назначение

Выводит указанную информацию в строку состояния и показывает индикатор прогресса. Процент заполнения индикатора рассчитывается по формуле: **Процент/Всего***100.

Пример

```
proc P1(Q :Query);  
var i: Integer;  
  
  Подсказка("Идет обработка данных...");  
  
  for i = 1..Q.Count do  
    -- здесь собственно обработка данных  
    -- ...  
    Hint("Идет обработка данных...",i,Q.Count);  
  end;  
  
  Hint("Обработка завершена",100);  
  
end;
```

Процедура ПоказатьИсториюЗаписи / ShowRecordHistory

Описание

```
ПоказатьИсториюЗаписи(ДляЗаписи :Запись);  
ShowRecordHistory(ForRecord :Record);
```

Аргументы

ДляЗаписи - запись, историю изменений которой требуется показать. Если ДляЗаписи = nil, будет сгенерирована ошибка "Значение ... не инициализировано".

Назначение

Процедура показывает историю изменений для записи, переданной в качестве аргумента.

Пример

```
proc ПоказатьИсториюИзменений(аПроцесс :Данные.интПроцесс;  
  Реквизит :Реквизиты.интРеквизит; Cell :TemplateCell; Index :Integer);  
  if (аПроцесс <> nil) and (аПроцесс.Record <> nil) then  
    ShowRecordHistory(аПроцесс.Record);  
  end;  
end;
```

Описание

```
ПравоваяПоддержка(Заголовок:Строка; ИмяФайла:Строка);  
LegalSupport(Title:String; FileName:String);
```

Аргументы

Заголовок - контекстное имя документа (темы), который необходимо показать;

ИмяФайла - имя файла со списком документов (необходимо указывать только для системы "Референт").

Назначение

Данная процедура открывает информационную правовую систему "Референт" или "Гарант" в зависимости от выбранных настроек.

Пример

```
ПравоваяПоддержка ("Информация о платежном поручении", "PLAT");
```


Описание

```
РазложитьЦвет(Цвет :Целое; var Красный :Целое; var Зеленый :Целое; var Синий :Целое);  
SplitColor (Color :Integer; var Red :Integer; var Green :Integer; var Blue :Integer);
```

Аргументы

Цвет - переменная типа **Целое** содержащий код цвета, который надо разложить;
Красный - переменная типа **Целое**, которой будет присвоен код насыщенности красного цвета;
Зеленый - переменная типа **Целое**, которой будет присвоен код насыщенности зеленого цвета;
Синий - переменная типа **Целое**, которой будет присвоен код насыщенности синего цвета.

Назначение

Процедура разложения цвета на 3 основных цвета: **красный**; **зеленый**; **синий**.

Пример

```
var КрасныйЦвет :Integer;  
var ЗеленыйЦвет :Integer;  
var СинийЦвет   :Integer;  
  
РазложитьЦвет(123456, КрасныйЦвет, ЗеленыйЦвет, СинийЦвет);
```

Описание

```
УстановитьСообщениеПриБлокировке(СтрокаСообщения :Строка);  
SetLockMessage(MessageString :String);
```

Аргументы

СтрокаСообщения - текст сообщения, свидетельствующий о блокировке записи. Если задана пустая строка, то выдается сообщение по умолчанию.

Назначение

Выдает текст сообщения, появляющийся на данном клиенте в диалоге, сообщающем о блокировке записи.

Пример

```
SetLockMessage('');  
-- сообщение по умолчанию.
```

Описание

```
ВзятьДатуЗакрытия(ОбластьУчета :Строка) :Дата;  
GetCloseDate (Domain :String) :Date;
```

Аргументы

ОбластьУчета - идентификатор [области учета](#);

Назначение

Возвращает дату закрытия периода для заданной области учета.

Пример

```
proc ДатаЗакрытия(Sender :Button);  
  var Дата3 : Date;  
  SetCloseDate("БухгалтерскийУчет", Now, True);  
  Дата3 = ВзятьДатуЗакрытия("БухгалтерскийУчет");  
  Message("Дата закрытия периода" + Str(Дата3));  
end;
```

Описание

ВыполнитьПрограмму(Файл:Строка; {ЖдатьЗавершения :Логическое}; {var КодЗавершения :Целое};
{ПоказыватьОкно :Логическое}):Целое;

ExecuteProgram(File:String; {WaitFor :Logical}; {var ExitCode :Integer};
{ShowWindow :Logical}):Integer;

Аргументы

Файл - название исполняемого файла, который требуется запустить, или название файла другого типа, который требуется загрузить в приложение, ассоциированное с данным типом.

Замечание. Можно указать как полный путь к файлу, так и одно название файла. В последнем случае программа просматривает каталоги компьютера, где по умолчанию следует искать файлы, для которых не задан полный путь (т.е. текущий рабочий каталог программы Студия, каталог Windows, системный каталог Windows, каталоги, упомянутые в переменной окружения PATH).

ЖдатьЗавершения - необязательный логический параметр, по умолчанию False, т.е. функция сразу после запуска внешней программы вернёт управление вызвавшей программе. Если значение равно True, то функция не вернёт управление в вызвавшую программу до тех пор, пока запущенная программа не завершит работу.

КодЗавершения - при ЖдатьЗавершения=False, параметр не имеет смысла, иначе - после завершения работы внешней программы указанная в этом параметре переменная получит код возврата внешней программы.

ПоказыватьОкно - необязательный логический параметр, по умолчанию True, т.е. окно запускаемой программы будет отображаться на экране. При значении ShowWindow=False окно не показывается, причём не будет и иконки в панели задач - приложение будет видно только в списке процессов Диспетчера Задач Windows.

Назначение

Запускает программу или загружает в соответствующую программу документ, заданный в первом аргументе. Возвращает 0, если программа успешно запущена, и код ошибки, который вернула система, если запуск не удался. В случае неудачного запуска на экран выдается сообщение.

Пример

```
proc P1;  
  -- запускаем MS Excel (если он установлен на компьютере)  
  -- и открываем в нем файл Квартальный отчет.xls  
  ExecuteProgram("c:\\Мои документы\\Квартальный отчет.xls");  
end;
```

В примере показано, как тип файла (в данном случае, XLS) определяет, какое именно приложение (для XLS - это, обычно, MS Excel), будет запущено.

Описание

ШиринаЭкрана: Целое;
ScreenHeight: Integer;

Назначение

Возвращает высоту экрана в пикселах.

Пример

```
if ШиринаЭкрана < 800 И ВысотаЭкрана < 600 then  
  Веер;  
  Сообщение("Рекомендуется использовать разрешение экрана"+  
    " 800 на 600 и выше ");  
fi;
```

Описание

КнопкаНажата (Код :Целое) :Логическое;
KeyPressed (KeyCode :Integer) :Logical;

Аргументы

Код - целочисленный код клавиши на клавиатуре или кнопки мыши; возможные варианты см. в [отдельном разделе](#).

Для получения кодов алфавитно-цифровых символов следует использовать выражение с использованием функции [Код / Ord](#): Ord('<символ>'), например, КнопкаНажата(Код('G')). При этом следует иметь в виду, что код клавиши одинаков для всех символов, которые с помощью данной клавиши получаются. Так, одна и та же клавиша 'G' в разных режимах генерирует символы 'G', 'g', 'П', 'п', однако идентификация клавиши (получение её кода) выполняется именно через заглавный символ английского алфавита (если речь идет о букве) или по символу, соответствующему этому же регистру (для остальных, небуквенных клавиш). Например, клавиша '4' идентифицируется по символу '4', а не '\$', расположенному на той же клавише.

Назначение

Возвращает TRUE, если указанная клавиша нажата в данный момент, FALSE - если клавиша не нажата.

Пример

```
-- если нажата клавиша 'Page Down'
if KeyPressed(VK_NEXT) then
    -- смещаемся на 20 строк вниз по шаблону
    ПрокрутитьВнизНа(20); -- прикладная функция
end;
```

Описание

```
КоррекцияСсылок(СтараяЗапись :Строка; НоваяЗапись :Строка; {ПоказыватьМастер :Логическое};  
{ПоказыватьСтатус :Логическое}) :Логическое;  
CorrectRefs(OldDocID:String; NewDocID :String; {ShowWizard :Logical;}  
{ShowStatus :Logical}) :Logical;
```

Аргументы

СтараяЗапись, НоваяЗапись - значение из поля **DocID** заменяемого и заменяющего документа соответственно;

ПоказыватьМастер - необязательный логический параметр. Если значение равно True (по умолчанию), то показывается [мастер коррекции ссылок](#);

ПоказыватьСтатус - необязательный логический параметр. По умолчанию значение равно True. В этом режиме в процессе коррекции ссылок в строке статуса будет выдано сообщение "Идёт коррекция ссылок...", иначе (False) - в статусной строке ничего не выводится.

Назначение

Функция производит коррекцию ссылочных полей и возвращает значение True, если была произведена хотя бы одна коррекция, иначе - False.

Описание

СоздатьЖурнал(ИмяЖурнала :Строка) : [ФормаКартотеки](#);
CreateJournal(JournalName :String) : [CardForm](#);

Аргументы

ИмяЖурнала - имя табличного или картотечного журнала.

Назначение

Функция по заданному имени журнала создает объект картотечного журнала, не показывая его на экране.

В случае, если журнал по заданному имени не найден, или найденный журнал не является табличным или картотечным, или для него не задана картотека, то будет сгенерирована исключительная ситуация и выдается сообщение об ошибке.

Пример

```
var CurJournal :Journal;  
var ФреймЖурналКонтейнер :TemplateFrame;  
var секЖурналы :TemplateSection;  
  
proc UpdateJournal;  
  if CurJournal <> nil then  
    ФреймЖурналКонтейнер.Clear;  
    if CurJournal.CardForm <> nil then  
      LoadForm(ФреймЖурналКонтейнер, CreateJournal(CurJournal.Name));  
    end;  
    секЖурналы.Cell[1, CurJourIndex].Contents = 'p';  
  end;  
end;
```


Описание

```
СтрокаПараметров(Индекс :Целое) :Строка;  
ParamStr(Index :Integer) :String;
```

Аргументы

Индекс - номер параметра в командной строке. Нумерация параметров начинается с единицы.

Назначение

Позволяет получить строковое представление параметра по его номеру в командной строке запуска программы. Количество параметров в командной строке определяется с помощью свойства [ParamCount](#).

Замечание. Параметры и имена файлов в командной строке запуска программы могут чередоваться и следовать в произвольном порядке. Общий формат строки запуска программы и назначение параметров, используемых в командной строке, приведен в теме: ["Параметры запуска программы"](#).

Пример

```
Message("Параметр = " + ParamStr(1));
```

Описание

```
ТипПравовойПоддержки:Целое;  
LegalSupportType:Integer;
```

Назначение

Возвращает предопределенный код, идентифицирующий тип правовой поддержки: ппРеферент или ппГарант.

Пример

```
if ТипПравовойПоддержки = ппГарант then  
  Message("Установлена система Гарант");  
fi;
```

Описание

УстановитьДатуЗакрытия(ОбластьУчета :Строка; Дата:Дата; [Показать :Логический]) :Логический;

SetCloseDate (Domain :String; Date :Date; [ShowWizard :Logical]) :Logical;

Аргументы

Область -идентификатор [области учета](#);

Дата - дата закрытия периода;

Показать - необязательный логический параметр, при значении "Истина" дата устанавливается с помощью диалога "Изменение даты закрытия", иначе - диалог не открывается.

Назначение

Возвращает значение "Истина", если для заданной области учета дата закрытия учетного периода установлена, иначе - "Ложь".

Пример

```
proc ДатаЗакрытия(Sender :Button);
  var ДатаЗ : Date;
  if SetCloseDate("БухгалтерскийУчет", Now, false):
    Message("Дата закрытия учетного периода успешно установлена");
end;
```

Описание

ЧислоБитНаПиксел: Целое;
BitsPerPixel: Integer;

Назначение

Возвращает количество битов, используемых для хранения цвета одной точки экрана в текущем видеорежиме. Допустимые значения: 4 (16 цветов), 8 (256 цветов), 16 (65536 цветов, High Color), 24 (16 млн. цветов, True Color), 32. Часть из них может быть недоступна из-за ограничений видеоплаты.

Пример

```
-- в зависимости от цветности видеорежима
-- загружаем подходящий значок проекта
if BitsPerPixel < 16 then
  LoadAppIcon("icon_poor.ico");
else
  LoadAppIcon("icon_rich.ico");
end;
```

Описание

ШиринаЭкрана: Целое;
ScreenWidth: Integer;

Назначение






Возвращает ширину экрана в пикселах.

Пример

```
if ШиринаЭкрана < 800 И ВысотаЭкрана < 600 then  
  Веер;  
  Сообщение("Рекомендуется использовать разрешение экрана"+  
    " 800 на 600 и выше ");  
fi;
```

Функции и процедуры для работы с буфером обмена

Для работы с буфером обмена предназначены следующие процедуры и функции из класса [Консоль](#):

-  [Функция ВзятьИзКармана / GetFromClipboard](#)
-  [Процедура ЗавершитьМодификациюКармана / ClipboardEndModify](#)
-  [Функция КарманСодержитДанные / ClipboardHasFormat](#)
-  [Процедура НачатьМодификациюКармана / ClipboardBeginModify](#)
-  [Процедура ПоместитьВКарман / PutToClipboard](#)

Описание

```
ПоместитьВКарман(Значение :Вариант {; Формат :Строка});  
PutToClipboard (Value :Variant {; Format :String});
```

Аргументы

Значение - значение, помещаемое в буфер обмена.

Формат - необязательный параметр, если он не указан, то процедура помещает передаваемое значение в буфер обмена в виде строки, внутреннего формата ТБ - IValue и специальных форматах. Если формат указан, то значения в буфер обмена *не будут записаны в формате строки*. Присвоив копируемому значению какой-либо формат, можно в дальнейшем интерпретировать его, т.е. выполнять какие-либо операции согласно присвоенному формату.

Внимание. Не рекомендуется использовать в названиях форматов общеупотребительные названия, чтобы избежать конфликта форматов с другими программами.

Назначение

Процедура помещает данные в буфер обмена. Последующие вызовы процедуры затирают данные, записанные ранее в буфер обмена (карман). Если требуется сохранить ранее записанные в буфер обмена данные, то воспользуйтесь процедурой [НачатьМодификациюКармана](#), которая используется только совместно с процедурой [ЗавершитьМодификациюКармана](#).

Пример

```
var ЗначениеДляПередачи :String;  
  
ЗначениеДляПередачи = "Передаваемая строка";  
ПоместитьВКарман(ЗначениеДляПередачи);
```

См. также функцию [ВзятьИзКармана](#).

Описание

```
ЗавершитьМодификациюКармана;  
ClipboardEndModify;
```

Назначение

Процедура завершает процесс помещение данных в буфер обмена, который был инициализирован процедурой [НачатьМодификациюКармана](#). Указанные процедуры являются парными и позволяют поместить в буфер обмена данные в разных форматах, не затирая их.

Рассматриваемые процедуры обладают эффектом накопления, т.е. допускается использовать вложенные друг в друга блоки ClipboardBeginModify - ClipboardEndModify. При этом следует следить за тем, чтобы каждому вызову ClipboardBeginModify обязательно соответствовал вызов ClipboardEndModify.

При обычной вставке данных в буфер обмена (карман) с помощью процедуры [ПоместитьВКарман](#) все ее последующие вызовы затирают данные предшествующих вызовов, независимо от того, какие форматы используются (одинаковые или разные).

Пример

```
ClipboardBeginModify;  
try  
    PutToClipboard(SomeValue);  
    PutToClipboard(SomeOtherValue, 'MyFormat1');  
    PutToClipboard(SomeOtherValue, 'MyFormat2');  
finally  
    ClipboardEndModify;  
end;
```

Внимание: в следующем примере -

```
ClipboardBeginModify;  
try  
    PutToClipboard(SomeValue);  
    PutToClipboard(SomeOtherValue);  
  
    PutToClipboard(SomeValue, 'MyFormat');  
    PutToClipboard(SomeOtherValue, 'MyFormat');  
  
finally  
    ClipboardEndModify;  
end;
```

SomeOtherValue затрёт в буфере значение SomeValue и в случае формата по умолчанию, и в случае заданного формата. Повторное помещение данных в любом формате затирает в буфере данные *этого же формата* даже в пределах блока ClipboardBeginModify /ClipboardEndModify.

Описание

НачатьМодификациюКармана;
ClipboardBeginModify;

Назначение

Данная процедура инициирует помещение данных в буфер обмена, а процедура [ЗавершитьМодификациюКармана](#) - завершает. Указанные процедуры являются парными и позволяют поместить в буфер обмена данные в разных форматах, не затирая их.

Рассматриваемые процедуры обладают эффектом накопления, т.е. допускается использовать вложенные друг в друга блоки ClipboardBeginModify - ClipboardEndModify. При этом следует следить за тем, чтобы каждому вызову ClipboardBeginModify обязательно соответствовал вызов ClipboardEndModify.

При обычной вставке данных в буфер обмена (карман) с помощью процедуры [ПоместитьВКарман](#) все ее последующие вызовы затирают данные предшествующих вызовов, независимо от того, какие форматы используются (одинаковые или разные).

Пример

```
ClipboardBeginModify;  
try  
    PutToClipboard(SomeValue);  
    PutToClipboard(SomeOtherValue, 'MyFormat1');  
    PutToClipboard(SomeOtherValue, 'MyFormat2');  
finally  
    ClipboardEndModify;  
end;
```

Внимание: в следующем примере -

```
ClipboardBeginModify;  
try  
    PutToClipboard(SomeValue);  
    PutToClipboard(SomeOtherValue);  
  
    PutToClipboard(SomeValue, 'MyFormat');  
    PutToClipboard(SomeOtherValue, 'MyFormat');  
  
finally  
    ClipboardEndModify;  
end;
```

SomeOtherValue затрёт в буфере значение SomeValue и в случае формата по умолчанию, и в случае заданного формата. Повторное помещение данных в любом формате затирает в буфере данные этого же формата даже в пределах блока ClipboardBeginModify /ClipboardEndModify.

Функция **ВзятьИзКармана** / **GetFromClipboard**

Описание

```
ВзятьИзКармана ({Формат :Строка }) :Вариант;  
GetFromClipboard ({Format :String}) :Variant;
```

Аргументы

Формат - необязательный параметр типа **Строка**, определяющий формат содержимого буфера обмена.

Назначение

Функция возвращает содержимое буфера обмена (кармана) и записывает его в переменную.

Если параметр **Формат** не указан, то функция вернет сохранённое в буфере значение либо в виде внутреннего формата ТБ - IValue, если такой формат есть, либо строку, если она есть в буфере, либо - пустой Variant.

Если формат задан, функция будет искать значения только в заданном формате (данные в строковом формате не ищутся), и вернёт пустой Variant, если в заданном формате данные в буфере обмена отсутствуют.

Пример

```
var СодержимоеБуфераОбмена :Variant;  
  
СодержимоеБуфераОбмена = ВзятьИзКармана();  
If (СодержимоеБуфераОбмена = nil) Then  
    Message("Буфер обмена пуст.");  
    Return;  
End;
```

См. также процедуру [ПоместитьВКарман](#).

Описание

```
КарманСодержитДанные(Формат :Строка) :Логическое;  
ClipboardHasFormat (Format :String) :Logical;
```

Аргументы

Формат - параметр типа **Строка**, содержащий имя формата.

Назначение

Данная функция позволяет проверить на соответствие формата содержимого буфера обмена.






Функция возвращает значение логического типа **True**, если заданный формат соответствует формату данных в буфере обмена, иначе - **False**.

Пример

```
var ФорматБуфераОбмена :String;  
var ПолученныеДанные :Variant;  
  
ФорматБуфераОбмена = "Наш формат";  
If (not КарманСодержитДанные(ФорматБуфераОбмена)) Then  
    Message("Несоответствие формата");  
    Return;  
End;  
ПолученныеДанные = ВзятьИзКармана(ФорматБуфераОбмена);
```

Процедуры и функции для работы с окнами

Для работы с окнами в Студии применяются следующие свойства и методы из класса [Консоль](#):

-  [Поле ОкноПрограммы / MainWindow](#)
-  [Функция ЕстьВерхнееОкно / TopWindowExists](#)
-  [Поле ЗаголовокПрог / AppCaption](#)
-  [ЗагрузитьИконкуПрог / LoadAppIcon](#)
-  [Функция ЗакрытьВсеОкна / CloseAllWindows](#)

Описание

```
ЗаголовокПрог :Строка;  
AppCaption :String;
```

Назначение

Поле позволяет прочитать и изменить *пользовательскую часть* заголовка главного окна текущей сессии. Под *пользовательской частью* понимается фрагмент, отображаемый в заголовке до символа ":". По умолчанию там выводится название текущего лицевого проекта. После символа ":" всегда выводится название сессии, т.е. информационной базы, к которой произведено подключение. В обобщенном виде это выглядит так:

```
'ТБ ' + <название проекта> + ':' + <сессия>
```

Если поле **AppCaption** устанавливается равным пустой строке, заголовок окна приводится к стандартному виду.

Внимание. Настоятельно рекомендуется вместо этого свойства, которое впоследствии может быть исключено, пользоваться свойством [MainWindow](#).

Пример

```
AppCaption = 'Тест';  
-- заголовок получает вид  
-- 'Тест' + ':' + <сессия>
```

Описание

```
ОкноПрограммы : Окно ;  
MainWindow : Window ;
```

Назначение

Поле возвращает объект класса [Окно](#), являющийся представлением главного окна приложения.

В этом объекте имеют смысл все поля, определённые в классе **Window**, за исключением поля [CurrentObject](#) (попытка его чтения/установки возбудит исключение). Поле [Active](#) объекта MainWindow вернёт True, если клиент на момент получения значения поля является активным приложением в системе, и False в противном случае.

Внимание. Настоятельно рекомендуется пользоваться свойством MainWindow вместо имеющихся в классе **Console** процедуры [LoadAppIcon](#) и поля [AppCaption](#), которые впоследствии могут быть исключены.

Описание

```
ЗагрузитьИконкуПрог(ИмяФайла :Строка);  
LoadAppIcon)(IconFileName :String;
```

Аргументы

ИмяФайла - имя файла с иконкой и путь к нему.

Назначение

Процедура загружает указанную иконку и меняет на нее значок главного окна программы.

Если файл иконки лежит в [каталоге](#) проекта, то доступ к нему может осуществляться с использованием макроса %projects%.

Рекомендуется всегда указывать имя и путь. Если передаваемая строка не содержит пути, файл ищется в текущем каталоге, который определяется способом запуска программы. Если имя файла пустое, восстанавливается стандартная иконка. Также стандартная иконка главного окна восстанавливается автоматически при закрытии сессии.

Если приложение свернуто в системную область панели задач, то при изменении иконки главного окна также меняется иконка в этой области.

Внимание. Настоятельно рекомендуется вместо этой процедуры, которая впоследствии может быть исключена, пользоваться свойством [MainWindow](#).

Пример

```
-- стандартная попытка установить иконку в  
-- соответствии с текущим лицевым проектом сессии  
proc InitIcon;  
  try  
    LoadAppIcon)(  
      SystemPath(spProject, SessionInfo(siProject, 0))  
      +  
      "title.ico");  
  except  
  end;  
end;
```

Описание

```
ЕстьВерхнееОкно(перем Форма : Форма) :Логическое;  
TopWindowExists(var From : Form) :Logical;
```

Аргументы

Форма - переменная, которая будет содержать ссылку на этот объект, при условии, что объект, размещенный в верхнем окне, принадлежит классу, являющемуся наследником класса **Форма**.

Назначение

Возвращает значение Истина, если открыто хотя бы одно окно (модальное или немодальное). При этом верхним окном будет модальное, если оно открыто на экране, или активное немодальное окно, если модальных окон нет.

Если в верхнем окне размещается объект из класса, который является наследником базового класса [Form](#), то ссылка на этот объект будет возвращена через переменную **Форма**.

Пример

```
-- процедура обработки учетных данных, во время которой  
-- текущий операционный документ должен быть закрыт  
proc Обработка;  
var AForm :Form;  
    -- находим верхнюю форму  
    if TopWindowExists(AForm) then  
        -- если это операционный документ, то закрываем его  
        if AForm is ОперДокумент then  
            AForm.Close;  
        end;  
    end;  
    -- дальнейшая обработка  
end;
```


Функция **ЗакретьВсеОкна** / **CloseAllWindows**

Описание

ЗакретьВсеОкна :Логическое
CloseAllWindows :Logical;

Назначение






Возвращает значение True, если все окна закрыты , иначе - False.

Пример

```
proc P1((Sender :Button);  
  if CloseAllWindows:  
    Message("Все окна закрыты");  
  fi;  
end;
```

Процедуры и функции для работы с отчетами

Для работы с отчетами применяются следующие процедуры и функции из класса [Консоль](#):

-  [Функция ВыборОтчета / ChooseReport](#)
-  [Процедура НастроитьОтчет / SetupReport](#)
-  [Процедура ПостроитьОтчет / BuildReport](#)
-  [Функция СоздатьОтчет / CreateReport](#)
-  [Процедура УдалитьОтчет / DeleteReport](#)

Описание

```
НастроитьОтчет(ИмяОтчета:Строка; [; Общий:Логическое] [; ИмяГруппы:Строка]);  
SetupReport(Name:String; [; Common:Logical] [; GroupName:String]);
```

Аргументы

ИмяОтчета - строка, содержащая полное имя отчета вместе с именем проекта (для отчетов, входящих в состав проекта в качестве классов, например, "ПроектТест.ОтчетПоОборотам") или название отчета (для пользовательских отчетов); если через этот аргумент передана пустая строка, создается новый отчет;

Общий - необязательный признак; используется только при создании нового отчета и указывает, является ли добавляемый отчет общим или нет; по умолчанию, **Общий** равно FALSE;

ИмяГруппы - имя группы отчетов, в которой следует создать отчет; имеет смысл только при создании нового отчета; если имя группы опущено, отчет создается в корне иерархии отчетов.

Назначение

Открывает диалог настройки для указанного отчета. Если производится настройка нового отчета, то у пользователя предварительно запрашивается название.

Если необходимо создать новый общий отчет с конкретным именем (т.е. имя должна установить сама программа, а не пользователь), то следует использовать методы и свойства специального класса записи Kernel.Отчеты - именно этот класс используется для хранения настроек общих отчетов.

Пример

```
proc ButtonReportClick (Sender :Button);  
    SetupReport(" ",TRUE);  
end;
```

Описание

```
ПостроитьОтчет(ИмяОтчета:Строка; Начало :Дата; Конец :Дата {; Счета:Строка } {;  
Параметры:Строка} {; СтьльОкна : Окно.СтилиОкон }));  
  
BuildReport(Name:String; Begin:Date; End:Date; {; Accounts:String } {; Parameters:String }  
{; WindowStyle : Window.WindowStyles }));
```

Аргументы

ИмяОтчета - строка, содержащая полное имя отчета вместе с именем проекта (для отчетов, входящих в состав проекта в качестве классов), например, "ПроектТест.ОтчетПоОборотам") или название отчета (для пользовательских отчетов);

Начало - начальная дата отчетного периода;

Конец - конечная дата отчетного периода;

Счета - необязательное условие отбора счетов;

Параметры - необязательное условие отбора счетов по параметрам;

СтьльОкна - необязательный параметр, предопределенная [константа](#), которая определяет способ открытия отчета. Если параметр опущен, то он считается равным AutoDetect.

Назначение

Строит указанный отчет. Если условия отбора по счетам и/или параметрам опущены, их значения берутся из настроек самого отчета.

По умолчанию, когда параметр **СтьльОкна** опущен, он считается равным AutoDetect, т.е. используется значение заданное в отчете. Но, если приложение уже работает в модальном режиме, то отчет тоже откроется модально.

Пример

```
Edit1 :Edit; -- начальная дата  
Edit2 :Edit; -- конечная дата  
proc ButtonReportClick (Sender :Button);  
    try  
        -- строим отчет в модальном окне  
        BuildReport("Торговля.отчПрибыль",  
            StringToDate(Edit1.Text),  
            StringToDate(Edit2.Text),,TRUE);  
    except  
    end;  
end;
```

Описание

```
УдалитьОтчет(ИмяОтчета:Строка);  
DeleteReport(Name:String);
```

Аргументы

ИмяОтчета - строка, содержащая полное имя отчета вместе с именем проекта (для отчетов, входящих в состав проекта в качестве классов, например, "ПроектТест.ОтчетПоОборотам") или название отчета (для пользовательских отчетов).

Назначение

Удаляет указанный отчет из перечня отчетов. Удалять можно только отчеты, созданные во время сессии пользователем или программным образом с помощью **НастроитьОтчет**. Отчеты, входящие в состав проекта, не могут быть удалены таким образом.

Для удаления общих отчетов, добавленных пользователем, можно также напрямую удалять соответствующие записи класса Kernel.Отчеты.

Пример

```
DeleteReport("123");
```

Функция **ВыборОтчета** / **ChooseReport**

Описание

```
ВыборОтчета(var ИмяОтчета :Строка) :Целое;  
ChooseReport (var RepName :String) :Integer;
```

Аргументы

ИмяОтчета - параметр передающийся по ссылке, т.е. переменная типа **Строка**, в которую заносится имя выбранного отчета.

Назначение

Данная функция вызывает диалог "Отчеты", где можно выбрать один из отчетов.

Функция возвращает одно из следующих predefined значений:

кmdВерно/смOK (нажата кнопка "OK" или "Выбор");

кmdОтказ/смCancel (нажата кнопка "Отмена").

Пример

```
var ИмяВыбранногоОтчета :Строка;  
  
If ВыборОтчета(ИмяВыбранногоОтчета) = смOK Then  
    Message('Вы выбрали отчет "' + ИмяВыбранногоОтчета + '"');  
End;
```

Если пользователь выберет отчет из списка и нажмет кнопку "OK", то появится сообщение с именем выбранного отчета.

Функция СоздатьОтчет / CreateReport

Описание

```
СоздатьОтчет(ИмяОтчета:Строка) : ФормаОтчета;  
CreateReport(Name:String) : ReportForm;
```

Аргументы

ИмяОтчета - строка, содержащая полное имя отчета вместе с именем проекта (для отчетов, входящих в состав проекта в качестве классов, например, "ПроектТест.ОтчетПоОборотам") или название отчета (для пользовательских отчетов).

Назначение

Функция создает объект класса-наследника **ФормаОтчета**, имя которого было передано как параметр. Как правило, функция используется для программного построения отчетов, когда необходима скрытая от пользователя обработка данных.

В результате вызова функции сам отчет не строится и не выводится на экран. Для построения отчета необходимо вызвать для полученного объекта метод **ReportForm.Report.Build**. При необходимости можно предварительно изменить настройки объектного свойства [Report](#). Если результаты отчета необходимо визуализировать, то можно либо изменить свойство **Visible** формы отчета, либо динамически создать шаблон с результатами ([FormTemplate](#)) и подгрузить его в какую-либо другую форму (например, в открытый бланк) с помощью [LoadTemplate](#).

Предупреждение. Функция возбуждает исключение, если отчет не имеет формы, например, в формате вывода Текст, Excel и т.п..

Пример

```
ФормаДинОтчета :ReportForm;  
ДатаНачалаПериода :Дата;  
ДатаОкончанияПериода :Дата;  
  
--...  
  
proc СформироватьТотИлиИнойОтчет(Имя :Строка);  
    ФормаДинОтчета = Console.CreateReport(self.ClassProject + '.' + Имя);  
    ФормаДинОтчета.Report.BegDate = ДатаНачалаПериода;  
    ФормаДинОтчета.Report.EndDate = ДатаОкончанияПериода;  
    ФормаДинОтчета.Report.Parameters = "Сумма=руб";  
    ФормаДинОтчета.Report.PreciseBaseClass =  
  
        self.ClassProject + '.' + Имя + "Уточняющий";  
    ФормаДинОтчета.Report.Build;  
    ФормаДинОтчета.FormTemplate;  
    LoadTemplate(SystemPath(spLocal) + 'Temp\' + Имя + '.tpl', -1);  
end;  
  
func КлеткаОтчета_ПриНажатии(Cell :TemplateCell; Action :Template.ClickTypes) :Logical;  
    if (Action <> Template.SingleClick) then  
        ФормаДинОтчета.PreciseReport(Cell);  
    end;  
    Result = False;  
  
end;
```

Для интерактивного взаимодействия с пользователем применяются следующие процедуры и функции из класса Консоль:

-  [Процедура Сообщение / Message](#)
-  [Функция Вопрос / Enquiry](#)
-  [Функция Альтернатива / Alternate](#)
-  [Функция ВопрДаОтказ / EnqOkCancel](#)
-  [Функция ВопрДаНетОтказ / EnqYesNoCancel](#)
-  [Функция Ввод / Input](#)
-  [Функция ВыборФайла / ChooseFile](#)
-  [Функция ВыборПапки / ChooseFolder](#)
-  [Функция ВыборЦвета / ChooseColor](#)
-  [Функция ВыборШрифта / ChooseFont](#)
-  [Функция ВыборПользователей / ChooseUsers](#)
-  [Функция Калькулятор / Calculator](#)
-  [Функция Календарь / Calendar](#)
-  [Функция ВыборПомощи / ChooseHelp](#)
-  [Процедура ОткрытьРедактор / OpenEditor](#)
-  [Функция ЗакрытьРедактор / CloseEditor](#)
-  [Функция ОткрытьБланк / OpenBlank](#)
-  [Функция ОткрытьБланкРедактор / OpenBlankEditor](#)
-  [Функция ОткрытьКартотеку / OpenCardfile](#)
-  [Процедура ОткрытьЖурнал / OpenJournal](#)
-  [Процедура ОткрытьСправочник / OpenReference](#)
-  [Функция ВыборКласса / ChooseClass](#)
-  [Функция ВыборСчета / ChooseAccount](#)
-  [Функция ВыборПризнака / ChooseSign](#)
-  [Функция ВыборБланка / ChooseBlank](#)
-  [Функция ВыборКартотеки / ChooseCardFile](#)
-  [Функция ВыборСправочника / ChooseReference](#)
-  [Функция ВыборЖурнала / ChooseJournal](#)
-  [Функция ВыборОперации / ChooseOperation](#)
-  [Функция ВыборПараметраСчета / ChooseAccountParam](#)
-  [Функция ВыборПоляЗаписи / ChooseRecordField](#)
-  [Функция ВыборПоляСправочника / ChooseReferenceField](#)
-  [Функция ВводФильтраПроводок / InputTransFilter](#)
-  [Функция ВводФильтраЗаписей / InputRecordFilter](#)
-  [Функция ВводФильтраСправочника / InputReferenceFilter](#)
-  [Функция ВводОперации / InputOperation](#)
-  [Функция ВставкаВТабЖур / InsertToTableJur](#)

Описание

```
ОткрытьПомощь(Параметр :Строка);  
OpenHelp(Parameter :String);
```

Аргументы

Параметр - тема помощи, которую нужно открыть.

Назначение

Процедура открывает заданную тему помощи.

Пример

```
OpenHelp( "Введение" );
```

Описание

```
ОткрытьЖурнал (Название:Строка);  
OpenJournal (Name:String);
```

Аргументы

Название - имя класса журнала.

Назначение

Открывает окно указанного журнала. Процедура позволяет программным образом выполнить действие, которое пользователь обычно выполняет из диалога выбора журнала, доступного по команде **Учет|Открыть журнал**. Процедура обеспечивает регламентированный доступ к журналам в случае, если данная команда намеренно скрыта в пользовательском интерфейсе.

Пример

```
proc КнопкаЖурнал_Нажата(К :Кнопка);  
  -- открыть журнал, соответствующий кнопке  
  ОткрытьЖурнал(К.Надпись);  
end;
```

Описание

```
ОткрытьРедактор (Название:Строка);  
OpenEditor (Name:String);
```

Аргументы

Название - название файла, который требуется открыть в окне редактора.

Назначение

Открывает файл с заданным именем в окне редактора.

Для того чтобы программно вызвать диалог редактирования [интерфейсной схемы](#), нужно указать полный путь к *.shi файлу.

Пример

```
proc P1;  
  if (ЕстьФайл ("Testing.idb") ) then  
    -- если файл существует, то его можно редактировать  
    ОткрытьРедактор ("Testing.idb");  
  fi;  
end;
```

Описание

```
ОткрытьСправочник (Название:Строка);  
OpenReference (Name:String);
```

Аргументы

Название - имя класса справочника.

Назначение

Открывает окно указанного справочника. Процедура позволяет программным образом выполнить действие, которое пользователь обычно выполняет из диалога выбора справочника, доступного по команде **Учет|Открыть справочник**.

Процедура обеспечивает регламентированный доступ к справочникам в случае, если данная команда намеренно скрыта в пользовательском интерфейсе.

Пример

```
proc КнопкаСправочник_Нажата(К :Кнопка);  
  -- открыть справочник, соответствующий кнопке  
  ОткрытьСправочник(К.Надпись);  
end;
```

Описание

Сообщение (Параметр:Вариант);
Message (Parameter:Variant);

Аргументы

Сообщение - текст сообщения (строка), переменная или выражение любого типа (автоматически преобразуется в строку внутри процедуры).

Назначение

Выводит модальное окно с заданным сообщением и кнопкой "OK".

Пример

Сообщение (КурсUSD);
Сообщение ("Текущий курс USD = " + Стр (КурсUSD));

Описание

Альтернатива (Заголовок:Строка; Строки[]:Строка [; Ширина:Целое]): Целое;
Alternate (Title:String; Lines[]: String [;Width:Integer]): Integer;

Аргументы

Заголовок - заголовок окна;
Строки[] - массив строк, выводимых на экран;
Ширина - ширина окна в знаках.

Назначение

Выводит окно с заданным заголовком и списком строк определенной длины. Возвращает номер одной выбранной в окне строки или 0 при отказе от выбора (нажатии клавиши Esc).

Пример

```
var Строки [ ] : Строка;  
var Номер : Целое;  
  Строки[1] = "Показать суммы";  
  Строки[2] = "Показать количество";  
  Номер = Альтернатива ( "Выберите показатель", Строки, 20) ;  
  if ( Номер = 1 ) then  
    Message("Суммы");  
  elseif ( Номер = 2 ) then  
    Message("Количество");  
  else  
    Message("Отказ");  
  fi;
```

Функция Ввод / Input

Описание

```
Ввод (var Параметр:[Строка|Целое|Число|Дата|Логическое]; Приглашение:Строка): Целое;  
Input (var Parameter:[String|Integer|Numeric|Date|Logical]; Prompt: String): Integer;
```

Аргументы

Параметр - переменная (локальная или глобальная), принимающая вводимое значение;

Приглашение -приглашение на ввод (произвольный текст).

Назначение

Выводит модальное окно с заданным текстом и полем ввода, которое пользователь может заполнить. Возвращает *кмдВерно*, если была нажата кнопка "ОК" (при этом присваивает введенное значение переменной), или *кмдОтказ* при нажатии кнопки "Отмена".

Если тип введенного выражения не соответствует типу переменной поля бланка, то будет сгенерирована исключительная ситуация.

Пример

```
proc P1;  
  var Код, Рез : Целое;  
  Рез = Ввод (Код, "Введите пароль" ) ;  
  if ( Рез <> кмдОк) then  
    ЗакрытьПрог;  
  else  
    if (Код <> КодБланка) then  
      ЗакрытьПрог;  
    fi;  
  fi;  
end;
```

Функция ВводОперации / InputOperation

Описание

```
ВводОперации ({ИмяОперации :Строка}) :Логическое;  
InputOperation ({OperationName :String}) :Logical;
```

Аргументы

ИмяОперации - необязательный аргумент, имя типовой операции, которая вводится в табличный журнал.

Назначение

Программным способом обеспечивается ввод типовой операции в табличный журнал, который *должен быть открыт на экране*. При успешном вводе операции функция возвращает значение "Истина". Если в текущем окне табличный журнал не открыт, функция возвращает значение "Ложь".

Описание

```

ВводФильтраЗаписей(var Филтp :Стpока; Записи : Класс Запись\[\]; {Класс :Класс
Стpуктура}) :Целое;
InputRecordFilter(var Filter :String; Records : Class Record\[\]; {Class :Class
Structure}) :Integer;

```

Аргументы

Филтp - филтp на запись, который установил пользователь;

Записи - ссылка на класс записи из массива классов записей, которые поддерживаются данной функцией;

Класс - необязательный параметр, класс типа [Стpуктура](#). Если структура задана, то формируется филтp на подтаблицу записи.

Назначение

Функция предназначена для формирования филтpа на запись или на подтаблицу записи, если задан третий параметр. Установка филтpа происходит с помощью диалога ["Филтp"](#), но предварительно открывается диалог ["Выбор поля"](#), в котором пользователь выбирает нужное поле. Филтp может формироваться по нескольким полям записи сразу.

Функция возвращает *cmOk*, если в диалоге была нажата кнопка **Выбор**, в этом случае через параметр **Филтp** возвращается сформированный пользователем филтp на запись. Функция возвращает значение *cmCancel*, если пользователь нажал кнопку **Отмена**.

Пример

```

inclass
  var vClass :Class;
...
proc Pl(Sender :Button);
  var аФилтp : String;
  -- выбираем класс записей
  if ChooseClass(Kernel.Record, vClass, 'Выбор записи') = cmOk then
    Message("Класс = " + Стp(vClass));
  end;
  -- задаем филтp записи для выбранного класса
  if ВводФильтраЗаписей(аФилтp, [vClass]) = cmOk then
    Message("Филтp = " + аФилтp);
  end;
end;

```

Функция ВводФильтраПроводок / InputTransFilter

Описание

```
ВводФильтраПроводок(var Фильтр :Строка {; УсловиеНаСчета :Строка} ) :Целое;  
InputTransFilter(var Filter :String {;AccCondition :String} ) :Integer;
```


Аргументы

Фильтр - строковая переменная, в которую возвращаются условия на параметры для отбора проводок, заданные пользователем;

УсловиеНаСчета - необязательный параметр, строковая переменная для ввода условий на счета в соответствии с [синтаксисом отбора счетов](#).

Назначение

Функция позволяет задать условия на параметры для отбора проводок и последовательно открывает диалоги ["Выбор параметра"](#) и ["Фильтр"](#).

Однако, если в списке параметров выбрать параметр "Документ", то дополнительно появляется список журналов хозяйственных операций (кроме текстовых) области учета, отфильтрованных по маске счетов. Для открытия списка требуется в диалоге "Фильтр" нажать кнопку  поля **Значение**. После выбора журнала отбор документов происходит из журнала, указанного пользователем.

Функция возвращает либо *cmOk*, если была нажата кнопка **Выбор**, либо *cmCancel*, если пользователь отказался от выбора, нажав кнопку **Отмена**. В случае успешного выбора через параметр **Фильтр** возвращается заданное пользователем условие на параметры для отбора проводок.

Пример

```
var аФильтр : String;  
  
if InputTransFilter(аФильтр,"|5?!|6?!") = кмдВерно then  
    Message("Фильтр = " + аФильтр); ....  
end;
```

Функция ВводФильтраСправочника / InputReferenceFilter

Описание

```
ВводФильтраСправочника(var Фильтр :Строка; ИмяСправочника :Строка) :Целое;  
InputReferenceFilter(var Filter :String; ReferenceName :String) :Integer;
```

Аргументы

Фильтр - фильтр на справочник, который пользователь установил в диалоге "Фильтр", открываемом данной функцией;

ИмяСправочника - имя *аналитического* справочника.

Назначение

Функция обеспечивает программный доступ к диалогу ["Фильтр"](#). Одновременно над ним появляется диалог "Выбор поля", в котором в виде дерева отображаются поля аналитического справочника, указанного во втором параметре. После выбора поля верхний диалог закрывается, а выбранное поле проставляется в соответствующее поле диалога "Фильтр".

Если в диалоге пользователь установил фильтр и нажал кнопку **Выбор**, то функция возвращает *cmOk*, а через параметр **Фильтр** возвращается сформированный пользователем фильтр на справочник. Функция возвращает значение *cmCancel*, если пользователь отказался от выбора.

Пример

```
var аФильтр : String;  
if InputReferenceFilter(аФильтр, "спрПроцесс") = cmOk then  
    Message("Фильтр = " + аФильтр);  
end;
```

Функция **ВопрДаНетОтказ** / **EnqYesNoCancel**

Описание

ВопрДаНетОтказ (Вопрос:Строка): Целое;
EnqYesNoCancel (Request: String): Integer;

Аргументы

Вопрос - строка с сообщением или вопросом.

Назначение

Выводит модальное окно с заданным сообщением или вопросом и возвращает константу:

кмдДа - нажата кнопка **Да**;
кмдНет - нажата кнопка **Нет**;
кмдОтказ - нажата кнопка **Отмена**.

Пример

```
прос P1;  
  if ( ВопрДаНетОтказ ("Вы будете обедать ?") = кмдДа ) then  
    Сообщение ("Приятного аппетита");  
  fi;  
end;
```

Функция **ВопрДаОтказ** / **EnqOkCancel**

Описание

```
ВопрДаОтказ (Вопрос:Строка): Целое;  
EnqOkCancel (Request: String): Integer;
```

Аргументы

Вопрос - строка с сообщением или вопросом.

Назначение

Выводит модальное окно с заданным сообщением или вопросом.

Возвращает *кмдВерно*, если была нажата кнопка **ОК** или *кмдОтказ* - если кнопка **Отмена**.

Пример

```
proc P1;  
  if ( ВопрДаОтказ ("Вы будете обедать ?") = кмдВерно ) then  
    Сообщение ("Приятного аппетита");  
  fi;  
end;
```

Описание

```
Вопрос ([Заголовок:Строка]; Вопрос:Строка; ТекстКнопок[]:Строка [; Фокус:Целое] [; ТипИконки:Консоль.ТипИконки]): Целое;  
Enquiry ([Title: String]; Request:String; ButtonFace[]:String [; Focus:Integer] [; IconType:Console.IconType]): Integer;
```

Аргументы

Заголовок - необязательная строка с заголовком диалогового окна;

Вопрос - строка с сообщением или вопросом;

ТекстКнопок[] - массив строк, содержащий названия кнопок;

Фокус - номер кнопки, которая получит фокус ввода. Кнопки нумеруются, начиная с 1. По умолчанию фокус получает первая кнопка.

ТипИконки - одна из predefined целочисленных констант, обозначающих стандартные значки, которые могут выводиться в диалоговом окне. Если этот параметр опущен, выводится иконка со знаком вопроса. Параметр **ТипИконки** может принимать одну из следующих констант:

itInformation|тиИнформация - системная иконка "информация" (с восклицательным знаком);

itQuestion|тиВопрос - системная иконка "вопрос" (с вопросительным знаком);

itWarning|тиПредупреждение - системная иконка "Предупреждение";

itError|тиОшибка - системная иконка "Ошибка";

itApplication|тиПрограмма - иконка Студии;

itWinLogo|тиСистема - системная иконка Windows.

Назначение

Выводит модальное окно с заданным сообщением или вопросом, в нижней части которого расположено несколько кнопок. Их число определяется по размерности массива ТекстКнопок.

Возвращает номер нажатой кнопки или 0, если окно было закрыто (или нажата клавиша **Esc**). Если первый параметр не задан, окно выводится с заголовком по умолчанию ("Подтверждение").

Пример

```
proc Pl;  
  if ( Вопрос ( , "Вы будете обедать?",  
    ["Несомненно", "Может быть", "Не знаю"] ) = 1 ) then  
    Сообщение ( "Приятного аппетита" );  
  fi;  
end;
```

Обратите внимание, что при отсутствии первого параметра в вызове функции явно указывается первая запятая.

Описание

```
ВставкаВТабЖур(ИмяЖурнала :Строка; Дата :Дата; Опер :Строка) :Запись;  
InsertToTableJur(JournalName :String; Date :Date; Oper :String) :Record;
```

Аргументы

ИмяЖурнала - имя табличного журнала;

Дата - дата записи хозяйственной операции в журнал;

Опер - текст проводки или операции, начиная с двоеточия, в формате текстового журнала.

Назначение

Вставляет проводку (типовую операцию) в заданный табличный журнал за указанную дату. Функция возвращает запись, соответствующую вставленной проводке или типовой операции.

Описание

```
ВыборБланка (var КлассБланка: Класс ФормаБланка ): Целое;  
ChooseBlank (var BlankClass: Class BlankForm): Integer;
```

Аргументы

КлассБланка - переменная, принимающая на выходе выбранный пользователем класс бланка; перед вызовом функции переменная может быть проинициализирована ссылкой на какой-либо класс бланка, в результате чего именно этот класс будет изначально выделен в диалоге.

Назначение

Функция выводит на экран диалог выбора типа (класса) бланка.

Функция возвращает одну из predefined констант из класса **Консоль:кмдВерно** - если выбор был осуществлен, **кмдОтказ** - в противном случае. В переданную переменную функция запишет выбранный класс.

Пример

```
proc КнопкаБланк_Нажата(К :Кнопка);  
  var В : Класс ФормаБланка;  
  if ВыборБланка(В) = кмдВерно then  
    В.СоздатьВидимым;  
  end;  
end;
```


Функция ВыборЖурнала / ChooseJournal

Описание

```
ВыборЖурнала (var ИмяЖурнала: Строка ): Целое;  
ChooseJournal (var RefName: String): Integer;
```

Аргументы

ИмяЖурнала - переменная, принимающая на выходе имя выбранного пользователем журнала. Перед вызовом функции переменная может быть проинициализирована именем какого-либо журнала, в результате чего именно этот журнал будет изначально выделен в диалоге.

Назначение

Функция выводит на экран диалог выбора журнала и возвращает одну из предопределенных констант из класса **Консоль**:

кmdВерно - если выбор был осуществлен;
кmdОтказ - в противном случае.

В переданную переменную функция запишет имя выбранного журнала.

Пример

```
proc КнопкаЖурнал_Нажата(К :Кнопка);  
  var J : Строка;  
  if ВыборЖурнала(J) = кmdВерно then  
    ОткрытьЖурнал(J);  
  end;  
end;
```

Функция ВыборКартотеки / ChooseCardFile

Описание

```
ВыборКартотеки (var КлассКартотеки: Класс ФормаКартотеки ): Целое;  
ChooseCardFile (var CardFileClass: Class CardForm): Integer;
```

Аргументы

КлассКартотеки - переменная, принимающая на выходе выбранный пользователем класс картотеки; перед вызовом функции переменная может быть проинициализирована ссылкой на какой-либо класс картотеки, в результате чего именно этот класс будет изначально выделен в диалоге.

Назначение

Функция выводит на экран диалог выбора типа (класса) картотек.

Функция возвращает одну из predefined констант из класса **Консоль:кмдВерно** - если выбор был осуществлен, **кмдОтказ** - в противном случае. В переданную переменную функция запишет выбранный класс.

Пример

```
proc КнопкаКартотека_Нажата(К :Кнопка);  
  var С : Класс ФормаКартотеки;  
  if ВыборКартотеки(С) = кмдВерно then  
    С.СоздатьВидимым;  
  end;  
end;
```

Описание

```
ВыборКласса(БазовыйКласс :Класс; var Класс :Класс; {Заголовок :Строка}):Целое;  
ChooseClass (BaseClass :Class; var Class :Class; {Caption :String}) :Integer;
```

Аргументы

БазовыйКласс - базовый класс, наследники которого будут перечислены в открывшемся окне;

Класс - класс, который требуется определить, если он задан (ссылка не равна nil), то при открытии окна курсор позиционирует на этот класс;

Заголовок - необязательный параметр, заголовок окна, если он отсутствует, то заголовок задается по умолчанию.

Назначение

Функция предназначена для выбора класса, отображаемого в [диалоге](#) с заданным заголовком. В диалоге перечислены все классы, наследуемые от заданного базового класса, если второй аргумент не равен nil, то заданный класс будет выделен при открытии диалога.

Функция возвращает предопределенную константу [кmdВерно / cmOk](#) или [кmdОтказ / cmCancel](#), зависящую от выбранной кнопки **Выбор** или **Отмена**. Если была нажата кнопка **Выбор**, то при выходе из функции ссылка на выбранный класс содержится в аргументе **Класс**.

Пример

```
var vClass :Class;  
.....  
vClass = блРазноскаПривязокОттрузок;  
if ChooseClass(Kernel.BlankForm, vClass, 'Выбор формы') = cmOk then  
    Message("Класс = " + Стр(vClass));  
end;
```

Описание

```
ВыборОперации(var ИмяОперации :Строка ):Целое;  
ChooseOperation(var OperationName :String) :Integer;
```

Аргументы

ИмяОперации - имя типовой операции.

Назначение

Функция предназначена для выбора типовой операции из списка. При открытии заданная типовая операция будет выделена в списке. Функция возвращает одну из предопределенных констант из класса **Консоль**:

- **кмдВерно** - если выбор был осуществлен, при этом в переменную будет записано имя выбранной операции;
- **кмдОтказ** - если выбор не сделан.

Пример

```
proc КнопкаОперация_Нажата(ТО :Кнопка);  
  if ВыборОперации("БанковскаяВыписка") = кмдВерно then  
    Message("Операция выбрана.");  
  end;  
end;
```

Описание

ВыборПапки (var Параметр:Строка; Приглашение:Строка): Целое;
ChooseFolder (var Parameter:String; Prompt: String): Integer;

Аргументы

Параметр - переменная (локальная или глобальная), принимающая вводимое значение;
Приглашение - приглашение на ввод (произвольный текст).

Назначение

Выводит модальное окно с заданным текстом и деревом иерархической структуры каталогов на компьютере, где пользователь может выбрать требуемый каталог.

Возвращает кмдВерно, если была нажата кнопка "ОК" (при этом присваивает введенное значение переменной), или кмдОтказ при нажатии кнопки "Отмена".

Пример

```
proc P1;  
  var SS : Строка;  
  var Рез : Целое;  
  Рез = ВыборПапки(SS, "Выберите каталог") ;  
  if ( Рез = кмдВерно) then  
    Message(SS);  
  fi;  
end;
```

Функция ВыборПараметраСчета / ChooseAccountParam

Описание

```
ВыборПараметраСчета(var ИмяПараметра :Строка {; УсловиеНаСчета :Строка} ) :Целое;  
ChooseAccountParam(var ParamName :String {;AccCondition :String} ) :Integer;
```

Аргументы

ИмяПараметра - строковая переменная, в которую возвращается выбранный пользователем параметр счета;

УсловиеНаСчета - необязательный параметр, строковая переменная для ввода условий на счета в соответствии с [синтаксисом отбора счетов](#).

Назначение

Функция позволяет программным способом вызывать диалог "[Выбор параметра](#)", содержащий иерархический список параметров счетов. В списке параметров также можно выбрать параметр "Документ".

Функция возвращает либо *стOk*, если была нажата кнопка **Выбор**, либо *стCancel*, если пользователь отказался от выбора, нажав кнопку **Отмена**. В случае успешного выбора через параметр **ИмяПараметра** возвращается заданный пользователем параметр счета.

Пример

```
var ПарамСчета : String;  
....  
if ChooseAccountParam (ПарамСчета,"2?!|44!|57!|7?!|6?!") = кмдВерно then  
    Message("Параметр счета = " + ПарамСчета);  
end;
```

Описание

```
СписокПользователей(var СписокПользователей :Строка[]; {var СписокИдентификаторов :Строка
[]}; {МножественныйВыбор :Логическое}) :Целое;
```

```
ChooseUsers(var UserList :String[]; {var SIDList :String[]; {MultiSelect :Logical}):Integer;
```

Аргументы

СписокПользователей - список имен пользователей;

СписокИдентификаторов - необязательный параметр, список SID'ов пользователей. Если при вызове будет передан непустой параметр SIDList, то на выходе в нём будут находиться строковые представления SID'ов пользователей в том порядке, как они указаны в массиве **СписокПользователей**;

МножественныйВыбор - необязательный параметр, указывающий вариант выбора:

False (по умолчанию) - разрешается выбрать только одного пользователя, т.е. массивы СписокПользователей и СписокИдентификаторов будут содержать один элемент [1];
True - разрешается выбирать более одного пользователя.

Примечание. При авторизации доменных пользователей добавлена возможность идентификации пользователя не по имени, а по уникальному идентификатору (SID - Security Identifier), который сохраняется неизменным при изменении имени пользователя.

Назначение

Данная функция вызывает системный диалог Windows, позволяющий выбрать одного или нескольких пользователей из списка пользователей в доменах, к которым относится клиентский компьютер. Возвращает одно из следующих предопределенных значений:

кмдВерно/смOK - пользователь выбрал запись/записи из списка и нажал кнопку **ОК** или **Выбор**;
кмдОтказ/смCancel - пользователь нажал кнопку **Отмена**.

В первом случае в списке пользователей (UserList) будут записи вида <UserName>@<DomainName>, во втором - список будет пуст.

Предупреждение 1. Функция будет работать на версии Windows 2000 (Server/Professional) и выше для получения диалога выбор пользователей. Для более низкой версии *вызов функции сгенерирует исключение*.

Предупреждение 2. Возвращение SIDов поддерживается операционной системой Windows NT, начиная с версии 4.0, на более низких версиях список идентификаторов (SIDList) будет содержать пустые строки.

Пример

```
var I :Integer;
var vUserList, vSIDList :String[];

if ChooseUser(vUserList, vSIDList) = cmOk then
  for I = 1..LengthOfArray(vUserList) do
    Trace("Пользователь N" + Str(I) +
      " Список пользователей: " + vUserList[I] + " SID: " + vSIDList[I]);
  end;
end;
```

Описание

```
ВыборПоляЗаписи(var ИмяПоля :Строка; Записи : Класс Запись\[\]; {Класс :Класс
Структура}) :Целое;
ChooseRecordField(var FieldName :String; Records : Class Record\[\]; {Class :Class
Structure}) :Integer;
```

Аргументы

ИмяПоля - параметр, в который будет передано имя поля записи, выбранное пользователем;

Записи - массив классов записей, которые будут показаны в диалоге;

Класс - класс типа [Структура](#). Если параметр задан, то в диалоге отображаются поля из заданной структуры, если не задана, то все - поля записей.

Назначение

Функция открывает диалог "[Выбор поля](#)" с иерархией классов записей, в котором пользователь может выбрать нужное поле. Если параметр **Класс** опущен, выводится все дерево классов записей, в противном случае - лишь те, которые принадлежат заданной подтаблице.

Функция возвращает либо *cmOk*, если была нажата кнопка **Выбор**, либо *cmCancel*, если пользователь отказался от выбора. В случае успешного выбора через параметр **ИмяПоля** возвращается выбранное поле записи с учетом всех разыменований.

Если при вызове функции параметр **ИмяПоля** был ненулевым, то при открытии диалога на данном поле будет установлен курсор.

Пример

```
var КлассЗаписи :Class;
....
proc P1(Sender :Button);
var локПоле : String;
var локКласс : Class;

-- выбираем класс записи
if ВыборКласса(Запись, локКласс, "Выберите запись") = кмдВерно then
    КлассЗаписи = локКласс;
end;
....
-- для выбранного класса определяем поле записи
if ВыборПоляЗаписи(локПоле, [КлассЗаписи]) = кмдВерно then
    ....
end;
end;
```


Функция ВыборПоляСправочника / ChooseReferenceField

Описание

```
ВыборПоляСправочника(var ИмяПоля :Строка; ИмяСправочника :Строка) :Целое;  
ChooseReferenceField(var FieldName :String; ReferenceName :String) :Integer;
```

Аргументы

ИмяПоля - имя поля справочника, которое требуется выбрать в диалоге;

ИмяСправочника - имя *аналитического* справочника.

Назначение

Функция позволяет выбрать нужное поле аналитического справочника и открывает диалог ["Выбор поля"](#) с иерархической структурой полей.

Функция возвращает либо *стOk*, если была нажата кнопка **Выбор**, либо *стCancel*, если пользователь отказался от выбора. В случае успешного выбора через параметр **ИмяПоля** возвращается выбранное поле аналитического справочника с учетом всех разыменований.

Если при вызове функции параметр **ИмяПоля** был ненулевым, то при открытии диалога на данном поле будет установлен курсор.

Пример

```
var локПоле : String;  
....  
локПоле= "Изготовитель.ВидДеятельности.Наименование";  
if ВыборПоляСправочника(локПоле, "спрРесурс") = кмдВерно then  
....  
end;
```

Описание

```
ВыборПомощи(var ТемаПомощи :Строка) :Целое;  
ChooseHelp(var Parameter :String) :Integer;
```

Аргументы

ТемаПомощи - параметр, которому присвоится выбранная тема помощи.

Назначение

Данная функция позволяет открыть список тем помощи и выбрать конкретную тему. Функция возвращает одно из следующих predefined значений:

кмДВерно/смOK - пользователь нажал кнопку **ОК** или **Выбор**. В этом случае в параметр **ТемаПомощи** возвращается имя выбранной темы;
кмДОтказ/смCancel - пользователь нажал кнопку **Отмена**.

Пример

```
var ТемаПомощи :Строка;  
  
If ВыборПомощи(ТемаПомощи) = смOK Then  
    Message('Выбранная тема: ' + ТемаПомощи + '');  
End;
```

Функция ВыборПризнака / ChooseSign

Описание

ВыборПризнака (ИмяСправочника :Строка; var Признак :Признак; {Фильтр :String}) :Целое;
ChooseSign (RefName :String; var Sign :Sign; {Filter :String}) :Integer;

Аргументы

ИмяСправочника - имя справочника;

Признак - переменная, в которую записывается выбранный пользователем признак;

Фильтр - необязательный параметр, фильтр, который устанавливается в качестве пользовательского фильтра при открытии справочника.

Назначение

Открывает в текущем окне указанный справочник с учетом фильтра, если он задан. Пользователь может выбрать необходимый ему признак, который будет записан в переменную **Признак**.

Функция возвращает предопределенную константу *кмдВерно* из класса **Консоль**, если выбор был осуществлен, или *кмдОтказ* - в противном случае.

Пример

```
ТекущаяВалюта: Валюта;  
  
proc КнопкаВалюта_Нажата(К :Кнопка);  
  var i : Integer;  
  var V : Валюта;  
  i = ВыборПризнака("Валюта",V);  
  if i = кмдВерно then  
    ТекущаяВалюта = V;  
  end;  
end;
```

Описание

```
ВыборСправочника(var ИмяСправочника :Строка) :Целое;  
ChooseReference(var ReferenceName :String) :Integer;
```

Аргументы

ИмяСправочника - имя справочника, при открытии курсор позиционирует на заданном справочнике.

Назначение

Открывает в текущем окне диалог выбора справочника, при этом заданный справочник выделяется. Пользователь может выбрать другой справочник, который будет записан в переменную **ИмяСправочника**.

Функция возвращает предопределенную константу *кмдВерно* из класса **Консоль**, если выбор был осуществлен, или *кмдОтказ* - в противном случае.

Функция ВыборСчета / ChooseAccount

Описание

ВыборСчета (перем Счет :Счет [; Условие :Строка]): Целое;
ChooseAccount (var Acc :Account [; Condition :String]): Integer;

Аргументы

Счет - переменная, содержащая на входе (при необходимости) и на выходе идентификатор счета;

Условие - записанное в стандартном виде [условие отбора счетов](#).

Назначение

Функция выводит на экран диалог "Счета", в котором пользователь может выбрать необходимый ему счет. Если параметр **Условие** опущен, выводится все дерево счетов, в противном случае - лишь те, что удовлетворяют условию отбора.

Функция возвращает либо *cmOk*, либо *cmCancel* (в зависимости от того сделал пользователь выбор или нет). В случае успешного выбора через параметр **Счет** возвращается выбранный счет.

Если при вызове функции параметр **Счет** был ненулевым и указанный счет удовлетворял условию, то при открытии диалога на данный счет будет установлен курсор.

Пример

Пример

```
var СчетВыбранныйРанее :Счет;
```

```
proc pl;  
var result : Integer;  
var СчетИкс : Счет;  
    СчетИкс = СчетВыбранныйРанее;  
    result = ВыборСчета(СчетИкс, "2?!|44!|57!|7?!|6?!");  
    if result = cmOk then  
        СчетВыбранныйРанее = СчетИкс;  
    end;  
end;
```

Функция ВыборФайла / ChooseFile

Описание

ВыборФайла (var Параметр:Строка; Приглашение:Строка [; Расширения:Строка]): Целое;
ChooseFile (var Parameter:String; Prompt: String [; Extensions:String]): Integer;

Аргументы

Параметр - переменная (локальная или глобальная), принимающая вводимое значение;

Приглашение - приглашение на ввод (произвольный текст);

Расширения - перечень расширений файлов в формате "Первое название типа файлов|*.xxx|Второе название типа файлов|*.yyy", где xxx, yyy - требуемые расширения типов файлов, количество типов не ограничено.

Назначение

Выводит модальное окно с заданным текстом и списком объектов файловой системы, в котором пользователь может выбрать требуемый файл.

Возвращает *кмдВерно*, если была нажата кнопка "ОК" (при этом присваивает введенное значение переменной), или *кмдОтказ* при нажатии кнопки "Отмена". Если последний параметр не задан, выводится список файлов всех типов.

Пример

```
proc Pl;  
  var SS : Строка;  
  var Рез : Целое;  
  Рез = ВыборФайла(SS, "Выберите файл",  
    "Журнал|*.jur|Структура учета|*.lis|Переменные|*.var" ) ;  
  if ( Рез = кмдВерно) then  
    Message(SS);  
  fi;  
end;
```

Функция **ВыборЦвета** / **ChooseColor**

Описание

ВыборЦвета (var Цвет:Целое [; Заголовок:Строка]): Целое;
ChooseColor (var Color:Integer [; Title:String]): Integer;

Аргументы

Цвет - переменная (локальная или глобальная), принимающая вводимое значение;

Заголовок - необязательный заголовок, поясняющий суть операции.

Назначение

Выводит модальное диалоговое окно для выбора цвета.

Возвращает *кмдВерно*, если была нажата кнопка "ОК" (при этом присваивает введенное значение переменной), или *кмдОтказ* при нажатии кнопки "Отмена".

Пример

```
proc Button1_Pressed(B:Button);
  var Цвет : Целое;
  var Рез : Целое;
  Рез = ВыборЦвета(Цвет);
  if ( Рез = кмдВерно) then
    Шаблон.Цвет = Цвет;
  fi;
end;
```

Функция ВыборШрифта / ChooseFont

Описание

ВыборШрифта (Шрифт :[Шрифт](#); {Заголовок:Строка}; {РедактироватьЦвет :Логическое}): Целое;
ChooseFont (Font :[Font](#); {Title:String}; {EditColor :Logical}): Integer;

Аргументы

Шрифт - ссылка на объект класса Шрифт, полученная ранее от системы (например, как свойство другого объекта);

Заголовок - необязательный заголовок, поясняющий суть операции.

РедактироватьЦвет - необязательный логический параметр, позволяющий настраивать цвет шрифта. Если параметр РедактироватьЦвет = True, в диалоге появится дополнительный выпадающий список выбора цвета. Если параметр отсутствует, то по умолчанию он равен False, и изменить цвет нельзя.

Назначение

Выводит модальное диалоговое окно для выбора шрифта. В результате работы функции объект Шрифт может изменить свои свойства на те, что предпочтет пользователь.

Возвращает *кмДВерно*, если была нажата кнопка **ОК** (при этом присваивает шрифту новые свойства), или *кмОтказ* при нажатии кнопки **Отмена**.

Пример

```
proc Button1_Pressed(B:Button);  
    ВыборШрифта (B.Шрифт);  
end;
```


Описание

`ЗакрытьРедактор` (Название:Строка): Логическое;
`CloseEditor` (Name:String): Logical;

Аргументы

Название - название файла, который в данный момент открыт в окне редактора.

Назначение

Закрывает окно редактора с заданным файлом. Если файл был отредактирован и пользователь в ответ на запрос, нужно ли сохранить изменения, нажал кнопку *Отмена*, функция возвращает значение ЛОЖЬ, а окно редактора не закрывается. В остальных случаях возвращается ИСТИНА.

Пример

```
proc Pl;  
  if ( НЕ ЗакрытьРедактор("Testing.txt") ) then  
    Trace("Редактирование продолжается");  
  fi;  
end;
```

Функция Календарь / Calendar

Описание

```
Календарь ([День:Дата]): Дата;  
Calendar ([ADay:Date]): Date;
```

Аргументы

День - начальная дата, которая будет выделена в календаре в момент его открытия.

Назначение

Открывает модальное окно с календарем и возвращает выбранную дату. Если функция вызвана без параметров, то при открытии календаря текущей будет системная дата.

Пример

```
ДатаПоездки : Дата;  
ДатаПоездки = Календарь;
```

Функция Калькулятор / Calculator

Описание

Калькулятор : Число;
Calculator : Numeric;

Назначение

Открывает модальное окно с калькулятором и возвращает значение числового типа - результат вычисления.

Пример

Сумма : Число;
Сумма = **Калькулятор**;

Функция ОткрытьБланк / OpenBlank

Описание

ОткрытьБланк (Название:Строка [; СтилЬОкна: [Окно.СтилиОкон](#)]): Целое;
OpenBlank (BlankName:String [; WindowStyle : [Window.WindowStyles](#)): Integer;

Аргументы

Название - название бланка;

СтилЬОкна - предопределенная [константа](#), которая определяет способ открытия бланка. Если параметр опущен, то он считается равным AutoDetect.

Назначение

Открывает окно с указанным бланком в заданном режиме. По умолчанию, когда второй аргумент не задан, он считается равным AutoDetect, т.е. используется значение, заданное в шаблоне бланка. Но если же приложение уже работает в модальном режиме, то новый бланк тоже откроется модально.

Если бланк открыт в модальном режиме, управление возвращается на фрагмент кода, идущий за вызовом данной функции, только после того как бланк будет закрыт. Если бланк открыт в немодальном режиме, управление сразу передается на следующую за вызовом строку кода.

Функция возвращает результат работы бланка (значение, которое передается через процедуру **Close** объекта [Форма / Form](#)). Как правило, это одна из предопределенных констант из класса **Консоль**: *кмдВерно*, *кмдОтказ*, *кмдДа* или *кмдНет*.

Пример

```
proc P1;  
  if (OpenBlank("БланкРедактор1",TRUE) = cmCancel) then  
    Message("Установки не сохранены");  
  end;  
end;
```

Функция ОткрытьБланкРедактор / OpenBlankEditor

Описание

ОткрытьБланкРедактор(Название:Строка; Запись:Запись [; СтилОкна: [СтилиОкон](#)]): Целое;
OpenBlankEditor(BlankName:String; Record:Record [; WindowStyle :WindowStyles]): Integer;

Аргументы

Название - название бланка;

Запись - ссылка на документ (запись) из картотеки, которую нужно загрузить в бланк;

СтилОкна - предопределенная [константа](#), которая определяет способ открытия бланка. Если параметр опустить, то он считается равным AutoDetect, и открывается дочернее окно.

Назначение

Открывает окно с указанным бланком и загружает в него заданную запись (документ) картотеки. Бланк должен быть назначен бланком- редактором картотеки.

Если третий параметр опущен, то по умолчанию он считается равным AutoDetect и открывается дочернее окно. Но если же приложение уже работает в модальном режиме, то бланк тоже открывается в модальном (монопольном) режиме.

Если бланк открыт в модальном режиме, управление возвращается на фрагмент кода, идущий за вызовом данной функции, только после того как бланк будет закрыт. Если бланк открыт в немодальном режиме, управление сразу передается на следующую за вызовом строку кода.

Функция возвращает результат работы бланка (значение, которое передается через процедуру **Close** объекта [Форма / Form](#)). Как правило, это одна из предопределенных констант из класса **Консоль**: *кмдВерно*, *кмдОтказ*, *кмдДа* или *кмдНет*.

Пример

```
proc Pl;  
  if (OpenBlankEditor("БланкРедактор1",Document,TRUE) = cmCancel) then  
    Message("Установки не сохранены");  
  end;  
end;
```

Описание

ОткрытьКартотеку (Название:Строка {; var Запись: Запись} {; Филтър:Строка} {; СтилъОкна: [Окно.СтилиОкон](#) }): Целое;

OpenCardfile(CardName:String {; var Record:Record} {; Filter:String} {; WindowStyle : [Window.WindowStyles](#) }): Integer;

Аргументы

Название - название картотеки;

Запись - ссылка на запись картотеки, которая выделена в бланке-редакторе картотеки;

Филтър - строковое выражение, определяющее условия отбора записей (документов), попадающих в картотеку.

СтилъОкна - предопределенная [константа](#), которая определяет способ открытия картотеки. Если параметр не задан, то он считается равным AutoDetect.

Любой из необязательных аргументов, заключенных в фигурные скобки, может быть опущен, но, если это не последний аргумент, вместо него должна стоять запятая. Например, для случая, когда не используется второй аргумент, следует записать:

```
ОткрытьКартотеку(Накладные, , "Дата=25.11.1999",Ложь);
```

Назначение

Открывает окно с указанной картотекой и, при необходимости, выделяет в ней заданную запись (документ). При возврате из функции, если пользователь выбрал какую-либо запись (документ) в картотеке, ссылка на него передается через аргумент **Запись**.

В строке с фильтром можно указать произвольное выражение логического типа, в котором участвуют условия на значения полей отбираемых записей. Самый простой фильтр, задающий отбор всех записей - это пустая строка.

По умолчанию, когда второй аргумент не задан, он считается равным AutoDetect, т.е. используется значение, заданное в шаблоне картотеки. Но если же приложение уже работает в модальном режиме, то картотека тоже откроется модально.

При открытии картотеки в модальном режиме управление возвращается на фрагмент кода, следующий за вызовом данной функции, только после того как картотека будет закрыта. Если картотека открыта в немодальном режиме, управление сразу передается на следующую за вызовом строку кода.











Функция возвращает результат работы картотеки (значение, которое передается через процедуру **Close** объекта [Форма / Form](#)). Как правило, это одна из предопределенных констант из класса **Консоль**: *кмдВерно*, *кмдОтказ*, *кмдДа* или *кмдНет*.

Пример

```
proc P1;
var DD : Накладная;
  if (ОткрытьКартотеку("Карта1",DD,"",True) = cmOK) then
    Message( "Выбрана накладная"+Str(DD.Номер) );
  end;
end;
```

Внимание! Более простым и эффективным способом открытия картотеки является вызов метода ShowForm, например, Карта1.ShowForm.

В класс **Математика** входят следующие математические процедуры и функции:

-  [Arcsin](#)
-  [Arctan](#)
-  [Div](#)
-  [Mod](#)
-  [Sin](#)
-  [Абс / Abs](#)
-  [Корень / Sqrt](#)
-  [Лог / Log](#)
-  [Окр / Round](#)
-  [Отбр / Trunc](#)
-  [Цел / Int](#)
-  [Числ / Num / КакЧисло](#)
-  [Эксп / Exp](#)

Описание

Arcsin(Выражение :Число) :Число;

Аргументы

Выражение - числовое выражение действительного типа, для которого необходимо вычислить арксинус. Аргумент должен лежать в области определения арксинуса $(-1, +1)$.

Назначение

Вычисляет арксинус заданного аргумента. Угол вычисляется в радианах.

Описание

Arctan(Выражение :Число) :Число;

Аргументы

Выражение - числовое выражение действительного типа, для которого необходимо вычислить арктангенс.

Назначение

Вычисляет арктангенс заданного аргумента. Угол вычисляется в радианах.

Описание

Div(Делимое:Целое; Делитель:Целое): Целое;

Аргументы

Делимое и Делитель - целочисленные значения.

Назначение

Возвращает целочисленный результат деления делимого на делитель.

Пример

```
proc P1;  
  var Дано : Целое;  
  Дано = Div ( 10, 3 ); -- Дано = 3  
end;
```

Описание

Mod(Делимое:Целое; Делитель:Целое): Целое;

Аргументы

Делимое и Делитель - целочисленные значения.

Назначение

Возвращает остаток от деления делимого на делитель

Пример

```
proc P1;  
  var Дано : Целое;  
  Дано = Mod ( 10, 3 ); -- Дано = 1  
end;
```

Описание

sin(Выражение :Число) :Число;

Аргументы

Выражение - числовое выражение действительного типа, для которого необходимо вычислить синус.

Назначение

Вычисляет синус заданного аргумента. Угол задается в радианах.

Описание

```
Абс(Выражение :Число) :Число;  
Abs(Expression :Numeric) :Numeric;
```

Аргументы

Выражение - числовое выражение действительного типа, для которого необходимо вычислить абсолютное значение.

Назначение

Возвращает положительное значение, равное по модулю заданному аргументу.

Описание

Корень(Выражение:Число): Число;
Sqrt(Expression:Numeric): Numeric;

Аргументы

Выражение - числовое выражение действительного типа (или просто число), для которого необходимо вычислить квадратный корень.

Назначение

Вычисляет квадратный корень от заданного аргумента.

Пример

```
proc P1;  
  var Катет1, Катет2, Гипотенуза : Число;  
  Катет1 = 3;  
  Катет2 = 4;  
  Гипотенуза = Корень ( Катет1*Катет1 + Катет2*Катет2 );  
end;
```

Описание

```
Лог(Выражение:Число): Число;  
Log(Expression:Numeric): Numeric;
```

Аргументы

Выражение - числовое выражение действительного типа (или просто число), для которого необходимо вычислить натуральный логарифм. Значение параметра должно принадлежать области определения логарифма (интервал]0;+ ∞), в противном случае генерируется исключение.

Назначение

Вычисляет натуральный логарифм (по основанию e) заданного числа.

Пример

```
proc Pl;  
  var Дано : Число;  
  Дано = Лог ( 10 ); -- Дано = 2.30258509299405  
end;
```

Описание

```
Окр(Выражение: {Целое|Число|Строка} [;Точность:Целое]): Число;  
Round(Expression: {Integer|Numeric|String} [;Accuracy:Integer]): Numeric;
```

Аргументы

Выражение - числовое выражение или строка, которые требуется преобразовать в действительное число и округлить с заданной точностью;

Точность - количество знаков в дробной части.

Назначение

Возвращает преобразованное в действительное число и округленное по математическим правилам значение первого параметра. Если в заданном строковом представлении числа содержится ошибка или тип параметра не соответствует допустимому типу аргумента, функция генерирует исключение. Если второй параметр отсутствует, то округление происходит до целого числа (вся дробная часть отбрасывается). Если второй параметр положителен, то он задает количество знаков в дробной части результирующего числа после десятичной точки. При отрицательном значении параметра округляются разряды целой части числа.

Пример

```
proc P1;  
  var Дано : Число;  
  Дано = Окp ( "49.125" ); -- Дано = 49  
  Дано = Окp ( "49.125", 2); -- Дано = 49.13  
  Дано = Окp ( 49.125, -1); -- Дано = 50  
  Дано = Окp ( "абвгд" ); -- сообщение об ошибке  
end;
```


Описание

```
Отбр(Выражение:{Целое|Число|Строка} [;Точность:Целое]): Число;  
Trunc(Expression:{Integer|Numeric|String} [;Accuracy:Integer]): Numeric;
```

Аргументы

Выражение - числовое выражение или строка, которые требуется преобразовать в действительное число и округить до указанного разряда.

Точность - количество значащих разрядов в дробной (если **Точность** положительна) или целой части (если **Точность** отрицательна) числа.

Назначение

Возвращает преобразованное в действительное число значение первого параметра. Если второй параметр не задан, то отбрасывается вся дробная часть числа. Если второй параметр отрицателен, то начиная с разряда, равного модулю значения второго параметра, все значащие цифры обнуляются, а дробная часть - отбрасывается. Если второй параметр положителен, то в дробной части результирующего числа сохраняется столько знаков, сколько указано во втором параметре. Если в заданном строковом представлении числа содержится ошибка или тип параметра не соответствует допустимому типу аргумента, функция генерирует исключение.

Пример

```
proc P1;  
  var Дано : Число;  
  Дано = Отбр ( "49.125" ); -- Дано = 49  
  Дано = Отбр ( "49.125", 2); -- Дано = 49.12  
  Дано = Отбр ( 49.125, -1); -- Дано = 40  
  Дано = Отбр ( "абвгд" ); -- сообщение об ошибке  
end;
```

Описание

Цел(Выражение: {Целое|Число|Строка}): Целое;
Int(Expression: {Integer|Numeric|String}): Integer;

Аргументы

Выражение - числовое выражение или строка, которые требуется преобразовать в целое число.

Назначение

Преобразует значение переданного параметра в целое число, с использованием, при необходимости, стандартных правил округления. Если заданная строка не является правильной записью числа или тип параметра не соответствует допустимому типу аргумента, функция генерирует исключение.

Пример

```
proc P1;  
  var Дано : Целое;  
  Дано = Int(100/3); -- даст результат 33  
  Дано = Цел (49.125); -- Дано = 49  
  Дано = Цел ( "49.125" ); -- Дано = 49  
  Дано = Цел ( "49.12S" ); -- сообщение об ошибке  
end;
```

Описание

```
Числ(Выражение:{Целое|Число|Строка}): Число;  
Num(Expression:{Integer|Numeric|String}): Numeric;
```

Аргументы

Выражение - числовое выражение или строка, которые требуется преобразовать в действительное число.

Назначение

Преобразует значение переданного параметра в действительное число. Если заданная строка не является правильной записью числа или тип параметра не соответствует допустимому типу аргумента, функция генерирует исключение.

Пример

```
proc P1;  
  var Дано : Число;  
  Дано = КакЧисло ( "49.125" ); -- Дано = 49.125  
  Дано = КакЧисло ( "49.12S" ); -- сообщение об ошибке  
end;
```

Описание

```
Эксп(Выражение:Число): Число;  
Exp(Expression:Numeric): Numeric;
```

Аргументы

Выражение - числовое выражение действительного типа (или просто число), для которого необходимо вычислить экспоненту (е в степени).

Назначение

Вычисляет значение экспоненты (показательной функции с основанием е) для заданного аргумента.



Пример

```
proc P1;  
  var Дано : Число;  
  Дано = Эксп ( 1 );  
  -- Дано = 2.71828182845905  
end;
```

Для ведения бухгалтерского учета в классе **Бухгалтерия / Books** используются следующие процедуры и функции:

-  [Функция ВзятьПеременную / GetVariable / Общ Пер / Com Var](#)
-  [Функция ВнутриБухИзоляции / InBooksIsolation](#)
-  [Функция Деб / Deb](#)
-  [Процедура ЗавершитьБухИзоляцию / EndBooksIsolation](#)
-  [Функция ЗначениеИзмерителя / UnitValue](#)
-  [Функция Кре / Cre](#)
-  [Функция НаимОперации / OperationName / Наим О / Name О](#)
-  [Функция НаимСчета / AccountName / Наим С / Name С](#)
-  [Процедура НачатьБухИзоляцию / BeginBooksIsolation](#)
-  [Функция Оборот / Turn](#)
-  [Функция ОборотДоп / TurnEx](#)
-  [Процедура Обработать / Refresh](#)
-  [Функция Остаток / Saldo](#)
-  [Функция ОстатокДоп / SaldoEx](#)
-  [Функция Перевести / Convert](#)
-  [Функция ПоказательИзмерителя / UnitFactor](#)
-  [Процедура ПолучитьПостоянноеУсловие / GetPermanentCondition](#)
-  [Функция ПравильныйПризнак / ValidSign](#)
-  [Функция ПроверитьУсловиеНаПараметры / CheckParamCond](#)
-  [Функция ПроверитьУсловиеНаСчета / CheckAccCond](#)
-  [Процедура ПровестиПолупроводку / PassHalfTrans](#)
-  [Процедура ПровестиПроводку / PassTransaction](#)
-  [Функция РаздДебОст / SepDebSal](#)
-  [Функция РаздКреОст / SepCreSal](#)
-  [Процедура РаздОстаток / SeparateSaldo](#)
-  [Процедура РаздОстатокДоп / SeparateSaldoEx](#)
-  [Процедура СброситьДанные / ResetBooks](#)
-  [Функция СреднийОстаток / AverageSaldo](#)
-  [Функция СреднийОстатокДоп / AverageSaldoEx](#)
-  [Функция СчетПоИмени / GetAccountByName](#)
-  [Функция ТекущаяОбластьУчета / CurrentDomain](#)
-  [Функция ТекущийЖурнал / CurrentJournal](#)
-  [Процедура УстановитьПеременную / SetVariable](#)
-  [Процедура УстановитьПостоянноеУсловие / SetPermanentCondition](#)

и перечислимые типы:

-  [Тип УровеньОбработки / RefreshLevel](#)
-  [Тип ПериодыОгрубления / CoarsePeriodTypes](#)

Перечислимый тип *ПериодыОгрубления / CoarsePeriodTypes*, определенный в классе *Бухгалтерия* предназначен для задания временного периода, используемого для расчета среднего остатка вычислимого показателя в отчетах.

В данном типе определены следующие константы, позволяющие считать все движения за заданные периоды времени:

- **byTrans** - по проводкам, т.е. средний остаток подсчитывается с точностью до даты проводки. Это значение задано по умолчанию, в том случае, когда в функции [СреднийОстаток](#) не задан параметр **ПериодОгрубления**.
- **byMinute** - за минуту;
- **byDay** - за день;
- **byWeek** - за неделю;
- **byMonth** - за месяц;
- **byQuart** - за квартал;
- **byYear** - за год.

Внимание. При установке периода огрубления надо иметь в виду, что период огрубления не должен превышать периода построения отчета или периода разбиения по времени при его наличии.

Константы, определенные в данном типе, используются в методе [СреднийОстаток](#) класса *Бухгалтерия*.

В классе *Бухгалтерия* используется перечислимый тип *УровеньОбработки / RefreshLevel*, содержащий набор констант, определяющих объем обработки учетных данных при выполнении [бухгалтерских изоляций](#).

Константы данного типа применяются в методе [НачатьБухИзоляцию](#) класса *Бухгалтерия*.

В типе *УровеньОбработки* определены следующие константы:

- **ОбработатьСчета / RefreshAccs** - обрабатываются только счета;
- **ОбработатьСправочники / RefreshReferences** - обрабатываются только справочники аналитических признаков, валют, единиц измерения;
- **ОбработатьПеременные / RefreshVariables** - обрабатываются только общие переменные;
- **ОбработатьЖурналы / RefreshJournals** - обрабатываются только журналы;
- **ОбработатьВсе / RefreshAll** - обрабатывается весь объем данных (справочники, переменные, журналы, счета).

Описание

```
ПолучитьПостоянноеУсловие({ОбластьУчета :Строка}; {var УсловиеНаСчета :Строка}; {var  
УсловиеНаПараметры :Строка});  
GetPermanentCondition ({Domain :String}; {var AccCondition :String}; {var  
ParamCondition :String});
```

Назначение

Процедура позволяет получить информацию об ограничениях, наложенных на область учета, счета и дополнительные параметры.

Аргументы

ОбластьУчета - необязательный параметр, задающий область учета.

УсловиеНаСчета - необязательный параметр типа **Строка**, в который после выполнения процедуры заносится условие на бухгалтерские счета.

УсловиеНаПараметры - необязательный параметр типа **Строка**, в который после выполнения процедуры заносится условие на дополнительные параметры.

См. также процедуру [УстановитьПостоянноеУсловие](#).

Описание

```
ЗавершитьБухИзоляцию;  
EndBooksIsolation;
```

Назначение

Закрывает изоляцию, ранее открытую на машине проводок с помощью процедуры **НачатьБухИзоляцию**. Вызовы этих двух процедур должны быть парными, то есть на каждый вызов **НачатьБухИзоляцию** должен приходиться один вызов **ЗавершитьБухИзоляцию**. Каждый вызов **НачатьБухИзоляцию** увеличивает внутренний счетчик изоляций на 1 (его начальное значение, когда изоляция не открыта, равен 0), а вызов **ЗавершитьБухИзоляцию** этот счетчик уменьшает на 1. Изоляция прекращает действовать, когда счетчик вновь становится равным 0.

Если делается попытка закрыть несуществующую изоляцию, генерируется ошибка.

Пример

```
-- проверяем наличие открытой изоляции  
if ВнутриБухИзоляции = True then  
    -- если изоляция была открыта, ее можно закрыть  
    ЗавершитьБухИзоляцию;  
end;
```

Описание

```
НачатьБухИзоляцию( {Уровень : Бухгалтерия.УровеньОбработки}; {Область : Строка};
{Додаты : Дата});
BeginBooksIsolation( {Level : Books.RefreshLevel}; {Domain : String}; {UpToDate : Date});
```

Аргументы

Уровень - одна из предопределенных [констант](#), задающая объем обработки учетных данных (аналитических признаков, переменных, журналов или всех вместе);

Область - [области учета](#), по которой необходимо начать изоляцию. Если аргумент опущен или указана пустая строка, производится обработка области *default*, используемой по умолчанию;

Додаты - дата, вплоть до которой необходимо обработать учетные данные (имеет смысл, только если задана обработка журналов или всех данных).

Любой из аргументов или сразу все аргументы могут быть опущены при вызове процедуры. Если опущен первый аргумент, производится полная обработка, Если опущен третий - обработка ведется за все даты.

Назначение

Открывает [изоляцию](#) на машине проводок (бухгалтерском ядре системы или сервере расчетов). После вызова данной процедуры система гарантирует, что все последующие обращения клиента к бухгалтерским функциям работают с одномоментным слепком учетных данных. Если другие клиенты изменяют учетные данные на сервере расчетов, это никак не сказывается на клиенте, открывшем изоляцию. Тем самым обеспечивается непротиворечивость анализируемых данных. При выполнении данной процедуры сервер, при необходимости, производит так называемое связывание журналов - виртуальное объединение всех источников учетной информации и хронологическое обновление показателей, допустимых для описанных в проекте счетов (обороты и остатки, свернутые или разделенные, возможно, в разрезе различных аналитических признаков и т.д.).

Действие изоляции, начатое с помощью вызова **НачатьБухИзоляцию**, завершается по вызову процедуры [ЗавершитьБухИзоляцию](#). Вызовы этих двух процедур должны быть парными, то есть на каждый вызов **НачатьБухИзоляцию** должен приходиться один вызов **ЗавершитьБухИзоляцию**. Каждый вызов **НачатьБухИзоляцию** увеличивает внутренний счетчик изоляций на 1 (его начальное значение, когда изоляция не открыта, равен 0), а вызов **ЗавершитьБухИзоляцию** этот счетчик уменьшает на 1. Изоляция прекращает действовать, когда счетчик вновь становится равным 0.

Следует иметь в виду, что открытая изоляция отнюдь не означает, что учетные данные не обновляются на сервере машины проводок - изменения отслеживаются и обрабатываются постоянно, причем их текущее состояние доступно всем клиентам, которые не открыли в данный момент изоляции. Таким образом, изоляция дает возможность клиенту временно отказаться от получения обновлений учетных данных.

Использовать изоляцию имеет смысл лишь в том случае, если требуется последовательно вызвать несколько логически связанных бухгалтерских методов (показатели которых должны соотноситься друг с другом). Одиночные вызовы бухгалтерских методов не имеет смысл заключать в изоляцию, поскольку сервер машины проводок при выполнении таких операций автоматически открывает и закрывает изоляцию внутри себя.

При активации бухгалтерской изоляции система автоматически открывает изоляцию на записи всех классов документов проекта. Это эквивалентно тому, как если бы внутри метода **НачатьБухИзоляцию** производился вызов метода [НачатьИзоляцию](#) с перечислением всех классов документов. Автоматически открытая изоляция по записям базы данных завершается синхронно с завершением бухгалтерской изоляции.

Пример

```
proc P1;
  var A, B, В : Число;
  -- обеспечиваем неизменность учетных данных и показателей
  -- для последующих вызовов функции Остаток
  НачатьБухИзоляцию;
  A = Остаток ( "62", "Сумма^Руб", "Пост=П.1", Сегодня);
  -- A - остаток по 62 счету для покупателя П.1 на сегодня
  B = Остаток ( "62", "Сумма^Руб", "Пост=П.2", Сегодня);
```

```
-- Б - остаток по 62 счету для покупателя П.2 на сегодня
ЗавершитьБухИзоляцию;
В = А + Б;
end;
```

Описание

```
Обработать (Область :Строка [; Все:Логическое]);  
Refresh (Domain :String [; All:Logical]);
```

Аргументы

Область - строка с идентификатором области учета, по которой необходимо произвести обработку. Если указана пустая строка, производится обработка области *default*, используемой по умолчанию. Можно также указать в качестве области учета строку "*" - в этом случае будут обработаны все области учета, присутствующие в базе;

Все - необязательный аргумент, определяет, нужно ли обработать все данные области (ИСТИНА) или только изменившиеся (ЛОЖЬ). Когда аргумент опущен, обрабатываются только изменившиеся данные области (эквивалент команды **Учет|Обработать**). Если передается значение ИСТИНА, обрабатывается всё (эквивалент команды **Учет|Обработать Все**).

Назначение

Запускает процесс обработки данных учета.

Пример

```
Обработать ( " ", ИСТИНА );
```

Описание

```
ПровестиПолупроводку(Счет :Счет; Данные : Полупроводка; ПоДебету :Логическое;  
{ВнПарам :ХранилищеПараметров});
```

```
PassHalfTrans(Acc :Account; Data :HTrans; ByDebit :Logical; {ExParams :ParamStorage});
```

Аргументы

Счет - объект класса, производного от класса **Счет**;

Данные - объект класса, производного от **Полупроводка**, с конкретными значениями параметров счета;

ПоДебету - признак того, проводится ли полупроводка по дебету или кредиту; значение true соответствует полупроводке по дебету, false - по кредиту;

ВнПарам - необязательный параметр для добавления свободной аналитики типа [ХранилищеПараметров](#).

Назначение

Процедура проводит новую полупроводку, используя предоставленную с помощью аргументов информацию. Особенности использования данной процедуры совпадают с рассмотренными для процедуры [ПровестиПроводку](#).

Пример

```
proc СформироватьПолупроводку(Сумма :Число; Валюта: спрВалюта);  
  var h1: Полупроводка_счетУпр;  
  h1 = Полупроводка_счетУпр.Создать;  
  h1.Сумма = Сумма^Валюта;  
  h1.Колво = Сколько^Единица;  
  ПровестиПолупроводку(Баланс.01, h1, true);  
end;
```

Описание

```
ПровестиПроводку( Деб :Счет; Кре : Счет; ДебДата : Полупроводка; КреДата : Полупроводка;
{ВнПарам :ХранилищеПараметров});
```

```
PassTransaction(Deb :Account; Cre :Account; DebData :HTrans; CreData :HTrans;
{ExParams :ParamStorage});
```

Аргументы

Деб - счет дебета; объект класса, производного от класса **Счет**;

Кре - счет кредита; объект класса, производного от класса **Счет**;

ДебДата - объект класса, производного от **Полупроводка**, с конкретными значениями параметров счета дебета;

КреДата - объекта класса, производного от **Полупроводка**, с конкретными значениями параметров счета кредита;

ВнПарам - необязательный параметр для добавления свободной аналитики типа [ХранилищеПараметров](#).

Назначение

Процедура проводит новую проводку, используя предоставленную с помощью аргументов информацию.

Класс счета аргумента **Деб** и класс счета, породивший класс полупроводки **ДебДата**, должны быть связаны отношением наследования. То же самое относится и к паре аргументов **Кре** и **КреДата**.

В том случае, если тип счета переданной полупроводки (**ДебДата** или **КреДата**) унаследован от типа счета формируемой проводки (**Деб** или **Кре**), то данных содержащихся в объекте полупроводки достаточно для того, чтобы провести проводку. При наличии обратного отношения родства провести проводку возможно только в том случае, если параметры, имеющиеся в классе счета, но отсутствующие в классе счета полупроводки, имеют значения по умолчанию.

Например, в структуре учета описан тип счетов *Базовый* с параметром *Сумма*. На основе этого описания система автоматически генерирует класс **Базовый**, производный от класса **Счет**, и класс **Полупроводка_Базовый**, производный от класса **Полупроводка**, причем в классе **Полупроводка_Базовый** содержится свойство **Сумма**.

Пусть также описан типа счета *Количественный*, производный от типа *Базовый*, и в этом типе дополнительно определен параметр *Количество*. На основе этого описания система автоматически генерирует класс **Количественный**, производный от класса **Базовый**, и класс **Полупроводка_Количественный**, производный от класса **Полупроводка_Базовый**, причем в классе **Полупроводка_Количественный** имеется как свойство **Сумма**, так и свойство **Количество**.

При вызове процедуры **ПровестиПроводку** мы можем передавать в нее следующие параметры (только на примере дебетовой части):

Деб	ДебДата	Пояснения
объект класса Базовый	объект класса Полупроводка_Базовый	ОК
объект класса Базовый	объект класса Полупроводка_Количественный	ОК; данные о количестве не используются
объект класса Количественный	объект класса Полупроводка_Базовый	Внимание! Проводка может быть сформирована, только если параметр Количество имеет значение по умолчанию, т.к. в противном случае неоткуда взять информацию о количестве - в объекте полупроводки нет данных о количестве
объект класса Количественный	объект класса Полупроводка_Количественный	ОК

Пример

```
proc СформироватьПроводку(Сумма :Число; Валюта: sprВалюта;
Сколько :Число; Единица :sprЕдиницы);
var h1, h2: Полупроводка_счетУпр;
```

```
h1 = Полупроводка_счетУпр.Создать;  
h2 = Полупроводка_счетУпр.Создать;  
h1.Сумма = Сумма^Валюта;  
-- количество только по дебету  
h1.Колво = Сколько^Единица;  
h2.Сумма = -Сумма^Валюта;  
ПровестиПроводку (Баланс.01, Баланс.02, h1, h2);  
end;
```

Описание

РаздОстаток (УсловиеОтбораНаСчета:Строка; Показатель:Строка;
УсловиеОтбораПоПараметрам:Строка; Разделитель:Строка; [День:Дата]; перем Дебет:Число; перем
Кредит:Число);
SeparateSaldo (AccCondition:String; Factor:String; ParamCondition:String; Separator:String;
[DateX:Date]; var Debet:Numeric; var Credit:Numeric);

Аргументы

УсловиеОтбораНаСчета - условие отбора [проводок по счетам](#), записанное в стандартном виде;

Показатель - [строка, задающая измеритель](#), в котором будет выдаваться результат;

УсловиеОтбораПоПараметрам - записанное в стандартном виде условие отбора [проводок по параметрам счетов](#);

Разделитель - строка, содержащая имя (или несколько имен через пробелы) параметров счетов (с аналитикой), относительно которой остатки будут разноситься на дебет и кредит;

День - дата, на которую требуется рассчитать остатки;

Дебет - переменная, принимающая дебетовый остаток;

Кредит - переменная, принимающая кредитовый остаток.

Назначение

Вычисляет дебетовый и кредитовый остаток по счету или группе счетов, заданных условиями отбора на указанную дату. Разделение заданного показателя на дебет и кредит производится в разрезе справочника (справочников), имя которого указано в параметре **Разделитель** (данный справочник должен использоваться в качестве типа некоторого параметра счетов).

Если дата опущена, то остаток считается по-разному в зависимости от того, откуда вызвана процедура. Если процедура вызвана из бланка, остаток считается за весь учетный период (т.е. конечная дата берется равной дате, следующей за датой последней существующей проводки). В типовых операциях возвращается остаток на дату, которой проведена типовая операция.

При вызове процедуры из типовых операций, если задана дата более поздняя, нежели дата операции, остаток считается на дату операции.

Пример

```

прос P1;
  var D : Число;
  var C : Число;
  РаздОстаток ( "60", "Сумма^Руб", "Регион=Москва", "Поставщик", , D, C );
  -- раздельное сальдо в суммовом выражении, в рублях,
  -- по 60 счету, по справочнику поставщиков, по операциям
  -- с московскими контрагентами, на конец учетного периода
  trace("Дебет="+Str(D));
  trace("Кредит="+Str(C));
end;
```


Описание

```

РаздОстатокДоп (УсловиеОтбораНаСчета:Строка; Показатель:Строка;
УсловиеОтбораПоПараметрам:Строка; Разделитель:Строка; [День:Дата]; перем Дебет:Измеритель;
перем Кредит:Измеритель);
SeparateSaldoEx (AccCondition:String; Factor:String; ParamCondition:String;
Separator:String; [DateX:Date]; var Debet:Unit; var Credit:Unit);

```

Аргументы

УсловиеОтбораНаСчета - условие отбора [проводок по счетам](#), записанное в стандартном виде;

Показатель - [строка, задающая измеритель](#), в котором будет выдаваться результат;

УсловиеОтбораПоПараметрам - записанное в стандартном виде условие отбора [проводок по параметрам счетов](#);

Разделитель - строка, содержащая имя (или несколько имен через пробелы) параметров счетов (с аналитикой), относительно которой остатки будут разноситься на дебет и кредит;

День - дата, на которую требуется рассчитать остатки;

Дебет - переменная, принимающая дебетовый остаток;

Кредит - переменная, принимающая кредитовый остаток.

Назначение

Вычисляет дебетовый и кредитовый остаток по счету или группе счетов, заданных условиями отбора на указанную дату. Разделение заданного показателя на дебет и кредит производится в разрезе справочника (справочников), имя которого указано в параметре **Разделитель** (данный справочник должен использоваться в качестве типа некоторого параметра счетов).

Процедура аналогична процедуре [РаздОстаток](#), однако, в отличие от нее, возвращает через параметры **Дебет** и **Кредит** не числовые значения, а измерители (число и единицу измерения). Если в результате работы функции оказалось, что запрошенный показатель составляют значения с различными единицами измерения (например, остатки в различных единицах), то фактор возвращаемого значения измерителя (получаемый с помощью функции **ПоказательИзмерителя / UnitFactor**) будет равен **nil**.

Если дата опущена, то остаток считается по-разному в зависимости от того, откуда вызвана процедура. Если процедура вызвана из бланка, остаток считается за весь учетный период (т.е. конечная дата берется равной дате, следующей за датой последней существующей проводки). В типовых операциях возвращается остаток на дату, которой проведена типовая операция.

При вызове процедуры из типовых операций, если задана дата более поздняя, нежели дата операции, остаток считается на дату операции.

Пример

```

proc P1;
  var D : Измеритель;
  var C : Измеритель;
  РаздОстаток ( "60", "Сумма^", "Регион=Москва", "Поставщик", , D, C );
  -- раздельное сальдо в суммовом выражении, в натуральных единицах,
  -- по 60 счету, по справочнику поставщиков, по
  -- операциям с московскими контрагентами, на конец учетного периода
  -- выводим дебет и кредит в формат: "<число> <валюта>"
  trace("Дебет="+Str(UnitValue(D))+ " "+Str(UnitFactor(D).Name));
  trace("Кредит="+Str(UnitValue(C))+ " "+Str(UnitFactor(C).Name));
end;

```

Описание

СброситьДанные ;
ResetBooks ;

Назначение

Процедура освобождает память для сервера расчетов, производит сброс пулов и кешей машины проводок (действие, аналогичное тому, что получается при указании в настройках параметров обработки: Область учёта - Все, Сбросить - Всё, Обработать - Ничего).

Замечание

Перед производением каких-либо расчетов, можно выполнить данную процедура для освобождения памяти.

Описание

```
УстановитьПеременную (ИмяПерем:Строка ;[ День:Дата]; Значение: Вариант);  
SetVariable (VarName:String ;[ADay:Date]; Value: Variant);
```

Аргументы

ИмяПерем - имя общей переменной, определенной в структуре учета;

День - произвольная дата;

Значение - значение, которое требуется присвоить данной переменной на указанную дату.

Назначение

Присваивает общей переменной произвольного типа (с именем, переданным через **ИмяПерем**) конкретное значение. Через необязательный аргумент **День** передается дата, для которой должно быть установлено значение переменной. Если он не задан, то значение записывается за дату операции, в которой эта процедура используется в журнале, или за системную дату, если она используется в бланках.

Пример

```
проц Р1(ДеньИкс: Дата);  
    УстановитьПеременную( "СтавкаНП_1", ДеньИкс, 2);  
    -- Ставка налога с продаж равна 2, начиная с даты ДеньИкс  
end;
```

Описание

```
УстановитьПостоянноеУсловие( [ОбластьУчета :Строка]; [УсловиеНаСчета :Строка];  
[УсловиеНаПараметры :Строка] );  
SetPermanentCondition ([Domain :String]; [AccCondition :String]; [ParamCondition :String ]);
```

Аргументы

ОбластьУчета - необязательный параметр строкового типа, через который передается область бухгалтерского учета;

УсловиеНаСчета - необязательный параметр строкового типа, через который передается условие на бухгалтерские счета;

УсловиеНаПараметры - необязательный параметр строкового типа, через который передается условие на дополнительные параметры.

Назначение

Процедура позволяет устанавливать ограничения на область учета, счета и дополнительные параметры.

Пример

```
SetPermanentCondition('Раздел_Баланс','Баланс: not 10.1/not 10.1', 'Сумма=руб');
```

См. также процедуру [ПолучитьПостоянноеУсловие](#).

Описание

```
СреднийОстатокДоп( УсловиеНаСчета :Строка; Показатель :Строка; {УсловиеНаПараметры :Строка};  
{Дата1 :Дата}; {Дата2 :Дата} ) :Измеритель;  
AverageSaldoEx (AccCondition :String; Factor :String; [ParamCondition :String];  
[Date1 :Date]); [Date2 :Date] ) :Измеритель;
```

Аргументы

УсловиеНаСчета - условие отбора [проводок по счетам](#), записанное в стандартном виде;

Показатель - [строка, задающая измеритель](#);

УсловиеНаПараметры - необязательный параметр, условие отбора проводок [по параметрам счетов](#);

Дата1 - необязательный параметр, дата начала периода;

Дата2 - необязательный параметр, дата конца периода.

Назначение

Функция возвращает средний остаток типа **Измеритель** в заданном промежутке времени с учетом дополнительных условий (на счета, показатели, параметры).

См. также функцию [СреднийОстаток](#).

Описание

ТекущаяОбластьУчета :Строка;
CurrentDomain :String;

Назначение

Функция возвращает текущую область учета в строковом виде.

Пример

```
var ТекОблУчета :String;  
  
ТекОблУчета = ТекущаяОбластьУчета;
```

Формат строки показателя:

`<ИмяПараметраСчета> ^ <ИмяЕдиницыИзмерения>`

Например, "Сумма^USD", если у счетов был описан параметр **Сумма** типа **измеритель Валюта**, а в справочнике валют имеется элемент с идентификатором **USD**.

В качестве имени единицы измерения можно также использовать полный DocID документа, являющегося источником аналитического признака этой единицы измерения (например, запись из справочника валют, в которой описывается доллар).

Формат описания DocID:

`{<ИмяКлассаДокумента>:<Значение поля DocID>}`

Например, "{Test.Справочники.Валюта:2}",

где 2 - DocID записи о **USD**, то есть формат описания DocID имеет тот же вид, что и для свойства [DocIDStr](#).

Для удобства можно использовать в выражениях функцию **Str**, например, "Сумма^Str(Doc)",

где Doc - объект класса **Запись** или производного.

Полностью значение показателя записывается в обобщенном виде:

`Показатель = <ИмяПараметраСчета> ^ <ИмяЕдиницыИзмерения>`

`ИмяЕдиницыИзмерения = <ИмяИзмерителя> | <КлючДокументаКакСтрока>`

Для запроса значений в натуральных единицах необходимо указать показатель, опустив имя единицы измерения. При этом символ '^' можно оставить, а можно также опустить. Таким образом результаты в натуральных единицах получаются, если параметр **Показатель** имеет вид:

`<ИмяПараметраСчета> ^`

или просто

`<ИмяПараметраСчета>`

Для запроса значений в базовых единицах необходимо вместо имени единицы измерения указать звездочку '*', например, "Сумма^*".

Описание

```
ВзятьПеременную (ИмяПерем:Строка [; День:Дата] ): Вариант;  
GetVariable (VarName:String [;ADay:Date]): Variant;  
Общ_Пер (ИмяПерем:Строка [; День:Дата] ): Вариант;  
Com_var (VarName:String [;ADay:Date]): Variant;
```

Аргументы

ИмяПерем - имя переменной, определенной в структуре учета;

День - произвольная дата.

Назначение

Возвращает значение общей переменной произвольного типа с именем **ИмяПерем**. Через необязательный аргумент **День** передается дата, для которой должно быть взято значение переменной. Если он не задан, то значение берется за дату операции, в которой эта функция используется в журнале, или за системную дату, если она используется в бланках.

Пример

```
proc P1;  
  var ОбщПерем : Число;  
  ОбщПерем = Общ_Пер( "СтавкаНП_1", 16.02.2000);  
  -- ОбщПерем = 4  
end;
```


Описание

ВнутриБухИзоляции :Логическое;
InBooksIsolation :Logical;

Назначение

Возвращает True, если изоляция на машине проводок была открыта (с помощью процедуры **НачатьБухИзоляцию**) и в данный момент действует. Возвращает False, если открытых изоляций в данный момент нет.

Пары вызовов **НачатьБухИзоляцию** и **ЗавершитьБухИзоляцию** могут быть вложенными. Каждый вызов **НачатьБухИзоляцию** увеличивает внутренний счетчик изоляций на 1 (его начальное значение, когда изоляция не открыта, равен 0), а вызов **ЗавершитьБухИзоляцию** этот счетчик уменьшает на 1. Изоляция прекращает действовать, когда счетчик вновь становится равным 0. Таким образом, функция **ВнутриБухИзоляции** возвращает True, если внутренний счетчик изоляций больше нуля.

Пример

```
-- проверяем наличие открытой изоляции
if ВнутриБухИзоляции = True then
    -- если изоляция была открыта, ее можно закрыть
    ЗавершитьБухИзоляцию;
end;
```

Описание

```
Деб (Сумма:Число): Число;  
Deb (Sum:Numeric): Numeric;
```

Аргументы

Сумма - числовое выражение.

Назначение

Возвращает значение своего параметра, если оно неотрицательно, или же ноль в противном случае. Используется для вычисления дебетовых остатков счетов.

Пример

```
proc P1;  
  var Д : Число;  
  Д = Деб(5);    -- Д = 5  
  Д = Деб(0);    -- Д = 0  
  Д = Деб(-3);   -- Д = 0  
end;
```

Описание

```
ЗначениеИзмерителя(Дано :Измеритель): Число;  
UnitValue(Given :Unit): Numeric;
```

Аргументы

Дано - переменная, константа или выражение типа **Измеритель** (или **Измеритель <Конкретный справочник аналитики>**, например, "Измеритель Валюта").

Назначение

Возвращает числовую составляющую измерителя, переданного в функцию через аргумент **Дано**.

Пример

```
var V : Unit Валюта;  
V = 100^DEM;  
trace(str(UnitValue(V))); -- выведет "100"
```

Описание

Кре (Сумма:Число): Число;
Cre (Sum:Numeric): Numeric;

Аргументы

Сумма - числовое выражение.

Назначение

Возвращает взятое по модулю значение своего параметра, если оно отрицательно, в противном случае - ноль. Используется для вычисления кредитовых остатков счетов.

Пример

```
proc Pl;  
  var Кр : Число;  
  Кр = Кре(5);  -- Кр = 0  
  Кр = Кре(0);  -- Кр = 0  
  Кр = Кре(-3); -- Кр = 3  
end;
```

Описание

```
НаимОперации (Оп:Строка): Строка;  
OperationName (Op:String): String;
```

Аргументы

Оп - краткое обозначение (идентификатор) типовой операции.

Назначение

Возвращает наименование типовой операции по ее идентификатору. Фактически, это строка, указанная в кавычках в заголовке класса типовой операции.

Пример

```
var Назв : Строка;  
-- Наименование операции  
Назв = НаимОперации ( "ЗАРПЛ_НАЧ" );  
-- Назв = "Начисление заработной платы"
```

Описание

```
НаимСчета (Сч:Строка): Строка;  
AccountName (Ac:String): String;
```

Аргументы

Сч - номер счета.

Назначение

Возвращает наименование счета по его идентификатору. Фактически, это строка, указанная с помощью ключевого слова **Название (Title)** при описании счета в структуре учета.

Пример

```
proc P1;  
  var Назв, Имя : Строка;  
  Имя = ВыборСчета;  
  Назв = НаимСчета( Имя );  
  Сообщение ( "Наименование счета - " + Назв );  
end;
```

Описание

Оборот (УсловиеОтбораНаСчета:Строка; Показатель:Строка; УсловиеОтбораПоПараметрам:Строка; [ДатаНач:Дата]; [ДатаКон:Дата]):Число;
Turn (AccCondition:String; Factor:String; ParamCondition:String; [Date1:Date]; [Date2:Date]): Numeric;

Аргументы

УсловиеОтбораНаСчета - условие отбора [проводок по счетам](#), записанное в стандартном виде;

Показатель - [строка, задающая измеритель](#), в котором будет выдаваться результат;

УсловиеОтбораПоПараметрам - записанное в стандартном виде условие отбора [проводок по параметрам счетов](#);

ДатаНач - дата начала периода;

ДатаКон - дата конца периода.

Назначение

Вычисляет оборот счета или группы счетов, заданных условиями отбора, за период от начальной до конечной даты (не включая ее).

Если опущена начальная дата, то обороты считаются от начала учета (начиная с даты первой проводки). Если опущена конечная дата, то обороты считаются по-разному в зависимости от того, откуда вызвана функция. Если функция вызвана из бланка, обороты считаются за весь учетный период (т.е. конечная дата берется равной дате, следующей за датой последней существующей проводки). В типовых операциях возвращается оборот на дату, которой проведена типовая операция.

При вызове функции из типовых операций, если задана конечная дата более поздняя, нежели дата операции, оборот считается на дату операции.

Пример

```
Итог = Оборот ( "20", "Сумма^Руб", "Пост=П.1", 01.01.2000, 01.04.2000);  
-- Переменная Итог равна обороту 20 счета с учетом признака  
-- П.1 за период от 01.01.2000 по 31.03.2000 включительно.
```

Описание

ОборотДоп (УсловиеОтбораНаСчета:Строка; Показатель:Строка; УсловиеОтбораПоПараметрам:Строка; [ДатаНач:Дата]; [ДатаКон:Дата]) :Измеритель;
TurnEx (AccCondition:String; Factor:String; ParamCondition:String; [Date1:Date]; [Date2:Date]) :Unit;

Аргументы

УсловиеОтбораНаСчета - условие отбора [проводок по счетам](#), записанное в стандартном виде;

Показатель - [строка, задающая измеритель](#), в котором будет выдаваться результат;

УсловиеОтбораПоПараметрам - записанное в стандартном виде условие отбора проводок [по параметрам счетов](#);

ДатаНач - дата начала периода;

ДатаКон - дата конца периода.

Назначение

Вычисляет оборот счета или группы счетов, заданных условиями отбора, за период от начальной до конечной даты (не включая ее).

Аналогична функции [Оборот](#), однако, в отличие от нее, возвращает не числовое значение, а измеритель (число и единица измерения). Если в результате работы функции оказалось, что запрошенный показатель составляют значения с различными единицами измерения (например, перечисления сумм в различных валютах), то фактор возвращаемого значения измерителя (получаемый с помощью функции **ПоказательИзмерителя / UnitFactor**) будет равен **nil**.

Пример

```
var x: Измеритель;  
var Итог: Число;  
var Вал: Признак;  
-- запрашиваем сумму в натуральных единицах измерения  
x = ОборотДоп ( "20", "Сумма^", "Пост=П.1", 01.01.2005, 01.04.2005);  
Итог = ЗначениеИзмерителя(x);  
Вал = ПоказательИзмерителя(x);  
if Вал = nil then  
    trace('неопределенность с валютой: требуется разбивка по валютам');  
end;
```


Описание

Остаток (УсловиеОтбораНаСчета:Строка; Показатель:Строка; УсловиеОтбораПоПараметрам:Строка [; День:Дата]):Число;
saldo (AccCondition:String; Factor:String; ParamCondition:String [; DateX:Date]): Numeric;

Аргументы

УсловиеОтбораНаСчета - условие отбора [проводок по счетам](#), записанное в стандартном виде;

Показатель - [строка, задающая измеритель](#), в котором будет выдаваться результат;

УсловиеОтбораПоПараметрам - записанное в стандартном виде условие отбора [по параметрам счетов](#);

День - дата, на которую требуется рассчитать остатки.

Назначение

Вычисляет остаток по группе счетов, заданных условиями отбора, на указанную дату. Если дата опущена, то остаток считается по-разному в зависимости от того, откуда вызвана функция. Если функция вызвана из бланка, остаток считается за весь учетный период (т.е. конечная дата берется равной дате, следующей за датой последней существующей проводки). В типовых операциях возвращается остаток на дату, которой проведена типовая операция.

При вызове функции из типовых операций, если задана дата более поздняя, нежели дата операции, остаток считается на дату операции.

Пример

```
proc P1;
  var A, B : Число;
  A = Остаток ( "20", "Сумма^Руб", "Пост=П.1", Сегодня);
  -- A = остатку 20 счета для аналитического признака П.1 на сегодня
  B = Остаток ( "Баланс:68*", "Сумма^Руб", "" );
  -- B = остатку 68 счета и всех его субъектов за весь учетный период
end;
```

Описание

ОстатокДоп (УсловиеОтбораНаСчета:Строка; Показатель:Строка; УсловиеОтбораПоПараметрам:Строка [; День:Дата]):Измеритель;
SaldoEx (AccCondition:String; Factor:String; ParamCondition:String [; DateX:Date]): Unit;

Аргументы

УсловиеОтбораНаСчета - условие отбора [проводок по счетам](#), записанное в стандартном виде;

Показатель - [строка, задающая измеритель](#), в котором будет выдаваться результат;

УсловиеОтбораПоПараметрам - записанное в стандартном виде условие отбора проводок [по параметрам счетов](#);

День - дата, на которую требуется рассчитать остатки.

Назначение

Вычисляет остаток по группе счетов, заданных условиями отбора, на указанную дату. Если дата опущена, то остаток считается по-разному в зависимости от того, откуда вызвана функция.

Функция аналогична функции [Остаток](#), однако, в отличие от нее, возвращает не числовое значение, а измеритель (число и единицу измерения). Если в результате работы функции оказалось, что запрошенный показатель составляют значения с различными единицами измерения (например, остатки в различных единицах), то фактор возвращаемого значения измерителя (получаемый с помощью функции **ПоказательИзмерителя / UnitFactor**) будет равен **nil**.

Пример

```
proc P1;
  var A : Измеритель;
  -- запрашиваем количество в натуральных единицах
  A = ОстатокДоп ( "20", "Колво^*", "Пост=П.1", Сегодня);
  if ПоказательИзмерителя(A) = nil then
    trace('требуется разбивка по единицам измерения');
  end;
end;
```

Описание

Перевести (Значение:Число; ИзмИсх: Признак; ИзмЦель:Признак [; День:Дата]): Число;
Convert (Value:Numeric; UnitFrom: Sign; UnitTo:Sign [; Day:Date]): Numeric;

Аргументы

Значение - числовое значение, представленное в единицах **ИзмИсх**, которое необходимо перевести в единицы **ИзмЦель**;

ИзмИсх - исходная единица измерения;

ИзмЦель - целевая единица измерения; в нее выполняется пересчет;

День - дата, на которую требуется пересчитать одну единицу в другую.

Назначение

Переводит указанное значение из одной единицы измерения (**ИзмИсх**) в другую (**ИзмЦель**). Пересчет производится по соотношению единиц на указанную дату. Если дата опущена, подразумевается либо текущая системная дата (в бланках), либо текущая дата операции (в типовой операции, вызванной из журнала).

Фактические параметры **ИзмИсх** и **ИзмЦель** должны относиться к одному и тому же аналитическому справочнику и иметь непосредственную или транзитивную связь с некоторой общей единицей измерения.

В простейшем случае одна из единиц должна быть базовой для другой (например, единицы измерения веса). Иными словами, в справочнике должна быть достаточная информация о пересчете из одной единицы в другую.

Нельзя проводить конвертацию между несвязанными единицами измерения, такими как литры и сантиметры.

Пример

```
-- передаем число и некоторую единицу измерения веса
-- из справочника Вес, в котором имеется единица Грамм;
-- получаем эквивалент в граммах
func Gramm(X:numeric, U:Bec): numeric;
    -- переводим вес X в граммы
    return Convert(x,u,Вес.Грамм);
end;
/pre>
```

Описание

```
ПоказательИзмерителя(Дано :Измеритель): Признак;  
UnitFactor(Given :Unit): Sign;
```

Аргументы

Дано - переменная, константа или выражение типа **Измеритель** (или **Измеритель <Конкретный справочник аналитики>**, например, "Измеритель Валюта").

Назначение

Возвращает аналитический признак, входящий в состав измерителя, переданного в функцию через аргумент **Дано**.

Пример

```
var V : Unit Валюта;  
V = 100^DEM;  
trace(str(UnitFactor(V))); -- выведет "DEM"
```

Описание

ПравильныйПризнак (Признак:Строка): Логическое;
ValidSign (Sign:String): Logical;

Аргументы

Признак - произвольная строка.

Назначение

Проверяет, может ли строковое выражение **Признак** использоваться в качестве аналитического признака, и при положительном результате возвращает значение ИСТИНА, иначе - ЛОЖЬ.

Пример

```
proc Test1;  
  var Signld :Строка;  
  if Not(Input(Signld, "Введите строку") = cmOk) then  
    Exit;  
  fi;  
  if Not(ПравильныйПризнак(Signld)) then  
    -- в строке есть недопустимые символы  
    Message("Строку " + Signld +  
      "нельзя использовать как аналитический признак");  
  fi;  
end;
```

Описание

```
ПроверитьУсловиеНаПараметры(Условие:Строка):Логическое;  
CheckParamCond (Condition:String): Logical;
```

Аргументы

Условие - строка, содержащая условие отбора проводок [по параметрам](#).

Назначение

Проверяет, является ли заданное условие отбора синтаксически корректным. Если да, возвращает значение ИСТИНА, иначе - ЛОЖЬ

Пример

```
proc ButtonBuildReportPress(B:Button);  
  if CheckParamCond(EditParamsCondition.Text) = false then  
    Message("Неверное условие отбора по параметрам");  
    Return;  
  fi;  
  --...  
  -- построение отчета с условием по параметрам  
end;
```

Описание

ПроверитьУсловиеНаСчета (Условие:Строка): Логическое;
CheckAccCond (Condition:String): Logical;

Аргументы

Условие - строка, содержащая условие отбора [проводок по счетам](#).

Назначение

Проверяет, является ли заданное условие отбора синтаксически корректным. Если да, возвращает значение ИСТИНА, иначе - ЛОЖЬ.

Пример

```
proc ButtonBuildReportPress(B:Button);
  if CheckAccCond(EditAccountsCondition.Text) = false then
    Message("Неверное условие отбора по счетам");
    Return;
  fi;
  --...
  -- построение отчета с условием по счетам
end;
```

Описание

```
РаздДебОст (УсловиеОтбораНаСчета:Строка; Показатель:Строка;  
УсловиеОтбораПоПараметрам:Строка; Разделитель:Строка [; День:Дата]):Число;  
SepDebSal (AccCondition:String; Factor:String; ParamCondition:String; Separator:String [;  
DateX:Date]): Numeric;
```

Аргументы

УсловиеОтбораНаСчета - условие отбора [проводок по счетам](#), записанное в стандартном виде;

Показатель - [строка, задающая измеритель](#), в котором будет выдаваться результат;

УсловиеОтбораПоПараметрам - записанное в стандартном виде условие отбора проводок [по параметрам счетов](#);

Разделитель - строка, содержащая имя (или несколько имен через пробелы) параметров счетов (с аналитикой), относительно которой остатки будут разноситься на дебет и кредит;

День - дата, на которую требуется рассчитать остатки.

Назначение

Вычисляет дебетовый остаток по счету или группе счетов, заданных условиями отбора на указанную дату. Разделение заданного показателя на дебет и кредит производится в разрезе справочника (справочников), имя которого указано в параметре **Разделитель** (данный справочник должен использоваться в качестве типа некоторого параметра счетов).

Если дата опущена, то остаток считается по-разному в зависимости от того, откуда вызвана функция. Если функция вызвана из бланка, остаток считается за весь учетный период (т.е. конечная дата берется равной дате, следующей за датой последней существующей проводки). В типовых операциях возвращается остаток на дату, которой проведена типовая операция.

При вызове функции из типовых операций, если задана дата более поздняя, нежели дата операции, остаток считается на дату операции.

Пример

```
proc P1;  
  var A : Число;  
  A = РаздДебОст ( "60", "Сумма^Руб", "Регион=Москва", "Поставщик", Сегодня );  
  -- раздельное дебетовое сальдо в суммовом выражении, в рублях,  
  -- по 60 счету, по справочнику поставщиков, по  
  -- операциям с московскими контрагентами, на текущую дату  
end;
```


Описание

```
РаздКреОст (УсловиеОтбораНаСчета:Строка; Показатель:Строка;  
УсловиеОтбораПоПараметрам:Строка; Разделитель:Строка [; День:Дата]):Число;  
SepCreSal (AccCondition:String; Factor:String; ParamCondition:String; Separator:String [;  
DateX:Date]): Numeric;
```

Аргументы

УсловиеОтбораНаСчета - условие отбора [проводок по счетам](#), записанное в стандартном виде;

Показатель - [строка, задающая измеритель](#), в котором будет выдаваться результат;

УсловиеОтбораПоПараметрам - записанное в стандартном виде условие отбора [проводок по параметрам счетов](#);

Разделитель - строка, содержащая имя (или несколько имен через пробелы) параметров счетов (с аналитикой), относительно которой остатки будут разноситься на дебет и кредит;

День - дата, на которую требуется рассчитать остатки.

Назначение

Вычисляет кредитовый остаток по счету или группе счетов, заданных условиями отбора на указанную дату. Разделение заданного показателя на дебет и кредит производится в разрезе справочника (справочников), имя которого указано в параметре **Разделитель** (данный справочник должен использоваться в качестве типа некоторого параметра счетов).

Если дата опущена, то остаток считается по-разному в зависимости от того, откуда вызвана функция. Если функция вызвана из бланка, остаток считается за весь учетный период (т.е. конечная дата берется равной дате, следующей за датой последней существующей проводки). В типовых операциях возвращается остаток на дату, которой проведена типовая операция.

При вызове функции из типовых операций, если задана дата более поздняя, нежели дата операции, остаток считается на дату операции.

Пример

```
proc P1;  
  var A : Число;  
  A = РаздКреОст ( "60", "Сумма^Руб", "Регион=Москва", "Поставщик", Сегодня );  
  -- раздельное кредитовое сальдо в суммовом выражении, в рублях,  
  -- по 60 счету, по справочнику поставщиков, по  
  -- операциям с московскими контрагентами, на текущую дату  
end;
```

Описание

```
СреднийОстаток(УсловиеНаСчета :Строка; Показатель :Строка; {УсловиеНаПараметры :Строка};  
{Дата1 :Дата}; {Дата2 :Дата}); {ПериодОгрубления : Бухгалтерия.ПериодыОгрубления};  
{ИгнорироватьПустыеПериоды :Логическое}) :Число;
```

```
AverageSaldo (AccCondition :String; Factor :String; {ParamCondition :String}; {Date1 :Date};  
{Date2 :Date}; {CoarsePeriod : Books.CoarsePeriodTypes};  
{IgnoreEmptyPeriods :Logical}) :Numeric;
```

Аргументы

УсловиеНаСчета - условие отбора [проводок по счетам](#), записанное в стандартном виде;

Показатель - строка, задающая [измеритель](#);

УсловиеНаПараметры - необязательный параметр, условие отбора проводок [по параметрам счетов](#);

Дата1, Дата2 - необязательные параметры, дата начала и конца периода;

ПериодОгрубления - константа, определяющая временной период (за минуту, за день и т. д.), за который подсчитывается средний остаток вычислимого показателя в отчетах. Если параметр не задан, то по умолчанию его значение равно byTrans, т.е. средний остаток подсчитывается с точностью до даты проводки;

ИгнорироватьПустыеПериоды - необязательный логический параметр, определяющий, следует ли игнорировать пустые периоды. По умолчанию принимает значение False. т.е. пустые периоды не игнорируются. Если параметр установлен в True, то периоды времени, в течение которых остаток равен 0 (с учетом точности единицы измерения), исключаются из общей суммы времени.

Внимание. При установке периода огрубления надо иметь в виду, что период огрубления не должен превышать периода построения отчета или периода разбиения по времени при его наличии.

Назначение

Функция возвращает средний остаток типа **Число** в заданном промежутке времени с учетом условий на счета, показатели и параметры.

См. также функцию [СреднийОстатокДоп.](#)

Описание

```
СчетПоИмени (Имя :Строка): Счет;  
GetAccountByName (Name :String): Account;
```

Аргументы

Имя - идентификатор счета, описанного в структуре учета.

Назначение

Возвращает объект класса **Счет**, содержащий ссылку на объект производного конкретного класса (типа) счетов. Например, если в структуре учета был описан тип счетов "Базовый", и на его основе - счет "60", то с помощью функции **СчетПоИмени** можно получить объект счета 60 класса Базовый.

Если в функцию передается несуществующий идентификатор счета, функция возвращает значение **nil (пусто)**.

Пример

```
var A : TradeAccount;  
-- тип счетов TradeAccount должен быть  
-- описан в структуре учета  
A = TradeAccount ( GetAccountByName("60") );
```

Описание

ТекущийЖурнал :Строка;
CurrentJournal :String;

Назначение

Функция возвращает строковое имя журнала, в рамках которого связывается типовая операция.

Функция допустима только на сервере расчетов

Пример

```
var ТекЖурнал :String;  
  
ТекЖурнал = CurrentJournal;
```

Класс **Объект** является родительским классом для всех остальных классов объектной модели ТБ.Скрипт, за исключением рассмотренных в других разделах классов [Система](#), [Математика](#), [Бухгалтерия](#) и [Консоль](#). Большинство классов, производных от класса **Объект**, наследуют его свойства, носящие фундаментальный характер, которые перечислены в теме ["Свойства базового класса"](#). Наследуемые свойства производных классов отображаются в окне [Иерархия классов](#) серым цветом, в то время как собственные или переопределенные свойства - черным.

Предупреждение. Класс **Объект** является абстрактным, то есть создавать объекты этого класса нельзя.

Объекты производных классов могут создаваться в программе с помощью соответствующих методов (как правило, **Create**) или неявным образом внутри системы (например, объект **Окно**, являющийся свойством **Формы**). В последнем случае программист на ТБ.Скрипт получает доступ к уже готовым объектам.

Каждый объект существует до тех пор, пока ссылка на него хранится в программе. Например, если указатель был присвоен локальной переменной, объект будет уничтожен при выходе из процедуры. Однако если объект был создан внутри функции и передан в качестве ее результата в вызывающую процедуру (функцию), где присвоен некоторой переменной, то уничтожение объекта откладывается. В этом случае время существования объекта определяется многими дополнительными факторами: например, если указатель был присвоен переменной, являющейся полем класса, то логика его работы зависит от того, была ли принимающая переменная описана в разделе [InClass](#) (свойство класса) или [InObject](#) (свойство объекта), а кроме того - собственно от особенностей класса. В частности, свойства класса **Запись** сохраняются навечно (в базе данных), а свойства класса **Бланк** - только до конца текущей сессии.

Как следует из вышесказанного, часть свойств классов являются фактически свойствами объектов, то есть описывают конкретные экземпляры объектов и воздействуют на них (когда речь идет о методах). Такие свойства допустимо использовать только в привязке к объектам. Остальные свойства относятся непосредственно к классам, и обращаться к ним можно не только через объект, но и посредством указания имени класса (даже если ни одного объекта класса еще нет). Такие свойства существуют в единственном экземпляре для всех объектов данного класса.









Все классы, производные от родительского класса **Объект**, в Справочной системе разделены по своему назначению на несколько групп:

- [Информационные классы](#)
- [Обслуживающие \(сервисные\) классы](#)
- [Классы для работы с сервером данных](#)
- [Классы для работы с шаблонами](#)
- [Классы для работы с картотеками](#)
- [Классы для работы с сервером расчетов](#)
- [Классы для работы с отчетами](#)
- [Пользовательские классы](#)






Базовый класс *Объект/Object* является родительским по отношению ко всем классам типа Объект. Наследуемые свойства производных классов отображаются в окне [Иерархия классов](#) серым цветом, в то время как собственные или переопределенные свойства черным.

Предупреждение. Класс *Объект* является абстрактным, то есть создавать объекты этого класса нельзя.

Свойства класса

-  [Поле ИмяКласса / ClassName](#)
-  [Поле ПроектКласса / ClassProject](#)
-  [Поле РодительскийКласс / ParentClass](#)
-  [Поле ИнформацияОКлассе / ClassInfo](#)
-  [Функция УнаследованОт / InheritsFrom](#)
-  [Функция ПорожденныеКлассы / ChildClasses](#)
-  [Функция ВзятьПолеКласса / GetClassField](#)
-  [Процедура УстановитьПолеКласса / SetClass](#)

Свойства объекта

-  [Поле ТипКласса / ClassType](#)
-  [Процедура Присвоить / Assign](#)
-  [Функция Вычислить / Evaluate](#)
-  [Функция ВзятьПоле / GetField](#)
-  [Процедура УстановитьПоле / SetField](#)

Описание

ИмяКласса: Строка;
ClassName: String;

Назначение

Возвращает название класса. Поле доступно только на чтение.

Пример

```
proc P1(O : Object);  
  Message("Это объект класса:" + O.ClassName);  
end;
```

Описание

ИнформацияОКлассе: [ИнфКласса](#);
ClassInfo: [ClassInfo](#);

Назначение

Возвращает указатель на объект RTTI-класса [ИнфКласса](#), содержащий описание текущего класса.

Пример

```
-- Класс1.cod
...
-- выводит количество членов (свойств и методов) в классе Класс1
trace(ClassInfo.MembersCount);
```


Описание

```
ПроектКласса :Строка;  
ClassProject :String;
```

Назначение

Возвращает имя проекта, которому принадлежит класс. Поле доступно только на чтение.

Пример

```
proc Инициализация(X :Класс Объект);  
  if X.ПроектКласса = "База" then  
    АктивироватьСхемуДоступа(X);  
  end;  
end;
```

Описание

РодительскийКласс: Класс Объект;
ParentClass: Class Object;

Назначение

Возвращает указатель на родительский класс. Если родительского класса не существует, возвращает nil. Поле доступно только на чтение.

Пример

```
proc Pl(O : Object);  
  if O.ParentClass = nil then  
    Message("У класса:" + O.ClassName + " нет родительского");  
  fi;  
end;
```

Описание

ТипКласса: Класс Объект;
ClassType: Class Object;

Назначение

Возвращает указатель на класс (именно на класс, а не на объект класса). Указатель позволяет создавать объекты на базе существующих (клонировать), проверять, принадлежат ли объекты одному классу или разным, а также определять наследственные связи в иерархии классов. Поле доступно только на чтение.

Пример

```
func F1(Init:Object; Base:Object):Logical;  
    var L : Logical;  
    if Base.ClassType = Init.ClassType then  
        Message(Init.ClassName+" = "+Base.ClassName);  
        L = TRUE;  
    else  
        Message(Init.ClassName+" <> "+Base.ClassName);  
        L = FALSE;  
    fi;  
    return L;  
end;
```

Описание

```
Присвоить(Об :Объект);  
Assign(Ob :Object);
```

Аргументы

Об - объект, выступающий в качестве эталона, из которого будет взята информация для заполнения свойств текущего объекта.

Назначение

Переносит данные из указанного объекта в текущий объект, если это возможно (то есть, если объекты принадлежат к одному классу).

Пример

```
func КлонироватьЗапись (Зап :Запись) :Запись;  
var КлассЗаписи :Класс Запись;  
    КлассЗаписи = Зап.ТипКласса;  
    Результат = КлассЗаписи.Создать;  
    Результат.Присвоить(Зап);  
end;
```

Описание

УстановитьПоле(ИмяПоля :Строка; Значение :Вариант);
SetField (FieldName :String; Value :Variant);

Аргументы

ИмяПоля - имя поля объекта, которое доступно на запись;

Значение - значение соответствующего типа, которое требуется записать в указанное поле. Заданное значение поля объекта должно совпадать с типом поля.

Назначение

Записывает требуемое значение в указанное по имени поле. Если такого поля нет или тип передаваемого значения во втором параметре не соответствует типу поля, генерируется исключительная ситуация. С помощью этого метода можно установить значение любого поля объекта по его имени.

Пример

```
proc P1(Sender :Button);
  var ob : BlankForm;
  ob = Бланк1.Create;
  -- устанавливаем значение поля "Продукты"
  ob.SetField("Продукты", "Колбаса");
  -- выводим значение поля "Продукты"
  Message("Значение поля= "+ ob.GetField("Продукты"));
end;
```

Описание

```
УстановитьПолеКласса(ИмяПоля :Строка; Значение :Вариант);  
SetClassField (FieldName :String; Value :Variant);
```

Аргументы

ИмяПоля - имя поля класса, которое доступно на запись;

Значение - значение соответствующего типа, которое требуется записать в указанное поле. Заданное значение поля класса должно совпадать с типом поля.

Назначение

Записывает требуемое значение в указанное по имени поле. Если такого поля нет или тип передаваемого значения во втором параметре не соответствует типу поля, генерируется исключительная ситуация. С помощью этого метода можно установить значение любого поля класса по его имени.

Пример

```
...  
-- фрагмент кода бланка "Бланк1",  
-- в котором объявлена переменная Склад  
inclass Public  
var Склад : String;  
...  
proc P1(Sender :Button); -- в текущем бланке  
...  
-- из текущего бланка изменяем значение переменной  
-- Склад другого бланка "Бланк1"  
    Бланк1.SetClassField("Склад", "МойСклад");  
    Message("Наименование склада = "+Бланк1.Склад);  
...  
end;
```

Описание

```
ВзятьПоле (ИмяПоля :Строка) :Вариант;  
GetField (FieldName :String) :Variant;
```

Аргументы

ИмяПоля - имя поля объекта.

Назначение

Функция возвращает значение поля по его имени. Если такого поля нет, генерируется исключительная ситуация. С помощью этого метода можно получить значение любого поля объекта по его имени.

Пример

```
proc P1(Sender :Button);  
  var ob : BlankForm;  
  ob=Бланк1.Create;  
  Message("Значение поля= "+ ob.GetField("Товар"));  
end;
```

Описание

```
ВзятьПолеКласса (ИмяПоля :Строка) :Вариант;  
GetClassField (FieldName :String) :Variant;
```

Аргументы

ИмяПоля - имя поля класса.

Назначение

Функция возвращает значение поля класса по его имени. Если такого поля нет, генерируется исключительная ситуация. С помощью этого метода можно получить значение любого поля класса по его имени.

Пример

```
proc P1(Sender :Button);  
  var ob : Record;  
  ob=Абонент.Create;  
  Message(Стр(ob.GetClassField("UserConstraintsCount")));  
  If Record.GetClassField("Abstract"):  
    Message(".....");  
  else  
    Message(".....");  
  end;  
end;
```


Описание

Вычислить(Выражение:Строка) :Вариант;
Evaluate(Expression:String) :Variant;

Аргументы

Выражение - текстовая строка, содержащая числа, названия переменных и операторы.

Назначение

Вычисляет указанное выражение, в которое могут входить названия полей класса (переменных-членов). Если в строке встречается неизвестное имя переменной, генерируется исключительная ситуация.

Пример

```
var O : Пример.Расчет;  
-- объект, производный от Объект  
O.Сумма = 120;  
O.СтавкаНДС = 0.20;  
O.Итого = O.Вычислить( "Сумма + Сумма*СтавкаНДС" );
```

Описание

```
ПорожденныеКлассы({Рекурсивно :Логический}) :Класс[];  
ChildClasses({Recursive :Logical}) :Class[];
```

Аргументы

Рекурсивно - определяет рекурсивность возвращаемых ссылок на классы. Другими словами, если от наследников данного класса были порождены другие классы, то при установленном параметре в Истину, будут возвращены также ссылки на классы, которые были порождены от наследников. Если параметр установлен в Ложь, то возвращаются ссылки на классы, порожденные только от данного класса. Параметр является не обязательным, по умолчанию он установлен в Истину.

Назначение

Функция возвращает массив ссылок на классы, объявленные наследниками данного класса. Элементы массива индексируются, начиная с 1.

Пример

```
proc TempClasses;  
  var Cls :Class[];  
  -- получаем массив классов объектов шаблона  
  Cls = TemplateObject.ChildClasses;  
  -- выводим размер массива  
  trace("Количество производных классов="+Str(LengthOfArray(Cls)));  
end;
```

Описание

```
УнаследованОт(Кл: Класс): Логическое;  
InheritsFrom(Cl: Class): Logical;
```

Аргументы

Кл - указатель на класс.

Назначение

Возвращает значение TRUE, если переданный через единственный аргумент класс является предком (непосредственным или более отдаленным) для того класса, в применении к которому вызван данный метод. Функция также возвращает TRUE, если переданный класс - тот же самый что и класс, использованный для вызова метода. Во всех остальных случаях возвращается FALSE.

Пример

```
proc Процедура (Зап :Запись);  
    if Зап.УнаследованОт(Документы.Накладная) then  
        -- это либо Накладная,  
        -- либо производный от нее класс документов  
        ...  
    else  
        ...  
    end;  
end;
```

Информационные классы языка ТБ.Скрипт позволяют программным путем получать сведения о текущей сессии, проекте, информационной базе, а также информацию [о типах во время исполнения](#) (run-time type information, RTTI) прикладных программ, написанных на языке ТБ.Скрипт.

В иерархии классов перечислены следующие информационные классы, являющиеся наследниками [класса Объект](#):

- [ИнфЧлена/MemberInfo](#) (родитель [RTTI-классов](#))
 - [ИнфТипа / TypeInfo](#)
 - [ИнфКласса / ClassInfo](#)
 - [ИнфПеречисления / EnumTypeInfo](#)
 - [ИнфПрограммногоТипа / UserTypeInfo](#)
 - [ИнфМетода / MethodInfo](#)
 - [ИнфСобытия / EventMethodInfo](#)
 - [ИнфПоля / FieldInfo](#)
 - [ИнфПоляПроводки / TransFieldInfo](#)
 - [ИнфПоляПризнака / SignFieldInfo](#)
 - [ИнфДинамическогоМетода / DynamicMethodInfo](#)
- [ИнфСессии](#)
- [ИнфБазы/BaseInfo](#)
- [ИнфПроекта](#)

Класс *ИнфБазы / BaseInfo*, производный от класса [Объект \(Object\)](#), предназначен для получения информации об информационной базе и входит в группу, так называемых [информационных классов](#).

Внимание! Ссылка на объект текущего класса хранится в поле [ИнфБазы / BaseInfo](#).

Непосредственно в классе *ИнфБазы / BaseInfo* вводятся следующие свойства:

- [Поле Имя / Name](#)
- [Поле КоличествоПроектов / ProjectCount](#)
- [Поле Проект / Project](#)
- [Поле ПроектПоИмени / ProjectByName](#)

Поле Имя / Name

Описание

Имя :Строка;
Name :String;

Назначение

Возвращает имя информационной базы.

Пример

```
proc P1(Sender :Button);  
...  
  Message("Имя инф.базы= " + BaseInfo.Name);  
...  
end;
```

Описание

КоличествоПроектов :Целое;
ProjectCount :Integer;

Назначение

Возвращает количество проектов.

Пример

```
proc P1(Sender :Button);  
  var Ob :ProjectInfo;  
  var j :Integer;  
  for j = 1 .. BaseInfo.ProjectCount do  
    Ob = BaseInfo.Project[j];  
    Message("Имя проекта= " + Ob.Name);  
  end;  
end;
```

Описание

Проект [Индекс :Целое] :ПроектКласса;
Project [Index :Integer]:ProjectInfo;

Аргументы

Индекс - индекс проекта.

Назначение

Возвращает ссылку на проект из класса **ПроектКласса** по его индексу или на текущий проект, если входной параметр отсутствует.

Пример

```
proc P1(Sender :Button);  
  var Ob :ProjectInfo;  
  var j  :Integer;  
  for j = 1 .. BaseInfo.ProjectCount do  
    Ob = BaseInfo.Project[j];  
    Message("Имя проекта = " + Ob.Name);  
  end;  
end;
```


Описание

ПроектПоИмени [Имя :Строка]:ПроектКласса;
ProjectByName [Name :String] :ProjectInfo:

Аргументы

Имя - имя проекта.

Назначение




Возвращает ссылку на проект из класса **ПроектКласса** по его имени.

Пример

```
proc P1(Sender :Button);
  var Ob :ProjectInfo;
  Ob = BaseInfo.ProjectByName[ "МойНовыйПроект" ];
  if Ob = nil:
    Message("Проект не открыт в текущей сессии");
  else
    Message("Имя проекта= " + Ob.Name);
  fi;
end;
```

Класс *ИнфДинамическогоКласса* / *DynamicClassInfo* предназначен для получения [RTTI-информации](#) о классах динамических методов и является наследником класса [ИнфКласса|ClassInfo](#), который, в свою очередь, является производным от родительских классов [Объект](#), [ИнфЧлена](#) и [ИнфТипа](#) и наследует от них все свойства и методы.

Непосредственно в классе *ИнфДинамическогоКласса* определены следующие свойства:

-  [Поле КоличествоДинамическихЧленов / DynamicMemberCount](#)
-  [Поле ДинамическийЧлен / DynamicMember](#)
-  [Функция ДинамическийЧленПоИмени / DynamicMemberByName](#)

Описание

`ДинамическийЧлен`[Индекс :Целое] : [ИнфДинамическогоМетода](#);
`DynamicMember`[Index :Integer] : [DynamicMethodInfo](#);

Аргументы

Индекс - номер динамического метода.

Назначение

Возвращает RTTI-информацию для динамического метода по его номеру, заданному в первом параметре свойства. Общее количество динамических методов, имеющихсся во время исполнения проекта, определяется в свойстве [DynamicMemberCount](#).

Описание

ДинамическихЧленов : Целое;
DynamicMemberCount : Integer;

Назначение

Возвращает общее количество динамических методов, использующихся во время исполнения проекта.

Описание

`ДинамическийЧленПоИмени(Имя :Строка) : ИнфДинамическогоМетода ;`
`DynamicMemberByName(Name :Строка) : DynamicMethodInfo ;`

Аргументы

Имя - имя динамического метода.

Назначение

Возвращает RTTI-информацию для динамического метода имя, которого задано в первом параметре функции.

Класс *ИнфДинамическогоМетода* / *DynamicMethodInfo* предназначен для получения [RTTI-информации](#) о динамических методах и является наследником класса [ИнфМетода](#), который, в свою очередь, является производным от родительских классов [Объект](#) и [ИнфЧлена](#).

Класс *ИнфДинамическогоМетода* наследует все свойства и методы от родительских классов и содержит два дополнительных поля:

- [Поле ИсходныйКод / SourceCode](#)
- [Поле Удален / IsDeleted](#)

Описание

```
ИсходныйКод :Строка[];  
SourceCode :String[];
```

Назначение

Возвращает массив строк исходного кода метода.

Поле доступно только на чтение.

Описание

Удален :Логическое;
IsDeleted :Logical;

Назначение






Свойство возвращает значение "Истина", если метод был удалён (см. [DynamicObject.DeleteMethod](#)).

Такая ситуация возможна, если объект с RTTI-информацией о методе ещё хранится, в то время как сам метод уже удалили.

ИнфКласса / ClassInfo

Класс *ИнфКласса* / *ClassInfo* предназначен для получения информации о классах, используемых в языке ТБ.Скрипт, и входит в группу, так называемых [информационных классов](#).

Непосредственно в классе *ИнфКласса* определены следующие свойства и методы:

-  [Поле НаКласс / ClassRef](#)
-  [Поле Описание / Description](#)
-  [Поле КоличествоЧленов / MembersCount](#)
-  [Поле Член / Member](#)
-  [Функция ЧленПоИмени / MemberByName](#)

Класс *ИнфКласса* является производным от родительских классов [Объект / Object](#), [ИнфЧлена / MemberInfo](#) и [ИнфТипа / TypeInfo](#) и наследует от них все свойства и методы.

Описание

КоличествоЧленов :Целое;
MembersCount :Integer;

Назначение

Возвращает количество членов в классе, информацию о котором содержит данный объект **ИнфКласса**.

Пример

```
proc P1(Sender :Button);  
  var N :Integer;  
  var j :Integer;  
  N=ClassInfo.MembersCount;  
  Message("Количество членов= " + Стр(N));  
  for j = 1 .. N do  
    Message(" Имя свойства члена " + ClassInfo.Member[j].Name);  
  end;  
end;
```

Описание

НаКласс :Класс;
ClassRef :Class;

Назначение

Возвращает ссылку на класс, информацию о котором содержит данный объект **ИнфКласса**.

Пример

```
proc P1(Sender :Button);  
...  
  Message(Ctp(ClassInfo.ClassRef));  
...  
end;
```

Описание

Описание :Строка;
Description :String;

Назначение

Возвращает описание класса, с которым связан текущий объект.

Поле доступно только на чтение.

Пример

```
proc P1(C: Class UserObject);  
  ...  
  Message(C.ClassInfo.Description);  
  ...  
end;
```

Описание

```
Член[Номер :Целое] :ИнфЧлена;  
Member[Number :Integer] :MemberInfo;
```

Назначение

По номеру члена возвращает другой объект с RTTI-информацией о члене класса, информация о котором содержится в данном объекте **ИнфКласса**. Номер должен лежать от 1 до общего числа членов класса, описываемых данным объектом **ИнфКласса** (см. поле [КоличествоЧленов](#)).

Хотя поле имеет тип **ИнфЧлена**, в нем содержится объект одного из производных конкретных классов: **ИнфКласса**, **ИнфПеречисления**, **ИнфПрограммногоТип**, **ИнфМетода**, **ИнфСобытия**.

Пример

```
proc P1(Sender :Button);  
  var N :Integer;  
  var j :Integer;  
  N=ClassInfo.MembersCount;  
  Message("Количество членов= " + Стр(N));  
  for j = 1 .. N do  
    Message(" Имя свойства члена " + ClassInfo.Member[j].Name);  
  end;  
end;
```

Описание

```
ЧленПоИмени (Имя :Строка):ИнфЧлена;  
MemberByName (Name :String):MemberInfo;
```

Аргументы

Имя - идентификатор члена класса.

Назначение

Функция ищет свойство или метод с указанным именем среди свойств и методов класса, информацию о котором содержит данный объект **ИнфКласса**. Если искомый идентификатор действительно определен в классе, функция возвращает информацию о нем в виде другого объекта RTTI-класса.

Хотя возвращаемое значение имеет тип **ИнфЧлена**, в нем содержится объект одного из производных конкретных классов: **ИнфКласса**, **ИнфПеречисления**, **ИнфПрограммногоТип**, **ИнфМетода**, **ИнфСобытия**.

Пример

```
proc P1(Sender :Button);  
...  
  Message(Стр(ClassInfo.MemberByName("ClassProject")));  
...  
end;
```

Класс *ИнфКлассаЗаписи*|*RecordClassInfo* предназначен для получения [RTTI-информации](#) о классе записи. Он является наследником класса [ИнфКласса](#)|*ClassInfo*, который, в свою очередь, является производным от родительских классов [Объект](#), [ИнфЧлена](#) и [ИнфТипа](#) и наследует от них все свойства и методы.

Для получения RTTI-информации о классе записи в классе *ИнфКлассаЗаписи* определены следующие свойства:

- [Поле Иерархическая / IsHierarchical](#)
- [Поле Абстрактная / IsAbstract](#)
- [Поле Привязано / IsMapped](#)
- [Поле Внешняя / External](#)
- [Поле ВнешнийКлюч / ForeignKeyField](#)
- [Поле ИменаОписательныхПолей / DescribeFieldNames](#)
- [Поле ИмяБазыДанных / DataBaseName](#)
- [Поле ИмяБДТаблицы / DBTableName](#)

Поле Абстрактная / Abstract

Описание

Абстрактная :Логическое;

Abstract :Logical;

Назначение

Свойство возвращает значение "Истина", если запись абстрактная, т.е. описана в MTL-файле с модификатором [Abstract](#). Модификатор **Abstract** позволяет определять так называемые абстрактные классы записей, которые не могут иметь экземпляров записей (документов), они лишь декларируют свойства, необходимые для классов-наследников.

Поле доступно только на чтение.

Описание

ВнешнийКлюч : [ИнфЧлена](#);
ForeignKeyField : [MemberInfo](#);

Назначение

Возвращает объект класса [MemberInfo](#) для поля, заданного как внешний ключ.

По умолчанию внешним ключом является поле ExtID записи, но это MTL-поле в настоящее время не имеет представителя в классе **Запись** и наследниках, т.к. в классе **Запись|Record** поле ExtID заведено на уровне ТБ.Скрипта. Таким образом, если внешний ключ записи задан по умолчанию, свойство ForeignKeyField вернёт информацию о поле [ВнешнийКлюч|ExtID](#) класса **Запись**.

Внимание. Если же внешний ключ задан, и он отличен от ExtID, то свойство ForeignKeyField вернёт информацию об этом поле, которое будет присутствовать в соответствующем наследнике класса **Запись**. В этом случае можно получить расширенную информацию о поле, т.к. фактически ForeignKeyField вернёт объект класса [ИнфПоля|FieldInfo](#) (т.е. ForeignKeyField is FieldInfo = True).

Поле Внешняя / External

Описание

Внешняя :Логическое;

External :Logical;

Назначение

Свойство возвращает значение "Истина", если запись является внешней записью, иначе - "Ложь".

Поле Иерархическая / Hierarchical

Описание

Иерархическая :Логическое;
Hierarchical :Logical;

Назначение

Свойство возвращает значение "Истина", если запись иерархическая, т.е. описана в [MTL-файле](#) с модификатором **Hierarchical**.

Поле доступно только на чтение.

Описание

ИменаОписательныхПолей :Строка;
DescribeFieldNames :String;

Назначение

Возвращает имена [описательных полей](#), перечисленных через точку с запятой ";", или пустую строку, если в MTL соответствующей записи описательные поля не заданы.

Описание

ИмяБазыДанных :Строка;
DataBaseName :String;

Назначение

Возвращает имя физической базы СУБД, в которой размещен соответствующий класс документа.

Пример

```
proc ButtonOnClick(Sender :Button);
var vSQL :String;
var vConnection, vRecordset :AutoObject;
vSQL = "SELECT * FROM [" + Справочники.Товар.ClassInfo.DBTableName +
"] WITH (NOLOCK)";
vConnection = AutoObject.Create("ADODB.Connection");
vConnection.ConnectionString =
"Provider=SQLOLEDB;Integrated Security=SSPI;Persist Security Info=False;" +
"Data Source=.;Initial Catalog=" + Справочники.Товар.ClassInfo.DataBaseName + ";" +
-- Получаем имя физической базы, в которой расположен документ Справочники.Товар
"Use Procedure for Prepare=1;" +
"Auto Translate=True;Packet Size=16384;Application Name=ИмяПрограммы;" +
"Use Encryption for Data=False;Tag with column collation when possible=False";
try
vConnection.Invoke("Open", vConnection.ConnectionString);
try
vRecordset = AutoObject.Create("ADODB.Recordset");
vRecordset.Invoke("Open", vSQL, vConnection);
try
...
finally
vRecordset.Invoke("Close");
end;
finally
vConnection.Invoke("Close");
end;
finally
vConnection = nil;
end;
end;
```

Описание

ИмяБДТаблицы :Строка;
DBTableName :String;

Назначение

Возвращает имя таблицы, под которой размещен соответствующий класс документа.

Пример

```
proc ButtonOnClick(Sender :Button);
var vSQL :String;
var vConnection, vRecordset :AutoObject;
vSQL = "SELECT * FROM [" + Справочники.Товар.ClassInfo.DBTableName +
"] WITH (NOLOCK)";
vConnection = AutoObject.Create("ADODB.Connection");
vConnection.ConnectionString =
"Provider=SQLOLEDB;Integrated Security=SSPI;Persist Security Info=False;" +
"Data Source=.;Initial Catalog=" + Справочники.Товар.ClassInfo.DataBaseName + ";" +
-- Получаем имя физической базы, в которой расположен документ Справочники.Товар
"Use Procedure for Prepare=1;" +
"Auto Translate=True;Packet Size=16384;Application Name=ИмяПрограммы;" +
"Use Encryption for Data=False;Tag with column collation when possible=False";
try
vConnection.Invoke("Open", vConnection.ConnectionString);
try
vRecordset = AutoObject.Create("ADODB.Recordset");
vRecordset.Invoke("Open", vSQL, vConnection);
try
...
finally
vRecordset.Invoke("Close");
end;
finally
vConnection.Invoke("Close");
end;
finally
vConnection = nil;
end;
end;
```

Поле Привязано / IsMapped

Описание

Привязано :Логическое;

Mapped :Logical;

Назначение













Свойство возвращает значение Истина, если записи сопоставлена таблица в базе данных, т.е. запись привязана, или "замэплена".

Поле доступно только на чтение.

Класс *ИнфМетода / MethodInfo* относится к [группе классов](#), предназначенных для получения информации [о типах во время исполнения](#) (run-time type information, RTTI) программы на ТБ.Скрипт.

Класс *ИнфМетода* является производным от классов [Объект / Object](#) и [ИнфЧлена / MemberInfo](#) и наследует все свойства и методы родительских классов.

Непосредственно в этом классе определены следующие свойства, методы и перечислимые типы:

-  [Поле СвойствоКласса / InClassProperty](#)
-  [Поле Вид / Kind](#)
-  [Поле ТолькоНаЧтение / ReadOnly](#)
-  [Поле ЧислоПараметров / ParamsCount](#)
-  [Процедура ИнфПараметра / ParamInfo](#)
-  [Поле ТипРезультата / ResultType](#)
-  [Функция ВызовФункции / CallFunc](#)
-  [Функция ВызовФункцииДоп / CallFuncEx](#)
-  [Процедура ВызовПроцедуры / CallProc](#)
-  [Процедура ВызовПроцедурыДоп / CallProcEx](#)
-  [Тип ВидМетода / MethodKind](#)
-  [Тип ВидПараметра / ParamKind](#)

Константы видов методов (**ВидМетода** / **MethodKind**)

Константы видов методов используются для задания значений свойства [Вид/Kind](#). Возможные значения определены с помощью перечисления **ВидМетода** / **MethodKind**:

- **вмСвойство** / **mkProp** - свойство (поле);
- **вмПроц** / **mkProc** - процедура;
- **вмФунк** / **mkFunc** - функция.

Константы видов параметров (ВидПараметра / ParamKind)

Константы видов параметров используются процедурой **ИнфПараметра**. Возможные значения определены с помощью перечисления **ВидПараметра / ParamKind**:

- **vpПростой / pkSimple** - обычный параметр для передачи значения в метод;
- **vpВозврат / pkVar** - параметр может использоваться не только для передачи значения в метод, но и для возврата значения из метода;
- **vpКонст / pkConst** - неизменяемый параметр для передачи в метод.

Описание

Вид :ИнфМетода.ВидМетода;
Kind :MethodInfo.MethodKind;

Назначение

Значение поля равно одному из predetermined значений, заданных перечислением [ИнфМетода.ВидМетода](#) и описывающих вид либо как свойство/поле, либо как процедуру, либо как функцию.

Пример

```
proc P1(Sender :Button);  
  ...  
  
  Message(Стп(MethodInfo.MethodKind.Kind));  
  ...  
end;
```

Описание

`СвойствоКласса` :Логическое;
`InClassProperty` :Logical;

Назначение

Значение поля равно TRUE, если свойство, описываемое данным объектом **ИнфМетода**, является свойством класса (определено в секции **InClass** cod-модуля). Для свойств, определенных в секции **InObject** и потому являющихся свойствами экземпляров класса (объектов), значение поля равно FALSE.

Пример

Описание

```
ТипРезультата :ИнфТипа;  
ResultType :TypeInfo;
```

Назначение

Поле позволяет узнать типа возвращаемого методом значения.

Фактически поле содержит ссылку на объект класса, производного от **ИнфТипа (ИнфКласса, ИнфПеречисление, ИнфПрограммногоТипа)**.

Значение поля определено только для свойств и функций; в случае процедуры поле равно nil.

Пример

Поле ТолькоНаЧтение / ReadOnly

Описание

ТолькоНаЧтение :Логическое;
ReadOnly :Logical;

Назначение

Значение поля равно TRUE, если свойство, описываемое данным объектом **ИнфМетода**, доступно только на чтение, и равно FALSE в противном случае, то есть если поле доступно также и на запись.

Пример

Поле ЧислоПараметров / ParamsCount

Описание

```
ЧислоПараметров :Целое;  
ParamsCount :Integer;
```

Назначение

Поле позволяет узнать количество параметров у метода, описываемого данным объектом класса **ИнфМетода**.

Примечание

К свойству **ЧислоПараметров / ParamsCount** класса "ИнфМетода" можно обращаться только в объектах класса, а не в самом классе.

Описание

```
ВызовПроцедуры (Экземпляр :Вариант {; <параметры>});  
CallProc (Self :Variant {; });
```

Аргументы

Экземпляр - ссылка на объект, в контексте которого будет производиться вызов метода.

Внимание! Процедура имеет переменное число аргументов.

Состав списка остальных параметров варьируется в зависимости от числа и типа параметров метода. Этих параметров может и не быть, если процедура текущего объекта **ИнфМетода** не имеет аргументов.

Назначение

Процедура осуществляет вызов данного метода, описываемого объектом **ИнфМетода**. Если данный метод является методом экземпляра класса (описан в разделе **InObject**), то в качестве **Экземпляра** должен передаваться объект, иначе (описан в разделе **InClass**) - класс. Если в качестве **Экземпляра** передать **nil**, то при вызове метода экземпляра класса произойдет ошибка "Вызов свойства или метода неинициализированного объекта", для метода класса в этом случае в качестве **Экземпляра** будет использоваться класс, где описан данный метод.

При вызове процедур с необязательными параметрами все параметры (включая и необязательные) должны быть переданы в **ВызовПроцедуры**.

Для случаев, когда заранее не известно количество параметров, удобнее пользоваться процедурой [ВызовПроцедурыДоп](#), принимающей вместо списка параметров один параметр-массив со всеми параметрами вызываемой процедуры.

Для получения значений функций или полей следует использовать функции [ВызовФункции](#) или [ВызовФункцииДоп](#).

Пример

```
-- рекурсивный вызов процедуры самой себя  
-- во время исполнения произойдет заикливание!  
proc P1;  
var mi :MethodInfo;  
  -- получаем объект-описание процедуры P1  
  mi = ClassInfo.MemberByName('P1');  
  -- вызываем опосредованно эту процедуру  
  mi. CallProc(mi);  
end;
```


Описание

```
ВызовПроцедурыДоп (Экземпляр :Вариант; Параметры :Вариант[]);  
CallProcEx (Self :Variant; Parameters :Variant[]);
```

Аргументы

Экземпляр - ссылка на объект, в контексте которого будет производиться вызов метода.

Параметры - массив вариантов со значениями всех параметров метода. Если метод не имеет параметров, здесь передается nil.

Назначение

Процедура осуществляет вызов данного метода, описываемого объектом **ИнфМетода**. Если данный метод является методом экземпляра класса (описан в разделе **InObject**), то в качестве **Экземпляра** должен передаваться объект, иначе (описан в разделе **InClass**) - класс. Если в качестве **Экземпляра** передать **nil**, то при вызове метода экземпляра класса произойдет ошибка "Вызов свойства или метода неинициализированного объекта", для метода класса в этом случае в качестве **Экземпляра** будет использоваться класс, где описан данный метод.

При вызове процедур с необязательными параметрами все параметры (включая и необязательные) должны быть переданы в **ВызовПроцедурыДоп**.

Если количество параметров вызываемой процедуры известно заранее, то предпочтительнее использовать [ВызовПроцедуры](#), так как она выполняется быстрее чем **ВызовПроцедурыДоп**.

Для получения значений функций или полей следует использовать функции [ВызовФункции](#) или [ВызовФункцииДоп](#).

Пример

```
proc P1;  
var mi :MethodInfo;  
var n1, n2 :Integer;  
  -- получаем объект-описание процедуры P2  
  mi = ClassInfo.MemberByName('P2');  
  -- вызываем опосредованно эту процедуру  
  n1 = 2;  
  n2 = 3;  
  mi.CallProcEx(self,n1,n2);  
  -- выводим результат  
  message(Str(n1)+' '+Str(n2));  
end;  
  
-- возвращает сумму и произведение переданных чисел  
-- через те же два параметра  
proc P2(var Num1 :Integer; var Num2 :Integer);  
var Sum :Integer;  
var Mul :Integer;  
  Sum = Num1+Num2;  
  Mul = Num1*Num2;  
  Num1 = Sum;  
  Num2 = Mul;  
end;
```

Описание

```
ИнфПараметра (Индекс :Целое; var ПарамИмя :Строка; var ПарамТип :ИнфТипа; var  
ПарамВид :ИнфМетода.ВидПараметра);  
ParamInfo (Index :Integer; var ParName :String; var ParType :TypeInfo; var  
ParKind :MethodInfo.ParamKind);
```

Аргументы

Индекс - номер параметра метода;

ПарамИмя - имя параметра (заполняется самой процедурой);

ПарамТип - выходной параметр, принимающий значение одного из классов, производных от **ИнфТипа** с описанием типа параметра (заданного номером **Индекс**);

ПарамВид - выходной параметр, принимающий одно из predetermined значений перечисления [ИнфМетода.ВидПараметра](#); возможные значения описывают простые параметры, var-параметры (работающие не только на вход, но и на выход), const-параметры.

Назначение

По номеру параметра, передаваемому через **Индекс**, процедура возвращает его имя, тип и вид. Номер должен лежать в пределах от 1 до общего числа параметров у метода (свойство [ЧислоПараметров](#)).

В параметр **ПарамТип** процедура помещает ссылку на объект класса, производного от ИнфТипа (**ИнфКласса**, **ИнфПеречисление**, **ИнфПрограммногоТипа**).

Пример

Описание

```
ВызовФункции (Экземпляр :Вариант {; <параметры>}) :Variant;  
CallFunc (Self :Variant {; }) :Variant;
```

Аргументы

Экземпляр - ссылка на объект, в контексте которого будет производиться вызов функции или поля.

Внимание! Функция имеет переменное число аргументов.

Состав списка остальных параметров варьируется в зависимости от числа и типа параметров свойства. Этих параметров может и не быть, если функция текущего объекта **ИнфМетода** не имеет аргументов или происходит обращение к полю.

Назначение

Функция осуществляет вызов функции или поля, описываемых объектом **ИнфМетода**. Если данное свойство является свойством экземпляра класса (описан в разделе **InObject**), то в качестве **Экземпляра** должен передаваться объект, иначе (описан в разделе **InClass**) - класс. Если в качестве **Экземпляра** передать **nil**, то при обращении к свойству экземпляра класса произойдет ошибка "Вызов свойства или метода неинициализированного объекта", для свойства класса в этом случае в качестве **Экземпляра** будет использоваться класс, где описан данный метод.

Функция возвращает значение, возвращаемое вызванным свойством.

Для случаев, когда заранее не известно количество параметров, удобнее пользоваться функцией [ВызовФункцииДоп](#), принимающей вместо списка параметров один параметр-массив со всеми параметрами вызываемой функции (свойства).

При вызове функций с необязательными параметрами, все параметры (включая и необязательные) должны быть переданы в **ВызовФункции**.

Для вызова процедур необходимо использовать процедуры [ВызовПроцедуры](#) или [ВызовПроцедурыДоп](#).

Пример

```
proc P1;  
var mi :MemberInfo;  
var n :Integer;  
  -- получаем объект-описание функции F1  
  mi = ClassInfo.MemberByName('F1');  
  -- вызываем опосредованно эту функцию  
  n = SubF1(mi,2);  
  -- выводим результат  
  message(n);  
end;  
  
-- возвращает квадрат переданного числа  
func F1(Num1 :Integer): Integer;  
  return Num1*Num1;  
end;  
  
-- SubF1 вызывает функцию, описываемую параметром mi, через CallFunc  
func SubF1(mi :MethodInfo; Num :Variant): Variant;  
  Result = mi. CallFunc (self,Num);  
end;
```

Описание

ВызовФункцииДоп (Экземпляр :Вариант; Параметры :Вариант[]) :Вариант;
CallFuncEx (Self :Вариант; Parameters :Вариант[]) :Вариант;

Аргументы

Экземпляр - ссылка на объект, в контексте которого будет производиться вызов метода или свойства.
Параметры - массив вариантов со значениями всех параметров метода. Если метод не имеет параметров или осуществляется обращение к полю, здесь передается **nil**.

Назначение

Функция осуществляет вызов функции или поля, описываемых объектом **ИнфМетода**. Если данное свойство является свойством экземпляра класса (описан в разделе **InObject**), то в качестве **Экземпляра** должен передаваться объект, иначе (описан в разделе **InClass**) - класс. Если в качестве **Экземпляра** передать **nil**, то при обращении к свойству экземпляра класса произойдет ошибка "Вызов свойства или метода неинициализированного объекта", для свойства класса в этом случае в качестве **Экземпляра** будет использоваться класс, где описан данный метод.

Функция возвращает значение, возвращаемое вызванным свойством.

При вызове функций с необязательными параметрами все параметры (включая и необязательные) должны быть переданы в **ВызовФункцииДоп**.

Если количество параметров вызываемого свойства известно заранее, то предпочтительнее использовать [ВызовФункции](#), так как она выполняется быстрее чем **ВызовФункцииДоп**.

Для вызова процедур необходимо использовать процедуры [ВызовПроцедуры](#) или [ВызовПроцедурыДоп](#).

Пример

```
proc P1;
var mi :MemberInfo;
var n :Integer;
-- получаем объект-описание функции F1
mi = ClassInfo.MemberByName('F1');
-- вызываем опосредованно эту функцию
n = SubF1(mi,2,3);
-- выводим результат
message(n);
end;

-- возвращает произведение переданных чисел
func F1(Num1 :Integer; Num2 :Integer): Integer;
return Num1*Num2;
end;

-- SubF1 вызывает функцию, описываемую параметром mi, через CallFunc
func SubF1(mi :MethodInfo; Num1 :Variant; Num2 :Variant): Variant;
-- кроме self передаем в CallFuncEx массив из двух параметров
Result = mi. CallFuncEx.x(self,[Num1,Num2]);
end;
```

Класс *ИнфПеречисления / EnumTypeInfo* относится к [группе классов](#), предназначенных для получения информации [о типах во время исполнения](#) (run-time type information, RTTI) проекта. Этот класс включает в себя свойства и методы для получения информации о неobjектных типах-перечислениях. Другие типы описываются с помощью классов *ИнфКласса* и *ИнфПрограммногоТипа*. Все они являются прямыми наследниками класса [ИнфТипа](#).

Класс *ИнфПеречисления* является производным от классов [Объект](#), [ИнфЧлена](#) и [ИнфТипа](#) и наследует от них все свойства и методы.

Непосредственно в классе *ИнфПеречисления* определены следующие свойства и методы:

- [Количество / ItemsCount](#)
- [Пункты / Items](#)

Описание

Количество :Целое;
ItemsCount :Integer;

Назначение

Возвращает количество predetermined поименованных констант в перечислении, описываемом данным объектом **ИнфПеречисления**.

Пример

```
proc P1(Sender :Button);
  var j :Integer;
  var N :Integer;
  var vInfo :EnumTypeInfo;
  vInfo = Record.ClassInfo.MemberByName("RecordStates")
    as EnumTypeInfo;
  N = vInfo.ItemsCount;
  Message(Стр(N));
  for j = 1.. N do
    Message("Имя константы " + vInfo.Items[j].Name);
  end;
```

Описание

Пункты [Индекс :Целое] :ИнфМетода;
Items [Index :Integer] :MethodInfo;

Назначение

По индексу (номеру) возвращает объект класса **ИнфМетода** с RTTI-информацией о константе перечисления. Индекс должен лежать в пределах от 1 до общего числа констант в перечислении (свойство [Количество](#)).

Пример

```
proc P1(Sender :Button);
  var j :Integer;
  var N :Integer;
  var vInfo :EnumTypeInfo;
  vInfo =Record.ClassInfo.MemberByName("RecordStates") as EnumTypeInfo;
  N = vInfo.ItemsCount;
  Message(Стр(N));
  for j = 1.. N do
    Message("Имя константы " + vInfo.Items[j].Name);
  end;
```

Класс *ИнфПодтаблицы* / *SubtableClassInfo* предназначен для получения программным путем [RTTI-информации](#) по подтаблицам и является наследником класса [ИнфКласса|ClassInfo](#).

Непосредственно в классе *ИнфПодтаблицы* определены следующие свойства объектов:

- [Поле ИмяБДТаблицы / DBTableName](#)

Описание

ИмяБДТаблицы :Строка;
DBTableName :String;

Назначение

Возвращает имя таблицы под которой размещена соответствующая подтаблица документа.

Пример

```
-- Формирование прямого SQL-запроса  
-- по подтаблице "Товары" документа "Накладная":  
vSQL = "SELECT * FROM [ " +  
    Подтаблица_Документы.Накладная.Товары.ClassInfo.DBTableName + " ] WITH (NOLOCK)";
```

ИнфПоля / FieldInfo

Класс *ИнфПоля* / *FieldInfo* предназначен для получения [RTTI информации](#) по полям записи, являясь производным от классов [Объект](#), [ИнфЧлена](#) и [ИнфМетода](#), наследует от них все свойства и методы.

Кроме унаследованных полей, в классе *ИнфПоля* определены еще следующие свойства:

- [Заголовок / Title](#)
- [МаксимальныйРазмер / MaxSize](#)

Замечание. Для проверки того, что свойство является полем достаточно выполнить операцию IS: `Members[I] is FieldInfo`.

Поле Заголовок / Title

Описание

```
Заголовок :Строка;  
Title :String;
```

Назначение

Поле позволяет получить описание поля, данное ему в MTL-файле.

Пример

Описание

МаксимальныйРазмер :Целое;
MaxSize :Integer;

Назначение

Данное свойство позволяет узнать размер строкового поля, указанный в MTL-описании. Свойство имеет смысл только для полей строкового типа.

Пример

Класс *ИнфПоляПризнака* / *SignFieldInfo* предназначен для получения [RTTI информации](#) о типе поля признака, он является производным от классов [Объект](#), [ИнфЧлена](#) и [ИнфМетода](#), наследует от них все свойства и методы.

Класс *ИнфПоляПризнака* является непосредственным наследником класса *ИнфМетода* и дополнительно включает свойство:

● [Привязка / Bind](#).

Описание

Привязка :ИнфПоля;
Bind :FieldInfo;

Назначение

Возвращает [RTTI](#) поля документа, из которого берется значение поля признака, или nil, если привязка отсутствует.




Пример

```
var локЭтоСправочник :Logical;  
var локИнфМетода :MethodInfo;  
var локПривязка :FieldInfo;  
var локПолеПривязки :String;  
if локЭтоСправочник then  
  Assert(локИнфМетода is SignFieldInfo);  
  локПривязка = (локИнфМетода as SignFieldInfo).Bind;  
  if локПривязка <> nil then  
    локПолеПривязки = ExtractWord(локПривязка.Name, 1, '|');  
    .....  
  end;  
end;
```

Класс *ИнфПоляПроводки* / *TransFieldInfo* относится к [группе классов](#), предназначенных для получения [RTTI информации](#) и содержит RTTI информацию по полям (т.е. параметрам) полупроводок (HTrans).

Класс *ИнфПоляПроводки* является производным от классов [Объект](#), [ИнфЧлена](#) и [ИнфМетода](#) и наследует от них все свойства и методы.

Класс *ИнфПоляПроводки* является непосредственным наследником класса *ИнфМетода* и включает в себя следующие дополнительные поля, доступные только для чтения:

-  [Поле Заголовок / Title](#)
-  [Поле ПривязкаДеб / BindDeb](#)
-  [Поле ПривязкаКре / BindCre](#)
-  [Поле РавенПолю / EqualField](#)
-  [Поле Виден / Visible](#)

Поле Виден / Visible

Описание

Виден : Логическое;
Visible : Logical;

Назначение

Возвращает True, если соответствующее поле проводки видимо, и False в противном случае.

Поле Заголовок / Title

Описание

```
Заголовок :Строка;  
Title :String;
```

Назначение

Возвращает отображаемый заголовок параметра.

Описание

ПривязкаДеб : [ИнфПоля](#);
BindDeb : [FieldInfo](#);

Назначение

Возвращает [RTTI](#) поля документа, к которому параметр привязан по дебету или nil, если привязка отсутствует.

Внимание. На сервере расчётов поле **ПривязкаДеб** имеет тип String, а не FieldInfo!

Описание

```
ПривязкаКре : ИнфПоля;  
BindCre : FieldInfo;
```

Назначение

Возвращает [RTTI](#) поля документа, к которому параметр привязан по кредиту или nil, если привязка отсутствует.

Внимание. На сервере расчётов поле **ПривязкаКре** имеет тип String, а не FieldInfo!

Описание

РавенПолю : [ИнфПоляПроводки](#);
EqualField : [TransFieldInfo](#);

Назначение

Возвращает [RTTI](#) базового параметра только для [псевдонимов параметров типов счетов](#) или nil - для обычных параметров.

Пример

```
var locInfo :MemberInfo;  
-- ...  
locInfo = Полупроводка_СчТовары.ClassInfo.MemberByName("Товар");  
if locInfo is TransFieldInfo then  
  with TransFieldInfo(locInfo) do  
    Trace(Name + " - " + if (EqualField = nil,  
      "простой параметр", "псевдоним счета"));  
  end;  
end;
```

Класс *ИнфПрограммногоТипа* / *UserTypeInfo* относится к [группе классов](#), предназначенных для получения информации [о типах во время исполнения](#) (run-time type information, RTTI) программы на ТБ.Скрипт. Этот класс включает в себя свойства и методы для получения информации о пользовательских типах, описанных с помощью ключевого слова *type*. Другие типы описываются с помощью классов *ИнфКласса* и *ИнфПеречисления*. Все они являются прямыми наследниками класса [ИнфТипа](#).

Класс *ИнфПрограммногоТипа* является производным от классов [Объект / Object](#), [ИнфЧлена / MemberInfo](#) и [ИнфТипа](#) *ИнфТипа* / *TypeInfo* и наследует от них все свойства и методы.

Непосредственно в классе *ИнфПрограммногоТипа* определены следующие свойства и методы:

- [Поле Размерность / Dimension](#)
- [Поле БазовыйТип / BaseType](#)

Поле БазовыйТип / BaseType

Описание

```
БазовыйТип :ИнфТипа;  
BaseType :TypeInfo;
```

Назначение

Поле позволяет получить ссылку на объект, описывающий базовый (родительский) тип для того программного типа, который описывается данным объектом класса **ИнфПрограммногоТипа**.

Хотя возвращаемое значение имеет тип **ИнфТипа**, в нем содержится объект одного из производных конкретных классов: **ИнфКласса**, **ИнфПеречисления**, **ИнфПрограммногоТип**.

Пример

Поле **Размерность** / **Dimension**

Описание

Размерность :Целое;
Dimension :Integer;

Назначение

Поле содержит размерность программного (пользовательского) типа. Скалярный тип имеет размерность 0, вектор (одномерный массив) - 1, двумерный массив - 2, и т.д.

Пример

Класс *ИнфПроекта / ProjectInfo* входит в группу, так называемых [информационных классов](#), и предназначен для получения информации о проектах, которая также представлена в [окне "Администрирование"](#). Класс наследует все свойства и методы от родительского класса [Объект](#).

В классе *ИнфПроекта / ProjectInfo* вводятся следующие свойства:

- [Поле Имя / Name](#)
- [Поле Название / Title](#)
- [Поле Комментарий / Comment](#)
- [Поле Версия / Version](#)
- [Поле Автор / Author](#)
- [Поле Код / Code](#)
- [Поле Поле СубКод / SubCode](#)
- [Поле Лицевой / Frontal](#)
- [Поле Каталог / Folder](#)

Описание

Автор :Строка;
Author :String;

Назначение

Возвращает сведения об авторе проекта.

Пример

```
proc P1(Sender :Button);  
  var Ob :ProjectInfo;  
  Ob= BaseInfo.Project[2];  
  Message("Автор проекта= " + Ob.Author);  
end;
```

Описание

Версия :Строка;
Version :String;

Назначение

Возвращает версию проекта.

Пример

```
proc P1(Sender :Button);  
  var Ob :ProjectInfo;  
  Ob= BaseInfo.Project[2];  
  Message("Версия проекта= " + Ob.Version);  
end;
```

Описание

Имя :Строка;
Name :String;

Назначение

Возвращает имя проекта.

Пример

```
proc P1(Sender :Button);  
  var Ob :ProjectInfo;  
  Ob= BaseInfo.Project[2];  
  Message("Имя проекта= " + Ob.Name);  
end;
```

Описание

Каталог :Строка;
Folder :String;

Назначение

Возвращает строку, содержащую требуемый путь (включая завершающий обратный слэш) к папке, в которой размещается текущий проект.

Возвращаемая строка содержит макрос (строки подстановки) '%Projects%'. Пути вместе с таким макросом могут передаваться в функции работы с [папками и файлами](#) для определения свойств объектов файловой системы на сервере.

Пример

```
proc P1(Sender :Button);
  var Ob :ProjectInfo;
  Ob:= BaseInfo.Project[2];
  Message("Папка = " + Ob.Folder);
  -- Папка = %Projects%\ИмяПроекта\
  -- ИмяПроекта - название папки с данными текущего проекта
end;
```

Описание

Код :Целое;
Code :Integer;

Назначение

Возвращает код проекта.

Пример

```
proc P1(Sender :Button);  
  var Ob :ProjectInfo;  
  Ob= BaseInfo.Project[2];  
  Message("Код проекта= " + Ob.Code);  
end;
```

Описание

Комментарий :Строка;
Comment :String;

Назначение

Возвращает комментарий (произвольный текст, поясняющий назначение проекта).

Пример

```
proc P1(Sender :Button);  
  var Ob :ProjectInfo;  
  Ob= BaseInfo.Project[2];  
  Message("Комментарий к проекту= " + Ob.Comment);  
end;
```

Описание

Лицевой :Догическое;
Frontal :Logical;

Назначение

Возвращает значение Истина, если данный проект является лицевым, иначе - Ложь.

Пример

```
proc P1(Sender :Button);  
  var Ob :ProjectInfo;  
  Ob= BaseInfo.Project[2];  
  if Ob.Frontal:  
    message("Проект лицевой");  
  else  
    message("Проект не является лицевым");  
  fi;  
end;
```

Описание

Название :Строка;
Title :String;

Назначение

Возвращает название проекта (произвольное строковое выражение, характеризующее назначение проекта).

Пример

```
proc P1(Sender :Button);  
  var Ob :ProjectInfo;  
  Ob= BaseInfo.Project[2];  
  Message("Название проекта= " + Ob.Title);  
end;
```


Описание

СубКод :Целое;
SubCode :Integer;

Назначение

Свойство возвращает [номер выпуска](#) программного продукта.

Поле доступно только на чтение.

Пример

```
proc P1(Sender :Button);  
  var Ob :ProjectInfo;  
  Ob = BaseInfo.Project[1];  
  Message("Номер выпуска = " + Ob.SubCode);  
end;
```

Класс *ИнфСессии / SessionInfo* входит в группу, так называемых [информационных классов](#), и предназначен для получения разнообразной RTTI - информации в режиме сессии о пользователях, серверах, информационных базах и размещении каталогов. Класс является производным от класса [Объект \(Object\)](#) и наследует от него все свойства и методы.

Внимание! Ссылка на объект текущего класса хранится в поле [ИнфСессии / SessionInfo](#).

В классе *ИнфСессии / SessionInfo* вводятся следующие свойства:

- [Поле ИмяСервера / ServerName](#)
- [Поле ИмяПользователя / UserName](#)
- [Поле ОписаниеПользователя / UserDescription](#)
- [Поле ЗаписьПользователя / UserRecord](#)
- [Поле ИмяРоли / RoleName](#)
- [Поле ОписаниеРоли / RoleDescription](#)
- [Поле ЗаписьРоли / RoleRecord](#)
- [Поле ДемоРежим / DemoMode](#)
- [Поле ИнфБаза / Infobase](#)
- [Поле КаталогВременный / TempFolder](#)
- [Поле КаталогОбщий / SharedFolder](#)
- [Поле КаталогЛокальный / LocalFolder](#)
- [Поле КаталогПодключения / ConnectFolder](#)
- [Поле КаталогОбщихФайловИБ / CommonIBFolder](#)
- [Поле КаталогМоиДокументыMyDocumentsFolder](#)
- [Поле МаксТочностьРазныхЕдИзмD / iffUnitsMaxAccuracy](#)
- [Поле ЗадержкаОбработкиИзменений / ChangedRecordsHandleDelay](#)
- [Поле ИзмененныеЗаписиОбрабатыватьСразу / HandleChangedRecordsAtOnce](#)
- [Поле МоиИзмененияОбрабатыватьСразу / HandleChangedByMeRecordsAtOnce](#)
- [Поле ВводОперацииБезИмениПроекта / InputOperationWithoutProjectName](#)
- [Поле ВЖурналахПоказыватьОписанияПризнаков / JournalShowSignDescriptions](#)

Описание

ВводОперацииБезИмениПроекта :Логическое;
InputOperationWithoutProjectName :Logical;

Назначение

Свойство разрешает или запрещает при вводе операции в табличный журнал сохранять имя проекта в имени операции. По умолчанию значение поля равно False. т.е. имя проекта не сохраняется в имени операции.

Возможность сохранять имя проекта (True) доступна только программно.

Пример

```
proc Pl(Sender :Button);
...
if SessionInfo.InputOperationWithoutProjectName then
    Message("Имя проекта сохраняется в имени операции.");
else
    -- не сохраняется
fi;
....
end;
```

Описание

ВЖурналахПоказыватьОписанияПризнаков :Логическое;
JournalShowSignDescriptions :Logical;

Назначение

Свойство разрешает (True) или запрещает (False) в журналах в списке проводок и в диалогах ввода проводки или операции при отображении значений параметров типа справочник (кроме измерителей) показывать описание признаков или имени. По умолчанию значение свойства равно True.

Поле доступно на чтение и запись.

Пример

```
proc Pl(Sender :Button);  
...  
if SessionInfo.JournalShowSignDescriptions then  
    Message("Разрешено показывать описание признаков.");  
fi;  
....  
end;
```

Описание

ДемоРежим :Логическое;
DemoMode :Logical;

Назначение

Свойство позволяет узнать, работает ли сессия в демонстрационном режиме.

Демонстрационный режим включается, если нет лицензии на один или несколько проектов, входящих в состав информационной базы и требующих лицензирования.

В этом режиме программа работает с некоторыми ограничениями функционала. Например, при печати любого документа выводится надпись "Демо-Версия", не доступен сетевой режим работы, есть ограничения на количество обрабатываемых данных.

С помощью свойства **DemoMode** можно реализовать дополнительные ограничения демонстрационного режима на уровне проектов.

Поле доступно только на чтение.

Пример

```
proc Button1_OnClick(Sender :Button);
  if SessionInfo.DemoMode then
    Message("В демо-режиме данная функция недоступна");
    Exit;
  end;
  -- ...
end;
```

Описание

ЗадержкаОбработкиИзменений :Целое;
ChangedRecordsHandleDelay :Integer;

Назначение

Поле доступно на чтение и на запись. При считывании возвращает время задержки обработки измененных записей в секундах. Если поле работает на запись, то программным путем можно изменить величину задержки, заданную на странице "Обработка" в разделе ["Журналы и отчеты"](#) диалога "Настройка учета".

Пример

```
proc P1(Sender :Button);  
  ...  
  SessionInfo.ChangedRecordsHandleDelay=15000;  
  Message("Задержка= " +  
    Str(SessionInfo.ChangedRecordsHandleDelay));  
  ....  
end;
```

Описание

ЗаписьПользователя :Запись;
UserRecord :Record;

Назначение

Возвращает ссылку на запись, содержащую сведения о пользователе.

Пример

```
proc P1(Sender :Button);  
...  
  Message(Str(SessionInfo.UserRecord));  
...  
end;
```

Описание

ЗаписьРоли :Запись;
RoleRecord :Record;

Назначение

Возвращает ссылку на запись, содержащую сведения о роли пользователя.

Пример

```
proc P1(Sender :Button);  
...  
  Message(Str(SessionInfo.RoleRecord));  
...  
end;
```


Описание

ИзмененныеЗаписиОбрабатыватьСразу :Логическое;
HandleChangedRecordsAtOnce :Logical;

Назначение

Поле доступно на чтение и на запись. При записи можно изменить состояние флага "Сразу". При считывании возвращает значение "Истина", если на странице "Обработка" в разделе ["Журналы и отчеты"](#) диалога "Настройка учета" установлен флаг "Сразу", иначе - "Ложь". При установке флага "Сразу" измененные записи сразу включаются в отчет, иначе с заданной временной задержкой.

Пример

```
proc Pl(Sender :Button);
...
if SessionInfo.HandleChangedRecordsAtOnce:
  -- флаг "Сразу" установлен
  Message("Измененные записи будут обрабатываться сразу");
fi;
SessionInfo.HandleChangedRecordsAtOnce=false;
-- флаг снят
....
end;
```

Описание

ИмяПользователя :Строка;
UserName :String;

Назначение

Возвращает зарегистрированное в системе имя пользователя, под которым происходит подключение к информационной базе.

Пример

```
proc P1(Sender :Button);  
    ...  
    Message(SessionInfo.UserName);  
    ....  
end;
```

Описание

ИмяРоли :Строка;
RoleName :String;

Назначение

Возвращает имя роли пользователя.

Пример

```
proc P1(Sender :Button);  
    ...  
    Message(SessionInfo.RoleName);  
    ...  
end;
```

Описание

ИмяСервера :Строка;
ServerName :String;

Назначение

Возвращает имя сервера.

Пример

```
proc P1(Sender :Button);  
...  
  Message(SessionInfo.ServerName);  
...  
end;
```

Описание

ИнфБаза :ИнфБазы;
Infobase:BaseInfo;

Назначение

Возвращает объект класса **ИнфБазы / BaseInfo**.

Пример

```
proc P1(Sender :Button);  
  var  Ob : BaseInfo;  
  ...  
  Ob:= SessionInfo.Infobase;  
  Message("Количество проектов= " + Стр(Ob.ProjectCount));  
  ....  
end;
```

Описание

КаталогВременный :Строка;
TempFolder :String;

Назначение

Возвращает полный путь к временному каталогу "Temp".

Пример

```
proc P1(Sender :Button);  
  ...  
  Message(SessionInfo.TempFolder);  
  ...  
end;
```

Описание

КаталогЛокальный :Строка;
LocalFolder :String;

Назначение

Возвращает полный путь к рабочему каталогу текущей информационной базы.

Пример

```
proc P1(Sender :Button);  
...  
  Message(SessionInfo.LocalFolder);  
...  
end;
```

Описание

КаталогМоиДокументы :Строка;
MyDocumentsFolder :String;

Назначение

Возвращает полный путь к каталогу "МоиДокументы".

Пример

```
proc P1(Sender :Button);  
  ...  
  Message(SessionInfo.MyDocumentsFolder) ;  
  ...  
end;
```


Описание

КаталогОбщий :Строка;
SharedFolder :String;

Назначение

Возвращает полный путь к каталогу Shared.

Пример

```
proc P1(Sender :Button);  
...  
  Message(SessionInfo.SharedFolder);  
...  
end;
```

Описание

КаталогОбщихФайловИБ :Строка;
CommonIBFolder :String;

Назначение

Возвращает полный путь к каталогу с общими файлами информационной базы.

Пример

```
proc P1(Sender :Button);  
...  
  Message(SessionInfo.CommonIBFolder);  
...  
end;
```

Описание

КаталогПодключения :Строка;
ConnectFolder :String;

Назначение

Возвращает полный путь к каталогу подключения

Пример

```
proc P1(Sender :Button);  
...  
  Message(SessionInfo.ConnectFolder);  
...  
end;
```

Описание

МаксТочностьРазныхЕдИзм :Целое;
DiffUnitsMaxAccuracy :Integer;

Назначение

Определяет единицу измерений с максимальной точностью, и возвращает заданное для нее количество знаков после запятой.

Пример

```
proc P1(Sender :Button);  
  ...  
  Message("Максимальная точность= " +  
    Str(SessionInfo.DiffUnitsMaxAccuracy));  
  ....  
end;
```

Описание

`МоиИзмененияОбрабатыватьСразу` :Логическое;
`HandleChangedByMeRecordsAtOnce` :Logical;

Назначение

Поле доступно на чтение и на запись. При считывании возвращает значение Истина, если [в настройках программы](#) в разделе "Журналы и отчеты" на [странице "Обработка"](#) установлен флаг **Мои изменения включаются сразу**, иначе - Ложь. При записи можно изменить состояние флага. Если включена радиокнопка **С задержкой**, то флаг **Мои изменения включаются сразу** активен, и при его установке собственные измененные записи сразу обрабатываются и включаются в отчет.

Пример

```
proc Pl(Sender :Button);
...
if SessionInfo.HandleChangedByMeRecordsAtOnce:
    -- флаг установлен
    Message("Мои измененные записи будут обрабатываться сразу");
fi;
SessionInfo.HandleChangedByMeRecordsAtOnce=false;
-- флаг снят
....
end;
```

Описание

ОписаниеПользователя :Строка;
UserDescription :String;

Назначение

Возвращает описание пользователя (произвольное строковое выражение, характеризующее пользователя).

Пример

```
proc P1(Sender :Button);  
  ...  
  Message(SessionInfo.UserDescription);  
  ....  
end;
```

Описание

ОписаниеРоли :Строка;
RoleDescription :String;

Назначение

Возвращает описание роли (произвольное строковое выражение, характеризующее роль пользователя).

Пример

```
proc P1(Sender :Button);  
  ...  
  Message(SessionInfo.RoleDescription);  
  ...  
end;
```

Класс *ИнфСобытия* / *EventMethodInfo* наследует все свойства и методы от родительских классов [Объект / Object](#), [ИнфЧлена / MemberInfo](#) и [ИнфМетода / MethodInfo](#).

Класс *ИнфСобытия* относится к группе, так называемых [информационных классов](#) и предназначен для получения информации [о типах во время исполнения](#) программы на ТБ.Скрипт (run-time type information, RTTI).

Непосредственно в классе *ИнфСобытия* определены следующие свойства:

- [Поле ОписаниеОбработчика / HandlerDescription](#)

Описание

ОписаниеОбработчика :ИнфМетода;
HandlerDescription :MethodInfo;

Назначение

Поле позволяет получить ссылку на объект RTTI-класса **ИнфМетода**, описывающий обработчик события.

Пример

Класс *ИнфТипа / TypeInfo* относится к группе, так называемых [информационных классов](#), и предназначен для получения информации [о типах во время исполнения](#) (run-time type information, RTTI) программы на ТБ.Скрипт. Этот класс является общим предком для классов *ИнфКласса*, *ИнфПеречисления*, *ИнфПрограммногоТипа*, описывающих все возможные (объектные и неobjектные) типы Студии.

Класс является производным от родительских классов [Объект / Object](#) и [ИнфЧлена / MemberInfo](#) и наследует от них все свойства и методы.

В классе *ИнфТипа* определено:

- [Поле ТипПеременной / VarType](#)

Поле ТипПеременной / VarType (свойство объекта)

Описание

ТипПеременной :Целое;
VarType :Integer;

Назначение






Возвращает константу, описывающую тип переменной, информация о которой хранится в текущем объекте **ИнфТипа**. Обращение к данному полю аналогично вызову функции [ТипПеременной](#) класса **Система**. Список возможных значений приведен в разделе, посвященном этой функции.

Класс *ИнфЧлена / MemberInfo* входит в группу [информационных классов](#) и предназначен для получения информации [о типах времени исполнения](#) (run-time type information, RTTI) программы на ТБ.Скрипт. Этот класс содержит общеприменительные свойства и методы, которые используются при чтении RTTI.

Класс является базовым для других узкоспециализированных классов, описывающих RTTI конкретных сущностей, в частности, методов ([ИнфМетода](#)), событий ([ИнфСобытия](#)), классов ([ИнфКласса](#)) и неobjектных типов ([ИнфПеречисления](#), [ИнфПрограммногоТипа](#)).

Класс *ИнфЧлена* является производным от класса [Объект \(Object\)](#) и наследует от все свойства и методы. Непосредственно с помощью objектов этого класса описываются основные параметры членов классов, общие для членов разного рода (классов, типов, методов, событий).

В классе *ИнфЧлена / MemberInfo* вводятся следующие свойства, методы и перечислимые типы:

-  [Поле Имя / Name](#)
-  [Поле Владелец / Owner](#)
-  [Поле ОбластьВидимости / VisualArea](#)
-  [Поле Встроенный / Internal](#)
-  [Тип ОбластиВидимости / VisualAreaType.](#)

Поле Владелец / Owner

Описание

Владелец :ИнфКласса;
Owner :ClassInfo;

Назначение

Возвращает указатель на объект класса **ИнфКласса** с описанием класса, в котором объявлен данный член (свойства, метода или типа).

Поле доступно только на чтение.

Поле Встроенный / Internal

Описание

Встроенный :Логическое;
Internal :Logical;

Назначение

Возвращает TRUE, если данный член класса является встроенным, т.е. реализованным на уровне инструментария, а не прикладным программистом. В противном случае возвращает FALSE.

Пример

Класс Объект : [ИнфЧлена/MemberInfo](#)

Поле Имя / Name

Описание

Имя :Строка;
Name :String;

Назначение

Возвращает имя члена класса (свойства, метода или типа).

Пример

```
proc WhatIsThis(x :Variant);  
  var infcls: ClassInfo;  
  -- ClassInfo - наследник MemberInfo  
  infcls = x.ClassInfo;  
  Message(infcls.Name);  
end;
```

Описание

```
ОбластьВидимости :ТипОбластиВидимости;  
VisualArea :VisualAreaType;
```

Назначение

Возвращает одно из значений перечислимого типа [ТипОбластиВидимости](#), описывающее контекст (личный или публичный), в котором данный член класса описан.

Пример

В классе **ИнфЧлена** определен перечислимый тип **ТипОбластиВидимости**, описывающий область видимости членов класса. В нем введены следующие константы:

- **твЛично / vtPublic** - член доступен извне класса;
- **твПублично / vtPrivate** - член доступен только внутри класса.

Значения этого типа возвращает свойство [ОбластьВидимости](#).

Область видимости члена класса определяется тем, в какой области видимости (**Public** или **Private**) он был указан в исходном модуле класса.

Все классы, наследующие свойства и методы от класса [Объект](#), сгруппированы по своему назначению на группы.

В группу классов, которые обеспечивают программный доступ как к картотекам, так и к отдельным ее составляющим (подтаблице, столбцу, строке), входят следующие:








- [СтолбецТаблицы / TableColumn](#)
 - [СтолбецКартотеки / CardfileColumn](#)
 - [СтолбецПодтаблицы / SubCardfileColumn](#)
- [Картотека / CardFile](#)
- [ПодтаблицаКартотеки / SubCardfile](#)
- [СписокЗаписей / RecordList](#)














Класс *Картотека / CardFile*, производный от класса *Объект*, используется для работы с картотеками Студии. Класс *Картотека* наследует от родительского класса *Объект* свойства, описанные в разделе [Свойства базового класса Объект](#).

Внимание! Создать объект класса *Картотека* напрямую нельзя. Картотеки создаются самой системой на основе файлов с описанием их структуры и внешнего представления (BRO, TPL, COD файлы).


Непосредственно в классе *Картотека* определены следующие свойства и методы:

-  [Поле Записи / Records / Документы / Documents](#)
-  [Поле Текущий / Current / Текущая](#)
-  [Поле ТекущаяГруппа / CurrentGroup](#)
-  [Поле Упорядочивание / Order](#)
-  [Поле Фильтр / Filter](#)
-  [Поле ФильтрПользователя / UserFilter](#)
-  [Поле ИспользоватьФильтрПользователя / UserFilterOn](#)
-  [Поле ФильтрГрупп / GroupFilter](#)
-  [Поле ФильтрДерева / TreeFilter](#)
-  [Поле КореньДерева / TreeRoot](#)
-  [Поле Иерархическая / Hierarchical](#)
-  [Поле ПоказыватьИерархию / ShowHierarchy](#)
-  [Поле ПоказыватьКоличество / ShowCount](#)
-  [Поле ПоказыватьУдаленные / ShowDeleted](#)
-  [Поле ГруппыВначале / GroupsFirst](#)
-  [Поле ОписаниеИзПоля / DescrFromField](#)
-  [Поле ПоискПриНаборе / FindAtType](#)
-  [Поле МожноДобавлять / CanInsert](#)
-  [Поле МожноУдалять / CanDelete](#)
-  [Поле МожноПеремещать / CanMove](#)
-  [Поле МожноКопировать / CanCopy](#)
-  [Поле МожноРедактировать / CanEdit](#)
-  [Поле МожноОткрыть / CanOpen](#)
-  [Поле МожноИзменятьПризнакГруппы / CanGroupSignModify](#)
-  [Поле Редакторы / Editors](#)
-  [Поле РедакторыГрупп / GroupsEditors](#)
-  [Процедура ВыделитьЗаписи / SelectRecords](#)
-  [Процедура СнятьВыделениеЗаписей / DeselectRecords](#)
-  [Поле КоличествоВыделенных / SelectedCount](#)
-  [Поле Выделенные / Selected](#)
-  [Процедура СохранитьВыделенное / SaveSelection](#)
-  [Процедура ВосстановитьВыделенное / RestoreSelection](#)
-  [Поле ТекущийСтолбец / CurrentColumn](#)
-  [Поле ПозТекущегоСтолбца / CurrentColumnPos](#)
-  [Поле КоличествоСтолбцов / ColumnsCount](#)
-  [Поле Столбец / Column](#)
-  [Поле СтолбецПоПолю / ColumnByField](#)
-  [Поле ФиксированныхСтолбцов / FixedColumns](#)
-  [Функция ДобавитьСтолбец / AddColumn](#)
-  [Функция ВставитьСтолбец / InsertColumn](#)
-  [Процедура УдалитьСтолбец / DeleteColumn](#)
-  [Поле КоличествоПодтаблиц / SubCardCount](#)
-  [Поле Подтаблица / SubCard](#)
-  [Поле ПодтаблицаПоПолю / SubCardByField](#)
-  [Функция ДобавитьПодтаблицу / AddSubCard](#)

-  [Функция ВставитьПодтаблицу / InsertSubCard](#)
-  [Процедура УдалитьПодтаблицу / DeleteSubCard](#)
-  [Процедура Удалить / Delete](#)
-  [Процедура Восстановить / Undelete](#)
-  [Процедура НачатьРедактирование / BeginEdit](#)
-  [Процедура ЗавершитьРедактирование / EndEdit](#)
-  [Процедура Обновить / Update](#)

-  [Событие ПриПеремещении / OnMove](#)
-  [Событие ПриНажатии / OnClick](#)
-  [Событие ПриСозданииЗаписи / OnCreateRecord](#)
-  [Событие ПриОткрытииБланка / OnOpenBlank](#)
-  [Событие ПриРедактировании / OnEdit](#)
-  [Событие ПриЗаписи / OnPost](#)
-  [Событие ПриОтмене / OnCancel](#)
-  [Событие ПриОформлении / OnRearrange](#)
-  [Событие ПриСменеВыделения / OnChangeSelected](#)
-  [Событие ПриСменеГруппы / OnChangeGroup](#)
-  [Событие ПередИзменением / BeforeModify](#)
-  [Событие ПриИзменении / OnModify](#)
-  [Событие ПриРисованииСтроки / OnDrawRow](#)

и перечислимые типы:

-  [Тип ДействияОформления / RearrangeActions](#)
-  [Тип ДействияИзменения / ModifyActions](#)
-  [Тип ТипыКолонки / ColumnTypes](#)
-  [Тип ФорматыКолонки / ColumnFormats](#)
-  [Тип ФорматыМаскиПоиска / FindMaskFormats](#)

Перечислимый тип **ДействияИзменения / ModifyActions** в классе **Картотека** предназначен для выполнения предопределенных действий пользователя с записями, которые регламентируются следующими константами:

- **УдалениеЗаписи / DeleteRecord** - удаление записи или набора записей;
- **ВосстановлениеЗаписи / UndeleteRecord** - восстановление записи или набора записей;
- **ПеремещениеЗаписи / MoveRecord** - перемещение записи или набора записей;
- **КопированиеЗаписи / CopyRecord** - копирование записи или набора записей;
- **УстановкаПризнакаГруппы / RecordGroupOn** - преобразование простой записи в групповую;
- **СнятиеПризнакаГруппы / RecordGroupOff** - преобразование групповой записи в простую;
- **КомментированиеСобытия / CommentEvent** - комментирование операции или проводки в журнале;
- **РаскомментированиеСобытия / UncommentEvent** - раскомментирование операции или проводки (в журнале);
- **РаскрытиеСобытия / ExpandEvent** - раскрытие операции на проводки (в журнале);
- **СверткаСобытия / CollapseEvent** - свертка операции (в журнале).
- **БыстроеУдалениеЗаписи / FastDeleteRecord** - удаление записи без проверки ссылочной целостности.

Внимание. Данная константа используется *только в событии ПриИзменении/OnModify* классов Картотека, КартотекаШаблона и ДеревоКартотеки.

Константы, определенные в данном типе, используются в обработчиках событий:

- **ПередИзменением** - классов [Картотека](#), [КартотекаШаблона](#) и [ДеревоКартотеки](#), за исключением константы **БыстроеУдалениеЗаписи**;
- **ПриИзменении** - классов [Картотека](#), [КартотекаШаблона](#) и [ДеревоКартотеки](#).

Константы действий над столбцами (ДействияОформления)

В классе **Картотека** определен специальный перечислимый тип **ДействияОформления / RearrangeActions**, константы которого описывают возможные действия пользователя со столбцами, а также действия, связанные с изменением ширины панелей или включением/выключением видимости удаленных записей.

В данном типе определены следующие константы:

- **ШиринаСтолбца / ResizeColumn** - изменена ширина столбца;
- **ПеремещениеСтолбца / MoveColumn** - столбец перемещен на новое место;
- **ВидимостьСтолбца / VisibilityColumn** - столбец сделан видимым или скрыт;
- **СортировкаПоСтолбцу / SortByColumn** - изменен порядок сортировки столбца;
- **СменаИерархическогоВида / ShowHierarchyChanged** - выполнена команда "Иерархический вид", т.е. произошла смена режима выводить/не выводить записи в иерархическом виде;
- **ДругоеДействиеОформления / OtherRearrangeActions** - действие, при котором изменяется ширина панелей или включается/выключается видимость удаленных записей.

Константы данного типа используются в обработчике **ПриОформлении** следующих классов:

- [Картотека](#)
- [КартотекаШаблона](#)
- [ПодтаблицаКартотеки](#)
- [ПодтаблицаШаблона](#)

Перечислимый тип *ТипыКолонки* / *ColumnTypes*, определенный в классе *Картотека* предназначен для задания типов колонок таблиц в картотеках.

В данном типе определены следующие константы:

- **КолонкаПоле | FieldColumn** - колонка, в которой содержатся поля ввода-вывода, не являющиеся вычислимыми полями;
- **КолонкаВычислимоеПоле | CalcFieldColumn** - колонка, содержащая вычисляемые поля, которые не связаны напрямую ни с одной переменной, ввод и вывод в них осуществляются на прикладном уровне с помощью обработчиков событий.

Константы, определенные в данном типе, используются в свойстве [ТипКолонки](#) класса *Картотека*.

Перечислимый тип *ФорматыКолонки|ColumnFormats*, определенный в классе *Картотека* предназначен для задания форматов колонок таблиц в картотеках.

В данном типе определены следующие константы:

- **ФорматОбщий | CommonFormat** - формат поля ввода-вывода без конкретизации типа;
- **ФорматСтроки | StringFormat** - формат поля строкового типа;
- **ФорматЧисла | NumericFormat** - формат поля, используемого для целых и числовых переменных;
- **ФорматДаты | DateFormat** - формат поля типа Дата, используемого для ввода дат и/или времени;
- **ФорматЛогический | LogicalFormat** - формат поля логического типа, принимающего значения Истина или Ложь;
- **ФорматПеречислимый | EnumFormat** - формат поля перечислимого типа;
- **ФорматСсылочный | ReferenceFormat** - формат поля ссылочного типа, используемого для отображения ссылки на другой документ.

Константы, определенные в данном типе, используются в свойстве [ФорматКолонки](#) класса *Картотека*.

Перечислимый тип *ФорматыМаскиПоиска|FindMaskFormats*, установленный в классе *Картотека*, управляет тем, как будет формироваться маска для поиска/фильтра строки по умолчанию.

В данном типе определены следующие константы:

- **FindWithin** - поиск по символам, содержащимся в середине текста (*abc*);
- **FindFromBegin** - поиск по начальным символам (abc*);
- **FindFromEnd** - поиск по конечным символам (*abc);
- **FindExact** - поиск только по заданной строке (abc).

Внимание. По умолчанию, а также при отсутствии соответствующего столбца в мастере фильтров формат считается равным **FindWithin**.

Константы, определенные в данном типе, используются в свойстве *ФорматМаскиПоиска* в классах [СтолбецТаблицы](#) и [КлеткаШаблона](#)

Описание

Выделенные[Индекс: Целое] : [Запись](#);
Selected[Index: Integer] : [Record](#);

Назначение

Данное поле позволяет получить ссылку на одну из выделенных записей (документов) по ее номеру. Индекс должен лежать в пределах от 1 до [количества выделенных записей](#)

Поле доступно только на чтение.

Пример

```
proc кнУдалить_ПриНажатии (Sender :String);
var i :Integer;
if ВопрДаОтказ('Удалить выделенные записи?') = cmOk then
  BeginTransaction([Справочники.КурсыВалют]);
  try
    for i = 1..Cardfile.SelectedCount do
      Hint('Удаление курса валюты на ' + Str(Cardfile.Selected[i].Дата) + '...');
      Cardfile.Selected[i].Delete;
    end;
    Hint('Обновление картотеки...');
    EndTransaction;
  except
    AbortTransaction;
    raise;
  end;
end;
end;
```

Описание

ГруппыВначале :Логическое;
GroupsFirst:Logical;

Назначение

При настройке иерархической картотеки позволяет задать метод сортировки элементов.

Если значение поля равно TRUE, группы будут находиться перед простыми элементами (в верхней части таблицы), в случае же FALSE группы сортируются наравне с элементами.

По умолчанию, группы располагаются в верхней части списка (поле имеет значение TRUE).

Пример

```
-- в зависимости от состояния флага
-- таблица картотеки сортируется либо с приоритетом
-- групп, либо без него
proc CheckBox1OnChange(Sender :CheckBox);
    Картотека..ГруппыВначале = Sender.State;
end;
```

Описание

Записи : Класс [Запись\[\]](#);
Records :Class [Record\[\]](#);

Назначение

Позволяет задать или узнать классы документов, по которым строится запрос в картотеке.

Использование этого поля более предпочтительно, чем разыменовывание поля [Запрос](#) объекта **ФормаКартотеки** и обращение к свойству **Записи** этого запроса, так как все изменения в запросе действуют лишь до первого изменения содержимого картотеки.

Пример

```
-- код класса формы картотеки
-- по нажатию кнопки отображаем в картотеке приходные накладные
proc кнПриход_ПриНажатии (Sender :String);
    Cardfile.Documents = [Пример.Накладные.Приходные];
end;
-- по нажатию другой кнопки отображаем в картотеке расходные накладные
proc кнРасход_ПриНажатии (Sender :String);
    Cardfile.Documents = [Пример.Накладные.Расходные];
end;
```

Описание

Запрос : [Запрос](#);
Query : [Query](#);

Назначение

Поле позволяет получить доступ к объекту [Запрос/Query](#), ассоциированному с данной картотекой. Манипулируя свойствами запроса, можно управлять поведением картотеки и получать сведения о ее текущем состоянии, в частности, определять текущий (выделенный) документ.

Следует иметь в виду, что в иерархической картотеке в режиме отображения иерархии (то есть, когда одновременно в таблице картотеки выводится содержимое только одной группы, одного уровня иерархии) запрос содержит только документы открытой группы. Поэтому попытка присвоить свойству [Текущий](#) запроса некоторый документ, не входящий в открытую группу, генерирует исключение. Иными словами, в запросе иерархической картотеки можно программно менять текущий документ только в пределах открытой группы. Если необходимо переместиться из одной группы в другую, следует изменить свойство **Текущий** непосредственно у картотеки шаблона. При этом система автоматически закроет старый запрос и построит новый.

Внимание! Свойства класса **Запрос - Записи, Фильтр, Упорядочивание** - имеются также и в классе **КартотекаШаблона**. Изменение этих свойств в запросе действуют лишь до первого изменения содержащихся в картотеке записей. После каждого изменения картотеки система инициализирует вышеуказанные свойства поля **Запрос** соответствующими значениями одноименных свойства объекта **КартотекаШаблона**. Таким образом, если необходимо перманентно изменить какое-либо из свойств, это следует делать через поле **Картотека**, а не **Запрос**.

Пример

```
Card: TemplateCardfile; -- объект картотеки на шаблоне
-- по нажатию кнопки на бланке удаляем текущую запись
-- из объекта Card класса КартотекаШаблона
proc кнУдалить_ПриНажатии (Sender :String);
    Card.Query.Current.Delete;
end;
```

Описание

Иерархическая :Логическое;
Hierarchical :Logical;

Назначение

Свойство позволяет узнать, является ли картотека иерархической, или нет. Если свойство равно True, то картотека является иерархической. Поле доступно на чтение и запись.

От значения данного свойства зависит работа свойства [ShowHierarchy](#).

Пример

```
var Cardfile1 :Cardfile;  
...  
  if Cardfile1.Hierarchical then  
    Cardfile1.ShowHierarchy = True;  
  fi;  
end;
```

Описание

ИпользоватьФильтрПользователя : Логическое;
UseUserFilter : Logical;

Назначение

Данное поле позволяет узнать, включен ли фильтр пользователя, а также включать/отключать его.

Пример

```
-- процедура включает/отключает "фильтр по Москве"
proc ButtonClick(S:String);
  if Картотека.ФильтрПользователя = '' then
    Картотека.ФильтрПользователя = 'COOTB(Адрес, "*Москва*")';
    Картотека.ИпользоватьФильтрПользователя = TRUE;
  else
    Картотека.ФильтрПользователя = '';
    -- автоматически сбрасывает ИпользоватьФильтрПользователя
  end;
end;
```

Описание

КоличествоВыделенных : Целое;
SelectedCount : Integer;

Назначение

Данное поле содержит количество выделенных в данный момент записей картотеки. Поле доступно только на чтение.

Пример

```
proc кнУдалить_ПриНажатии (Sender :String);
var i :Integer;
if ВопрДаОтказ('Удалить выделенные записи?') = смOk then
  BeginTransaction([Справочники.КурсыВалют]);
  try
    for i = 1..Cardfile.SelectedCount do
      Hint('Удаление курса валюты на ' +
        Str(Cardfile.Selected[i].Дата) + '...');
      Cardfile.Selected[i].Delete;
    end;
    Hint('Обновление картотеки...');
    EndTransaction;
  except
    AbortTransaction;
    raise;
  end;
end;
end;
```


Описание

КоличествоПодтаблиц : Целое;
SubCardCount : Integer;

Назначение

Позволяет узнать количество подтаблиц в картотеке, определяемое MTL-описанием соответствующей записи. Поле доступно только на чтение.

Непосредственно подтаблицы можно получить через свойство [Подтаблица](#) или [ПодтаблицаПоПолю](#).

Пример

```
-- цикл по подтаблицам
for i = 1..Cardfile.SubCardCount do
  -- обработка таблиц
end;
```

Описание

КоличествоСтолбцов : Целое;
ColumnsCount : Integer;

Назначение

Данное свойство содержит количество столбцов в картотеке. Поле доступно только на чтение.

Для программного добавления/удаления столбцов необходимо пользоваться методами [ДобавитьСтолбец](#), [ВставитьСтолбец](#), [УдалитьСтолбец](#).

Пример

```
-- симулируем метод ColumnByField
func FieldByName(Name: String):CardFileColumn;
var i: integer;
  for i=1..CardFile.ColumnsCount do
    if Name = CardFile.Column[i].FieldName then
      return CardFile.Column[i];
    end;
  end;
  return nil;
end;
```

Описание

КореньДерева: Строка;
TreeRoot: String;

Назначение

Данное поле позволяет узнать и изменить название корневой группы картотеки. По умолчанию, корень имеет имя "Картотека".

Пример

```
Картотека.КореньДерева = "Контрагенты";
```

Описание

МожноДобавлять :Логическое;
CanInsert :Logical;

Назначение

Поле определяет, можно ли интерактивно добавлять записи в картотеку.

В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге [свойств картотеки](#) (на странице "Правка").

Пример

```
Cardfile1.CanInsert = CheckBox1.State;
```

Описание

МожноИзменятьПризнакГруппы :Логическое;
CanGroupSignModify :Logical;

Назначение

Поле определяет, можно ли интерактивно преобразовывать запись картотеки в группу и обратно.

В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге [свойств картотеки](#) (на странице "Правка").

Пример

```
Cardfile1.CanGroupSignModify = CheckBox7.State;
```

Описание

МожноКопировать :Логическое;
CanCopy :Logical;

Назначение

Поле определяет, можно ли интерактивно копировать записи в картотеке. Поле доступно на чтение и запись.

В дизайн-режиме данное свойство задается с помощью флага **Копировать записи** в диалоге ["Свойства картотеки"](#) на странице ["Правка"](#).

Пример

```
var Cardfile1 :Cardfile;  
...  
Cardfile1.CanCopy = False;
```

Описание

МожноОткрыть :Логическое;
CanOpen :Logical;

Назначение

Поле определяет, можно ли открыть бланк-редактор для редактирования записи картотеки. Поле доступно на чтение и запись.

В дизайн-режиме данное свойство задается с помощью флага **Редактировать в бланке** в диалоге ["Свойства картотеки"](#) на странице ["Правка"](#).

Пример

```
var Cardfile1 :Cardfile;  
...  
Cardfile1.CanOpen = False;
```

Описание

МожноПеремещать :Логическое;
CanMove :Logical;

Назначение

Поле позволяет программным способом включить|отключить режим перемещения записей в картотеке. Если свойство равно True, то записи разрешается перемещать.

Поле доступно на чтение и запись.

В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге "[Свойства картотеки](#)" на странице "[Правка](#)".

Пример

```
var Cardfile1 :Cardfile;  
...  
Cardfile1.CanMove = False;
```


Описание

МожноРедактировать :Логическое;
CanEdit :Logical;

Назначение

Поле позволяет программным способом включить|отключить режим редактирования значений в клетках картотеки. Если свойство равно True, то редактирование разрешено.

Поле доступно на чтение и запись.

В дизайн-режиме данное свойство задается с помощью флага **Редактировать в ячейке** в диалоге ["Свойства картотеки"](#) на странице ["Правка"](#).

Пример

```
var Cardfile1 :Cardfile;  
...  
Cardfile1.CanEdit = True;
```

Описание

МожноУдалять :Логическое;
CanDelete :Logical;

Назначение

Поле определяет, можно ли интерактивно удалять записи из картотеки.

В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге [СВОЙСТВ картотеки](#) (на странице "Правка").

Пример

```
Cardfile1.CanDelete = CheckBox2.State;
```

Описание

ОписаниеИзПоля : Строка;
DescrFromField : String;

Назначение

Поле позволяет узнать или указать имя поля записи, содержимое которого будет использоваться для обозначения записей. В частности, это значение, преобразованное в строку, будет выводиться в диалоговых окнах с запросом на подтверждение операции с записью. Например, при удалении: "Валюта: рубли. Удалить текущую запись?" – здесь значение 'рубли' взято из поля описания.

В случае иерархической картотеки описание из того же поля применяется для отображения групп в дереве.

Если оставить поле **ОписаниеИзПоля** пустым, то по умолчанию используется содержимое поля **DocID** записи.

Пример

```
-- процедура RadiolOnChange назначена обработчиком
-- события для двух переключателей 'Имя' и 'Код';
-- в зависимости от положения переключателей,
-- группы в дереве картотеки идентифицируются либо
-- по имени, либо по коду;
-- Caption каждого переключателя соответствует
-- имени поля записи
proc RadiolOnChange(Sender :RadioButton);
    Картотека1. ОписаниеИзПоля = Sender.Caption;
end;
```

Описание

Подтаблица [Индекс:Целое] : [ПодтаблицаКартотеки](#);
SubCard [Index:Integer] : [SubCardfile](#);

Аргументы

Индекс - номер подтаблицы. Подтаблицы нумеруются от 1. Указание индекса несуществующей подтаблицы вызывает ошибку.

Назначение

Данное поле позволяет по номеру подтаблицы получить соответствующий объект класса [ПодтаблицаКартотеки](#).

Поле доступно только на чтение. Количество и состав подтаблиц задаются с помощью MTL-описания того класса записей, который отображается в данной картотеке.

Пример

```
-- цикл по подтаблицам
for i = 1..Cardfile.SubCardCount do
  -- цикл по выделенным структурам в подтаблицах
  for j = 0..Cardfile.SubCard[i].SelectedCount-1 do
    -- обработка Cardfile.SubCard[i].Selected[j]
  end;
end;
```

Описание

`ПодтаблицаПоПолю[Название:Строка] : ПодтаблицаКартотеки;`
`SubCardByField[Name:String] : SubCardfile;`

Аргументы

Название - имя структурного или многозначного поля (подтаблицы). Указание индекса несуществующей подтаблицы вызывает ошибку.

Назначение

Данное поле позволяет по имени структурного или многозначного поля (подтаблицы) получить соответствующий объект класса [ПодтаблицаКартотеки](#).

Поле доступно только на чтение. Количество и состав подтаблиц задаются с помощью MTL-описания того класса записей, который отображается в данной картотеке.

Пример

```
var j:integer;
var sc:SubCardfile;
sc = Cardfile.SubCardByField["Позиции"];
-- цикл по выделенным структурам в подтаблице
for j = 0..sc.SelectedCount-1 do
  -- обработка sc.Selected[j]
end;
```

Описание

ПозТекущегоСтолбца : Целое;
CurrentColumnPos : Integer;

Назначение

Данное свойство позволяет узнать номер текущего (видимого) столбца картотеки, а также сделать текущим другой видимый столбец. Текущим (выделенным) считается столбец, в котором находится курсор. Свойство доступно на чтение и запись.

Внимание! Свойство **ПозТекущегоСтолбца**=CurrentColumn.Index, только в том случае, когда видимыми являются все столбцы картотеки. В общем случае номер видимого столбца ПозТекущегоСтолбца не совпадает с порядковым номером столбца картотеки CurrentColumn.Index.

Валидность диапазона при установке свойства не проверяется, поэтому, чтобы встать на последний столбец можно написать CurrentColumnPos = 999; или задать общее [количества столбцов](#) (см. пример ниже).

Пример

```
-- выделяем клетку в самом последнем столбце  
Cardfile1.CurrentColumnPos = Cardfile1.ColumnsCount;
```

Описание

Поле ПоискПриНаборе :Логическое;
FindAtType :Logical;

Назначение

Свойство разрешает или запрещает открывать диалог поиска в картотеке. Если свойство равно True, то в режиме редактирования (inplace-редактор) при нажатии на любую клавишу открывается диалог. Для изменения данных в ячейке необходимо нажать клавишу *Enter*.

Поле доступно на чтение и запись.

Данное свойство по своему действию эквивалентно флагу **Поиск при наборе** в диалоге "[Свойства картотеки](#)" на странице "[Правка](#)".

Пример

```
var Cardfile1 :Cardfile;  
...  
Cardfile1.FindAtType = False;
```

Описание

ПоказыватьИерархию : Логическое;
ShowHierarchy: Logical;

Назначение

Данное свойство позволяет узнать, включен ли режим показа иерархического представления картотеки, а также включать/отключать его. При значении ShowHierarchy = True в окне картотеки отображаются записи текущего уровня (одной группы или корня картотеки), иначе (False) - в так называемом плоском виде, когда отображаются все записи картотеки из всех групп с учетом фильтра, вне зависимости от их уровня вложенности.

Данное свойство используется только для *иерархических картотек*, у которых свойство [Hierarchical](#) = True. Если свойство Hierarchical = False, то независимо от значения данного свойства отобразить иерархию невозможно, поэтому в режиме сессии команда **Иерархический вид** в выпадающем меню картотеки будет недоступна.

Пример

```
-- код картотечной формы
-- ...
-- процедура включает/отключает показ дерева
proc ButtonClick(B:Button);
  if Картотека.ПоказыватьИерархию then
    ПоказыватьДерево = NOT ПоказыватьДерево;
  end;
end;
```


Описание

ПоказыватьИтоги :Логическое;
ShowTotal :Logical;

Назначение

Свойство разрешает или запрещает показывать итоговую строку в картотеке. Поле доступно на чтение и запись.

Данное свойство по своему действию эквивалентно флагу **Показывать итоги** в диалоге ["Свойства картотеки"](#) на странице ["Вид"](#).

Пример

```
var Cardfile1 :Cardfile;  
...  
Cardfile1.ShowTotal = True;
```

Описание

`ПоказыватьКоличество` :Логическое;
`ShowCount` :Logical;

Назначение

Свойство позволяет программным способом показывать (True) количество записей в картотеке или запретить показ (False). Поле доступно на чтение и запись.

Данное свойство по своему действию эквивалентно флагу **Показывать количество** в диалоге ["Свойства картотеки"](#) на странице ["Вид"](#).

Пример

```
var Cardfile1 :Cardfile;  
...  
Cardfile1.ShowCount = True;
```

Описание

ПоказыватьУдаленные : Логическое;
ShowDeleted : Logical;

Назначение

Данное поле позволяет узнать, включен ли режим просмотра удаленных записей или нет, а также включать/отключать его.

Пример

```
-- процедура включает/отключает
-- видимость удаленных записей
proc ButtonClick(S:String);
    Картотека.ПоказыватьУдаленные =
        NOT Картотека.ПоказыватьУдаленные;
end;
```

Описание

Редакторы :Класс [ФормаБланка](#)[];
Editors :Class [BlankForm](#)[];

Назначение

Данное поле предоставляет доступ к массиву классов бланков-редакторов, назначенных для данной картотеки. Массив можно считывать поэлементно и целиком. Поэлементное присвоение новых значений не допускается, однако можно записать в поле целиком новый массив классов. Количество элементов в массиве можно получить с помощью функции [ДлинаМассива](#) класса Система.

Поле позволяет узнать и изменить перечень классов бланков-редакторов картотеки, использующихся для *обычных (негрупповых) записей*. Для того чтобы узнать или изменить перечень классов бланков-редакторов *групповых записей* применяется поле [РедакторыГрупп](#).

Пример

```
var n: integer;  
  n = LengthOfArray(Cardfile.Editors);  
  if n > 1 then  
    Cardfile.Editors = [ДвижениеСредств];  
  end;
```

Описание

```
РедакторыГрупп :Класс ФормаБланка[];  
GroupsEditors :Class BlankForm[];
```

Назначение

Данное поле предоставляет доступ к массиву классов бланков-редакторов, назначенных для групп данной картотеки. Массив можно считывать поэлементно и целиком. Поэлементное присвоение новых значений не допускается, однако можно записать в поле целиком новый массив классов. Количество элементов в массиве можно получить с помощью функции [ДлинаМассива](#) класса Система.

Поле позволяет узнать и изменить перечень классов бланков-редакторов картотеки, использующихся для *групповых записей*. Для того чтобы узнать или изменить перечень классов бланков-редакторов *обычных (негрупповых) записей* применяется поле [Редакторы](#).

Пример

```
var n: integer;  
n = LengthOfArray(Cardfile.GroupsEditors);  
if n = 0 then  
    Cardfile.GroupsEditors = [ДвижениеСредств];  
end;
```

Описание

Столбец[Индекс: Целое] :[СтолбецКартотеки](#);
Column [Index: Integer] :[CardfileColumn](#);

Аргументы

Индекс - задает позицию, в которую будет вставлен новый столбец. Индекс может изменяться в пределах от 1 до [количества столбцов](#).

Назначение

Данное поле позволяет получить доступ к конкретному столбцу картотеки по его номеру. Поле доступно только на чтение.

Пример

```
-- симулируем метод ColumnByField
func FieldByName(Name: String):CardFileColumn;
var i: integer;
  for i=1..CardFile.ColumnsCount do
    if Name = CardFile.Column[i].FieldName then
      return CardFile.Column[i];
    end;
  end;
  return nil;
end;
```

Описание

СтолбецПоПолю[Имя: Строка] : [СтолбецКартотеки](#);
ColumnByField[Name: String] : [CardfileColumn](#);

Аргументы

Имя - имя столбца картотеки.

Назначение

Данное поле позволяет получить доступ к конкретному столбцу картотеки по имени поля, содержащегося в нем. Поле доступно только на чтение.

Пример

```
proc BoldByName;  
var cc: CardFileColumn;  
  cc = CardFile.ColumnByField["Сумма"];  
  cc.Font.Bold = TRUE;  
end;
```

Описание

ТекущаяГруппа : [Запись](#);

CurrentGroup: [Record](#);

Назначение

Данное свойство позволяет узнать или установить запись, задающую текущую группу в иерархической картотеке.

Пример

```
-- после каждого перемещения курсора по картотеке
-- проверяем, не является ли запись группой, и если да -
-- переходим в нее
proc CardfileFirm_OnMove(D:Document);
  var x: Справочники.Товары;
  try
    x = cardfile.Current;
    if x.IsGroup then
      cardfile.CurrentGroup = x;
    end;
  except
  end;
end;
```


Описание

Текущий : [Документ](#);
Current : [Record](#);
Текущая : [Запись](#);

Назначение

Данное свойство позволяет узнать и установить текущую запись картотеки.

Пример

```
-- после каждого перемещения курсора по картотеке
-- выводим название текущей записи
proc CardfileFirm_OnMove(D:Document);
  var x: Справочники.ЮрЛицо;
  try
    x = cardfile.Current;
    hint(x.ПолнИмя);
  except
    hint('');
  end;
end;
```

Описание

ТекущийСтолбец : [СтолбецКартотеки](#);

CurrentColumn : [CardfileColumn](#);

Назначение

Поле возвращает ссылку на текущий столбец картотеки, содержащий выделенную клетку. Текущим (выделенным) считается столбец, в котором находится курсор. Свойство доступно на чтение и запись.

Внимание! При попытке выделить невидимый столбец или столбец, отсутствующий в таблице картотеки, текущая позиция *не меняется*.

Пример

```
func Картотека_ПриЗаписи(Record :Record) :Logical;  
  if Record.Имя = '' then  
    CardFile.CurrentColumn = CardFile.ColumnByField['Имя'];  
    Message('Не указано наименование счета.');    Result = False;  
  elseif Record.Тип = '' then  
    CardFile.CurrentColumn = CardFile.ColumnByField['Тип'];  
    Message('Не указан тип счета.');    Result = False;  
  else  
    Result = True;  
  end;  
end;
```

Описание

Упорядочивание : Строка;
Order : String;

Назначение

Позволяет задать или узнать порядок сортировки документов в картотеке. В строке указывается название одного или нескольких полей (разделенных точкой с запятой), по которым должны быть отсортированы документы в запросе. После имени каждого поля можно указать символ '+' (плюс) или '-' (минус). Плюс задает сортировку по возрастанию, минус – по убыванию.

Использование этого поля более предпочтительно, чем разыменовывание поля [Запрос](#) объекта **ФормаКартотеки** и обращение к свойству **Упорядочивание** этого запроса, так как все изменения в запросе действуют лишь до первого изменения содержимого картотеки.

Пример

```
-- код формы бланка
-- по нажатию кнопки меняем порядок сортировки
proc OnButtonClick(Button1 :Button);
  Cardfile.Order = "дата;время;филиал";
end;
```

Описание

ФиксированныхСтолбцов : Целое;
FixedColumns : Integer;

Назначение

Данное свойство позволяет узнать и установить количество фиксированных столбцов в картотеке. Фиксированные столбцы остаются у левого края картотеки при ее прокрутке по горизонтали.

Пример

```
proc ButtonClick(S:String);  
    Картотека.ФиксированныхСтолбцов = 1;  
end;
```

Описание

Фильтр : Строка;
Filter : String;

Назначение

Данное поле позволяет установить и прочитать текущее значение фильтра, задающего условие отбора документов, отображаемых в картотеке.

Общие правила составления фильтрующего выражения совпадают с правилами [установки пользовательских фильтров](#) в картотеках. Кроме [стандартных функций](#), которые можно использовать в выражениях фильтров, допускается применять [имена служебных полей](#) и так называемые [кванторы](#).

Фильтр, заданный с помощью этого поля, недоступен пользователю в отличие от [пользовательского фильтра](#), который может им меняться интерактивно через диалог.

Следует иметь в виду, что выражения некоторых видов выполняются в фильтрах оптимизированным способом – посредством одного SQL-запроса, в то время как в остальных случаях выполняется непосредственный анализ содержимого записей. К оптимизированным выражениям для фильтров относятся:

- сравнение полей с константами;
- сравнение полей по разыменованным ссылкам с константами;
- функция Match, если ее аргумент не содержит функций, а только текст и подстановочные символы;
- функция Pos;
- выражения вида:
 <ИмяПодтаблицы>[<Выражение>].<ИмяПоляПодтаблицы> <ОперацияСравнения> <Выражение>
 и
 <ИмяМассива>[<Выражение>] <ОперацияСравнения> <Выражение>

Пример

```
-- процедура включает/отключает "фильтр по Москве"
proc ButtonClick(S:String);
  if Картотека.Фильтр = '' then
    Картотека.Фильтр = 'COOTB(Адрес, "*Москва*")';
  else
    Картотека.Фильтр = '';
  end;
end;
```

Описание

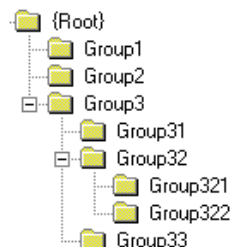
ФильтрГрупп : [СписокЗаписей](#);
GroupFilter : [RecordList](#);

Назначение

Данное поле позволяет установить и прочесть текущее значение фильтра групп, определяющего условие отбора записей по их расположению в иерархии групп. Иными словами, с помощью данного поля можно указать, элементы каких групп должны отображаться в дереве картотеки. На табличную часть картотеки данное поле не влияет. Поэтому для фильтрации записей по их принадлежности к тем или иным группам необходимо использовать свойство **Фильтр** с условием на поле **GroupPath** класса записи.

Поле хранит значение типа **СписокЗаписей**. Список задается в виде массива ссылок на записи, которые могут быть как групповыми (их свойство **IsGroup** равно TRUE), так и негрупповыми (простыми элементами). После заполнения списка дерево картотеки будет отображать только те группы, которые содержат записи из списка, и все их родительские группы различных уровней, вплоть до вышележащей группы, которая включает в себя все отобранные для отображения группы. Если список пуст, фильтрация по группам не выполняется. Дублирование в списке не допускается (одна и та же группа или запись не может быть указана два и более раза).

Например, рассмотрим следующую иерархию групп:



Задав фильтр групп "[Group1,Group31,Group322]", получим такой результат:

- на уровне корня картотеки останется Group1 и Group3;
- на уровне Group3 будут показаны Group31 и Group32;
- на уровне Group32 останется только группа Group322.

В результате изменения фильтра групп у дерева картотеки при необходимости соответствующим образом меняется корень. Например, если фильтр задает отбор лишь элементов группы Икс, то именно она станет корнем дерева. В более сложных случаях система выбирает в качестве корня наиболее низко лежащую группу, охватывающую все элементы, подходящие под фильтр. В частности, если бы для вышерассмотренной иерархии был применен групповой фильтр "[Group31,Group322]", то в качестве корня была бы выбрана запись Group3.

Если разработчику необходимо изменить правила отображения отфильтрованных групп и записей за счет назначения другой групповой записи, он может сделать это с помощью свойства [Корень](#) объекта **СписокЗаписей**.

Пример

```
proc УстановитьОбзорГруппы(Карт: Тест.Картотеки.Товары; ГруппаТоваров:
    Тест.Справочники.Товар);
    Карт.Картотека.ФильтрГрупп.Очистить;
    Карт.Картотека.ФильтрГрупп.Добавить(ГруппаТоваров);
end;
```

Описание

ФильтрДерева: Строка;
TreeFilter: String;

Назначение

Данное поле позволяет установить и прочесть текущее значение дополнительного пользовательского фильтра, задающего условие отбора групп документов.

Пример

```
Cardfile1: Cardfile  
ФильтрНаДерево: Строка;  
...  
ФильтрНаДерево = CardTree1.TreeFilter;
```

Описание

ФильтрПользователя : Строка;
UserFilter : String;

Назначение

Данное поле позволяет установить и прочесть текущее значение пользовательского фильтра, задающего дополнительное условие отбора документов, отображаемых в картотеке. Пользовательский фильтр применяется к набору документов, отвечающих условию общего фильтра (свойство [Фильтр](#)). Пользовательский фильтр вступает в действие, только когда поле [ИспользоватьФильтрПользователя](#) содержит значение TRUE.

В отличие от общего фильтра пользовательский фильтр может быть интерактивно изменен пользователем с помощью [диалога "Установка фильтра"](#).

Пример

```
Пример
-- процедура включает/отключает "фильтр по Москве"
proc ButtonClick(S:String);
  if Картотека.ФильтрПользователя = '' then
    Картотека.ФильтрПользователя = 'COOTB(Адрес, "*Москва*")';
    Картотека.ИспользоватьФильтрПользователя = TRUE;
  else
    Картотека.ФильтрПользователя = '';
    -- автоматически сбрасывает ИспользоватьФильтрПользователя
  end;
end;
```


Описание

Восстановить;
Undelete;

Назначение

Восстанавливает все выделенные записи. Если выделенных записей нет, восстанавливает текущую запись. Если нет ни выделенных записей, ни текущей записи, происходит ошибка.

Восстанавливаемые записи должны быть помечены как удаленные. В противном случае также возникает ошибка. Иными словами, нельзя восстановить неудаленную запись.

Удаленные записи видны в окне картотеки только при ее соответствующей настройке.

Пример

```
proc ButtonUnDelOnClick(Sender :Кнопка);  
  try  
    Cardfile1.UnDelete;  
  except  
  end;  
end;
```

Описание

```
ВосстановитьВыделенное(Поле :Строка; ИмяФайла :Строка);  
RestoreSelection(Field :String; FileName :String);
```

Аргументы

Поле - имя поля записи (причем, может быть указано любое поле), которое используется для идентификации картотеки;

ИмяФайла - имя текстового файла, содержащего записи картотеки, которые требуется выделить.

Назначение

Процедура производит выделение записей картотеки, которые были ранее записаны в заданном по имени текстовом файле с помощью процедуры [СохранитьВыделенное / SaveSelection](#).

Описание

```
ВыделитьЗаписи([Записи : Запись\[\] ]);  
SelectRecords([Records : Record\[\] ]);
```

Аргументы

Записи - необязательный параметр, массив записей, которые должны быть выделены.

Назначение

Выделяет массив записей, заданный в первом параметре, или все записи, если параметр опущен.

Пример

```
var локКартотека           :Cardfile;  
var локКолВоОбработанных  :Integer;  
var локОбработанныеПроцессы :Данные.Процесс[];  
  
proc ВыделениеКартотеки;  
  if локОбработанныеПроцессы <> nil then  
    локКартотека.SelectRecords(локОбработанныеПроцессы);  
  end;  
end;  
end;
```

Описание

```
ЗавершитьРедактирование[ (ПринятьИзменения :Логическое) ];  
EndEdit[(AcceptChanges: Logical)];
```

Аргументы

ПринятьИзменения - определяет, следует ли принять или отклонить сделанные изменения.

Назначение

Закрывает встроенный (in-place) редактор, открытый ранее с помощью [BeginEdit](#) или пользователем. Если **ПринятьИзменения** равно TRUE, изменения (если они были) передаются в обработчик **OnVerify** соответствующего столбца картотеки (если этот обработчик установлен) и затем сохраняются во временном представлении текущей записи. Впоследствии, изменения будут окончательно приняты, если будет вызван метод [Post](#), или отменены, если будет вызван метод [Cancel](#) для этой записи (объекта **Document**).

Если **ПринятьИзменения** равно FALSE, изменения не попадают из встроенного редактора во временный буфер, где хранится текущая редактируемая запись. Если параметр **ПринятьИзменения** опущен, он считается равным TRUE.

Пример

```
proc ButCancelClick(S:String);  
    Cardfile1.EndEdit(FALSE);  
end;
```

Описание

НачатьРедактирование;
BeginEdit;

Назначение

Открывает в выделенной клетке встроенный (in-place) редактор, позволяющий изменить значение в ней. Редактирование возможно только в том случае, если в настройках картотеки разрешена ее модификация.

Пример

```
-- обработчик события "щелчок мышью в колонке картотеки"
func CardfileTTN_OnClick(Action :Integer; Column :CardfileColumn;
Document :Document):Logical;
  if Action>0:
    CardFile.BeginEdit;
    -- приступить к редактированию клетки
  fi;
  return Ложь;
end;
```

Описание

```
Обновить[(Только1строка:Логическое)];  
Update[(Only1row:Logical)];
```

Аргументы

Только1строка - логическое выражение, которое определяет, следует ли обновить только одну текущую строку или все окно картотеки.

Назначение

Иницирует перерисовку окна картотеки, если параметр равен FALSE, или только одной текущей строки картотеки, в противном случае. Если параметр опущен, то он по умолчанию принимается равным FALSE, то есть обновляется все окно целиком.

Данная процедура может быть полезна при выполнении длительных расчетов, влияющих на содержимое картотеки, и когда требуется визуально отображать ход процесса. Кроме того, обновление может потребоваться, если внешний вид некоторых вычисляемых полей зависит не только от их значений, но и внешних факторов. Дело в том, что картотека автоматически перерисовывается только при изменении значений в ее полях и не может реагировать на какие-либо другие события вне ее, даже если они по логике программы связаны с картотечкой.

Пример

```
proc OnEditChange(E1:Edit);  
  Update(TRUE);  
  -- при изменении в редакторе обновляем строку картотеки  
end;
```

Описание

```
СнятьВыделениеЗаписей([Записи : Запись\[\] ]);  
DeselectRecords([Records : Record\[\] ]);
```

Аргументы

Записи - необязательный параметр, массив записей, у которых требуется снять выделение.

Назначение

Снимает выделение массива записей, заданных в первом параметре, или всех записей, если параметр опущен. Записи могут выделены с помощью процедуры [ВыделитьЗаписи](#).

Пример

```
var локКартотека           :Cardfile;  
var локКолВоОбработанных  :Integer;  
var локОбработанныеПроцессы :Данные.Процесс[];  
  
proc СнятьВыделениеКартотеки;  
    локКартотека.DeselectRecords(локОбработанныеПроцессы);  
    локОбработанныеПроцессы = nil;  
end;
```

Описание

```
СохранитьВыделенное(Поле :Строка; ИмяФайла :Строка);  
SaveSelection(Field :String; FileName :String);
```

Аргументы

Поле - имя поля записи (причем, может быть указано любое поле), которое используется для идентификации картотеки;

ИмяФайла - имя текстового файла, в котором сохраняется множество выделенных записей в картотеке.

Назначение

Процедура сохраняет в текстовом файле, имя которого задано во втором параметре, множество записей, выделенных в картотеке, в формате "одна строка - одна запись".

Описание

Удалить;
Delete;

Назначение

Удаляет все выделенные записи. Если записи не выделены, удаляет текущую запись. Если нет ни выделенных, ни текущей записей, происходит ошибка.

Удаляемые записи не должны быть помеченными как удаленные. В противном случае также возникает ошибка. Иными словами, нельзя повторно удалить уже удаленную запись.

Пример

```
proc ButtonDelOnClick(Sender :Кнопка);  
  try  
    Cardfile1.Delete;  
  except  
  end;  
end;
```

Описание

```
УдалитьПодтаблицу(Номер :Целое);  
DeleteSubCard(Index :Integer);
```

Аргументы

Номер - номер удаляемой подтаблицы в картотеке.

Назначение

Удаляет существующую подтаблицу по ее номеру в картотеке. Номер удаляемой подтаблицы не может превышать [количества](#) имеющихся в картотеке подтаблиц.

Описание

```
УдалитьСтолбец(Номер :Целое);  
DeleteColumn(Position :Integer);
```

Аргументы

Номер - задает номер столбца, который следует удалить. Номер должен лежать в пределах от 1 до [количества столбцов](#).

Назначение

Процедура удаляет указанный столбец из картотеки.

Пример

```
-- код класса формы бланка  
-- ...  
-- удаляем первый столбец  
Cardfile.DeleteColumn(1);
```

Описание

ПередИзменением: Строка;
BeforeModify: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит перед началом изменения одной или нескольких выделенных записей. Если какое-либо действие выполняется сразу над группой записей, то событие **ПередИзменением** происходит лишь один раз – при этом перечень выделенных записей можно получить из обработчика, обратившись к полю [Выделенные](#).

Обработчик

```
func BeforeModify(Action :Cardfile.ModifyActions; Record :Record; Group :Record; var  
AskConfirm :Logical) :Logical;
```

Параметры:

Action - действие, которое производится над записями картотеки (удаление, восстановление, перемещение или копирование);

Record - ссылка на запись, над которой производится действие, если действие выполняется сразу над несколькими записями, параметр **Record** равен **nil**;

Group - целевая группа, в которую переносится запись или набор записей в случае перемещения или копирования;

AskConfirm - логический параметр, позволяет указать системе, следует ли запрашивать подтверждения у пользователя. Если данный параметр при выходе из обработчика равен TRUE, то пользователю выдается запрос, а в случае FALSE - нет. По умолчанию (при входе в обработчик) AskConfirm равно TRUE.

Если обработчик возвращает TRUE, то тем самым разрешается выполнение действия и будут сгенерированы все последующие события - в первую очередь [ПриИзменении](#) (это событие будет генерироваться для каждой записи из общего числа выделенных). Если обработчик **ПередИзменением** вернет FALSE, выполнение действия над записями картотеки прекращается.

При входе в обработчик Result равен FALSE (действия запрещены).

Пример

```
func Card_OnBeforeModify(Action :Cardfile.ModifyActions;  
Record :Record; Group :Record; var AskConfirm :Logical) :Logical;  
    Result = TRUE;  
    if Action = Cardfile.CopyRecord then  
        -- копирование запрещено  
        return FALSE;  
    elsif Action = Cardfile.DeleteRecord then  
        -- удаление требует подтверждения  
        AskConfirm = TRUE;  
    else  
        -- остальные действия выполнять без подтверждений  
        AskConfirm = FALSE;  
    end;  
end;
```

Описание

ПриВходе: Строка;
OnEnter: String;

Назначение

Данное событие происходит, когда картотека шаблона получает фокус ввода (то есть, становится текущим элементом управления на шаблоне). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB* или по щелчку мыши. Поле **ПриВходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnEnter**(TC :TemplateCardfile);
TC - объект класса **КартотекаШаблона**, с которым произошло событие.

Пример

```
proc КартотекаШаблона_ПриВходе(TC :TemplateCardfile);  
    TC.Font.Bold = TRUE;  
end;
```

Описание

ПриВыходе: Строка;
OnExit: String;

Назначение

Данное событие происходит, когда картотека шаблона теряет фокус ввода (то есть, перестает быть текущим элементом управления). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB* или по щелчку мыши. Поле **ПриВыходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnExit**(TC :TemplateCardfile);
TC - объект класса **КартотекаШаблона**, с которым произошло событие.

Пример

```
proc КартотекаШаблона_ПриВыходе(TC :TemplateCardfile);  
    TC.Font.Bold = FALSE;  
end;
```

Описание

ПриЗаписи: Строка;
OnPost: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при попытке сохранить измененную запись картотеки.

Обработчик

```
func OnPost(Rec :Record) :Logical;
```

Параметры:

Rec - редактируемая запись (документ).

Событие позволяет контролировать запись новых значений в поля документа, указанного с помощью параметра **Rec**. Если обработчик возвращает TRUE, документ сохраняется, иначе – нет.

Пример

```
func Картотека_ПриЗаписи(Record :Record) :Logical;  
  if Record.Имя = '' then  
    CardFile.CurrentColumn = CardFile.ColumnByField['Имя'];  
    Message('Не указано наименование счета.');
```

```
    Result = False;  
  elseif Record.Тип = '' then  
    CardFile.CurrentColumn = CardFile.ColumnByField['Тип'];  
    Message('Не указан тип счета.');
```

```
    Result = False;  
  else  
    Result = True;  
  end;  
end;
```

Описание

ПриИзменении: Строка;
OnModify: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит перед началом изменения конкретной записи. Это событие генерируется сразу после события **ПередИзменением**, причем если определен обработчик события **ПередИзменением**, то в нем должно быть разрешено проведение действия (функция-обработчик **ПередИзменением** должна вернуть TRUE).

Если какое-либо действие выполняется сразу над группой записей, то событие **ПриИзменении** происходит для каждой записи отдельно.

Обработчик

```
func OnModify(Action :Cardfile.ModifyActions; Record :Record; Group :Record) :Logical;
```

Параметры:

Action - действие, которое производится над записью картотеки (удаление, восстановление, перемещение или копирование);

Record - ссылка на запись, над которой производится действие;

Group - целевая группа, в которую переносится запись в случае перемещения или копирования. При копировании записи параметр **Record** содержит не исходную запись, а её копию, причем в этой копии еще не заполнено поле **GroupDoc** (значения всех остальных полей соответствуют исходной записи).

Если обработчик возвращает TRUE, система выполняет стандартную обработку выполняемого действия. Если обработчик возвращает FALSE, система не выполняет стандартную обработку, однако программист может закодировать непосредственно в теле обработчика все необходимые операции (в том числе, непредусмотренные стандартной обработкой).

В частности, обработчик может исправить часть полей записи (например, чтобы избежать конфликтов уникальных полей) и вернуть TRUE. Либо он может самостоятельно записать запись и вернуть FALSE. Если обработчик вернет FALSE, не записав запись, это будет означать, что запись не будет скопирована. При этом, если копировавшаяся запись является групповой, то не копируются и все её элементы.

При входе в обработчик Result равен FALSE (действия запрещены).

Пример

```
func Card_OnModify(Action :Cardfile.ModifyActions; TheRecord :Record; TheGroup :Record) :Logical;  
  if Action = Cardfile.UndeleteRecord then  
    -- восстановление запрещено  
    return FALSE;  
  elseif Action = Cardfile.CopyRecord then  
    -- изменяем значения некоторых полей  
    TheRecord.Name = "<пусто>";  
    -- соглашаемся разместить запись в группе,  
    -- куда её копирует пользователь  
    TheRecord.GroupDoc = TheGroup;  
    -- сохраняем запись  
    TheRecord.Post;  
    -- говорим системе, что дальнейшая  
    -- автоматическая обработка события не требуется  
    return FALSE;  
  end;  
  return TRUE;  
end;
```


Описание

ПриНажатии: Строка;
OnClick: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при одиночном или двойном щелчке мышью на записи, при щелчке мышью по значку записи у левого края окна картотеки, а также при нажатии клавиши *Enter*.

Если клетка картотеки имеет кнопку выбора и доступна только на чтение, то нажатие в ней кнопки выбора также генерирует событие **ПриНажатии**.

Обработчик

```
func OnClick(Action : Template.ClickTypes; Column :CardfileColumn; Rec :Record) :Logical;
```

Параметры:

Action - одна из предопределенных констант, описанных в типе [Шаблон.ТипыНажатия](#), которая определяет действие пользователя;

Column - текущий столбец картотеки;

Rec - текущий документ (запись).

Если функция возвращает TRUE, происходит стандартная обработка события, то есть открытие бланка-редактора, открытие встроенного (in-place) редактора, вход в группу или выбор текущего документа, в зависимости от настроек картотеки.

Если функция возвращает FALSE, стандартная обработка не выполняется.

Пример

```
-- обработчик события "ПриНажатии" в картотеке
func CardfileTN_OnClick(Action :Template.ClickTypes;
  Column :CardfileColumn; Document :Document):Logical;
-- по одиночному щелчку...
if Action = Template.SingleClick then
  -- приступить к редактированию клетки в самой картотеке
  CardFile.BeginEdit;
-- по нажатию кнопки у левого края записи...
elseif Action = Template.ButtonPressed then
  -- отредактировать её в бланке-редакторе
  Cardfile.Editors[1].ExecuteBlank(Document);
fi;
return Ложь;
end;
```

Описание

ПриОткрытииБланка: Строка;
OnOpenBlank: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции- обработчика события, которое происходит перед открытием бланка-редактора с текущей записью.

Обработчик

func **OnOpenBlank**(Action :Integer; Rec :Record) :Logical;

Параметры:

Action - определяет тип события:

- 0** - редактируется существующий документ;
- 1** - добавляется новый документ;
- 2** - добавляется копия документа (документ клонируется).

Rec - редактируемый документ (запись).

Если функция возвращает TRUE, происходит стандартная обработка события, то есть открытие бланка-редактора. Если функция возвращает FALSE, стандартная обработка не выполняется.

Пример

```
func Картотека1_ПриПопыткеСоздать(Action :Integer; Rec :Record) :Logical;  
  if Action=0:  
    return TRUE;  
  else  
    message('Для создания новых документов '+  
      'воспользуйтесь командами Добавить или Дублировать');  
  fi;  
  return FALSE;  
end;
```

Описание

ПриОтмене: Строка;
OnCancel: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при попытке отменить сделанные в картотечной записи изменения и вернуть ее в первоначальное состояние.

Обработчик

```
func OnCancel(Rec :Record) :Logical;
```

Параметры:

Rec - исходный документ (запись).

Событие позволяет контролировать "откат" к старому состоянию записи. Если обработчик возвращает TRUE, запись, заданная с помощью параметра **Rec**, возвращается в исходное состояние; если функция возвращает FALSE – ничего не происходит.

Пример

```
func ПриОтмене(Операция :Операции.Оплата) :Логическое;  
-- если в новых записях какие-то поля проставляются  
-- программой по умолчанию, то при "сбросе" такой записи  
-- нужно повторить инициализацию  
  if Операция.НовыйДокумент then  
    Операция.УстановитьЗначенияПоУмолчанию; -- прикладная процедура  
  end;  
  return TRUE;  
end;
```

Описание

ПриОформлении: Строка;
OnRearrange: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при изменении атрибутов столбца пользователем. Системой отслеживаются изменения ширины, положения, видимости и сортировки.

Обработчик

proc **OnRearrange**(Action : [Cardfile.RearrangeActions](#); Column :CardfileColumn);

Параметры:

Action - параметр, детализирующий, какой именно атрибут столбца был изменен и принимающий одно из значений перечислимого типа [RearrangeActions](#);

Column - указатель на объект **СтолбецКартотеки**, в котором произошло событие.

Внимание. Если параметр Action=Cardfile.OtherRearrangeActions, параметр Column = nil. В этом случае событие может вызываться, например, при изменении ширины панелей или включении/выключении видимости удаленных записей.

Пример

```
proc Card_OnRearrange(Action :Cardfile.RearrangeActions;  
    Column :CardfileColumn);  
    -- в случае сортировки по какому-либо столбцу  
    if Action = Cardfile.Sort then  
        -- вызываем прикладную функцию для сортировки  
        -- служебных массивов  
        SortAuxArrays(Column);  
    end;  
end;
```

Описание

ПриПеремещении: Строка;
OnMove: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при перемещении текстового курсора в картотеке от одной записи к другой.

Обработчик

```
proc OnMove(Rec :Record);
```

Параметры:

Rec - ссылка на текущую запись.

Событие позволяет управлять состоянием и значением полей записи картотеки в зависимости от контекста ввода.

В некоторых случаях, например, после установки пользовательского фильтра или при переходе из группы в группу, может возникнуть ситуация, когда ни один документ не выделен. Тогда параметр **Rec** будет равен nil. Поэтому в обработчике данного события прежде чем обращаться к какому-либо свойству документа всегда требуется проверять **Rec** на nil.

Пример

```
proc CardfileTTN_OnMove(D:Document);  
  if D <> nil then  
    hint(D.Номер);  
  end;  
end;
```

Описание

ПриРедактировании: Строка;
OnEdit: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит в момент начала редактирования записи в картотеке, минуя бланк-редактор. Иными словами, событие происходит после того, как пользователь вызвал команду редактирования или вставки записи (например, нажал кнопку *Ins*), но до того, как началось ее редактирование. Событие позволяет проинициализировать требуемые поля записи или вообще запретить ее создание.

Обработчик

```
func OnEdit(Action :Integer; Rec :Record): Logical;
```

Параметры:

Action - выполняемое действие:

- 0** - редактируется имеющаяся запись;
- 1** - добавляется новая запись;
- 2** - добавляется копия записи (запись клонируется);

Rec - добавляемая запись.

Если функция возвращает True, происходит стандартная обработка (переход в режим редактирования или вставка новой записи) и позиционирование на нее. Если функция возвращает False, редактирование не начинается (в случае попытки добавить новую запись, она не добавляется), то есть для новой записи автоматически делается [Cancel](#) (так же, как и при событии [OnOpenBlank](#)).

Необходимо отметить, что запись в момент наступления события еще не вставлена в Запрос картотеки (хотя и отображается в окне картотеки) – это происходит только после завершения редактирования, то есть после вызова метода [Post](#).

Пример

```
func CardfileOrders_OnEdit(Kind :Integer; D :Document):Logical;  
  if Kind = 1 then  
    СквознойНомер = СквознойНомер + 1  
    D.Номер = СквознойНомер;  
  end;  
  return true;  
end;
```

Описание

```
ПриРисованииСтроки : Строка;  
OnDrawRow : String;
```

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит перед изменением цвета иконки картотеки.

Обработчик

```
проц OnDrawRow(Record :Record; Selected :Logical; var Color :Integer; Image :Image);
```

Параметры:

Record - указатель на редактируемую запись (документ) объекта Картотека;
Selected - логический параметр. Если он равен True, запись картотеки (строка) выделена;
Color - цвет, который будет иметь иконка после срабатывания процедуры-обработчика;
Image - параметр не используется.

Пример

```
проц картотека_ПриРисованииСтроки(Record :Справочники.СтатусДвижения;  
Selected :Logical; var Color :Integer; Image :Image);  
    Color = Record.Цвет;  
end;
```

Описание

ПриСменеВыделения: Строка;
OnChangeSelected: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при изменении выделения нескольких документов.

Обработчик

```
proc OnChangeSelected(Count :Integer);
```

Параметры:

Count - количество выделенных документов.

Когда выделение снимается, **Count** равен нулю.

Количество выделенных документов и ссылки на сами документы можно получить соответственно с помощью свойств [КоличествоВыделенных](#) и [Выделенные](#).

Пример

```
proc CardfileOrders_OnChangeSelected(Count : Integer);  
  hint("Выделено:" + Str(Count));  
end;
```


Описание

ПриСменеГруппы: Строка;
OnChangeGroup: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при переходе из одной группы в другую.

Обработчик

```
func OnChangeGroup(Group :Record) :Logical;
```

Параметры:

Group - ссылку на запись той группы, в которую осуществляется переход.

Если обработчик возвращает True – выполняется стандартная обработка, если False – ничего не происходит. Подразумевается, что в последнем случае смену текущей группы, если это разрешено алгоритмом, сделает сам обработчик.

Внимание! В обработчике **OnChangeGroup** нельзя менять текущие группы через свойство картотеки **Cardfile.CurrentGroup**, так как при этом обработчик вызовется снова и произойдет заикливание. Для смены группы программным образом следует использовать свойство **CurrentGroup** из запроса картотеки (см. пример).

Пример

```
func Card_OnChangeGroup(Group :Record) :Logical;  
  -- Действия до смены группы  
  --...  
  Query.CurrentGroup = Group;  
  -- Действия после смены группы  
  --...  
  return False;  
end;
```

Описание

ПриСозданииЗаписи: Строка;
OnCreateRecord: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при создании записи в картотеке.

Данное событие возникает только при добавлении новой записи в картотеку, но не при дублировании старой. Оно предназначено для управления созданием новой записи и позволяет, например, автоматизировать добавление новой записи в гетерогенных картотеках. Там, как известно, при попытке добавить запись открывается стандартный диалог выбора класса добавляемой записи, что не всегда удобно.

Обработчик

```
func OnCreateRecord(var Rec :Record) :Logical;
```

Параметры:

Rec - выходной параметр для создаваемой записи.

Обработчик должен вернуть FALSE в случае отказа от дальнейшей стандартной обработки и TRUE, если ее нужно продолжить. При входе в обработчик параметр **Rec** равен *nil*. [ПриОткрытииБланка](#) или [ПриРедактировании](#), в зависимости от способа редактирования записи, и т.д.).

Таким образом, сначала происходит событие **ПриСозданииЗаписи** и только потом, в случае если последнее вернуло TRUE, произойдет одно из двух событий: **ПриРедактировании** – в случае inplace-редактирования, или **ПриОткрытииБланка** – при редактировании с помощью бланка.

Пример

```
-- На шаблоне картотеки размещен выпадающий список
-- ComboBoxClassName с именами классов записей;
-- При добавлении новой записи в картотеку, программа
-- автоматически выбирает класс, выделенный в списке.
func Cardfile_OnCreateRecord (var Rec :Record) :Logical;
  try
    Rec = FindClass(ComboBoxClassName.Text).Create;
  except
  end;
  return TRUE;
end;
```

Описание

ВставитьПодтаблицу(Номер :Целое) : [ПодтаблицаКартотеки](#);
InsertSubCard(Index :Integer) : [SubCardfile](#);

Аргументы

Номер - определяет номер подтаблицы, вставляемой в картотеку.

Назначение

Вставляет новую подтаблицу в картотеку в заданное по номеру место, отодвигая, в случае необходимости, имеющиеся подтаблицы вниз. Номер должен находиться в пределах от 1 до [количества](#) уже имеющихся в картотеке подтаблиц плюс 1. Например, если в картотеке уже есть 2 подтаблицы, то новую можно вставить на позиции от 1 до 3, причем номер 3 означает, подтаблица будет помещена в самый низ, после всех имеющихся подтаблиц картотеки.

Описание

```
ВставитьСтолбец(Номер :Целое) :СтолбецКартотеки;  
InsertColumn(Position :Integer) :CardfileColumn;
```

Аргументы

Номер - задает позицию, в которую будет вставлен новый столбец. Номер должен лежать в пределах от 1 до [количества столбцов](#).

Назначение

Функция вставляет новый столбец в картотеку. Позиция нового столбца задается аргументом **Номер**.

Пример

```
-- код класса формы бланка  
  
proc PlusColumn1(Fieldname :String);  
var Column: CardfileColumn;  
  -- создаем столбец, который должен располагаться первым  
  Column = Cardfile.InsertColumn(1);  
  -- настраиваем столбец  
  Column.FieldName = Fieldname;  
  Column.Width = 200;  
  -- ...  
end;
```

Класс Объект : Картотека

Функция ДобавитьПодтаблицу / AddSubCard

Описание

ДобавитьПодтаблицу : ПодтаблицаКартотеки;

AddSubCard : SubCardfile;

Назначение

Функция добавляет новую подтаблицу в картотеку. Подтаблица вставляется после последней подтаблицы в картотеке. Для вставки подтаблицы между уже имеющимися подтаблицами используется метод ВставитьПодтаблицу.

Описание

ДобавитьСтолбец : [СтолбецКартотеки](#);
AddColumn : [CardfileColumn](#);

Назначение

Функция добавляет новый столбец в картотеку. Столбец становится последним в массиве столбцов.

Пример

```
-- код класса формы бланка

proc PlusColumn(FieldName :String);
var Column: CardfileColumn;
  -- создаем столбец и получаем ссылку на новый объект
  Column = Cardfile.AddColumn;
  -- настраиваем столбец
  Column.FieldName = FieldName;
  Column.Width = 200;
  -- ...
end;
```

Класс *ПодтаблицаКартотеки* / *SubCardfile*, производный от класса [Объект](#), используется для работы с визуальным представлением [многозначных полей](#) (массивов), в том числе и [структурных](#), определенных с помощью MTL-описания в прикладных классах записей, отображаемых в картотеках Студии.

Внимание! Создать объект класса *ПодтаблицаКартотеки* напрямую нельзя. Подтаблицы в картотеках создаются самой системой на основе файлов с описанием их структуры и внешнего представления (BRO, TPL, COD файлы) и программно доступны через соответствующие свойства объекта *Картотека*.

Непосредственно в классе *ПодтаблицаКартотеки* определены следующие свойства:

-  [Поле Надпись / Caption](#)
-  [Поле ИмяПодтаблицы / SubtableName](#)
-  [Поле ФильтрПодтаблицы / SubtableFilter](#)
-  [Подтаблица / Subtable](#)
-  [Поле Текущая / Current](#)
-  [Поле КоличествоВыделенных / SelectedCount](#)
-  [Поле Выделенные / Selected](#)
-  [Поле ТекущийСтолбец / CurrentColumn](#)
-  [Поле ПозТекущегоСтолбца / CurrentColumnPos](#)
-  [Поле ФиксированныхСтолбцов / FixedColumns](#)
-  [Поле КоличествоСтолбцов / ColumnsCount](#)
-  [Поле Столбец / Column](#)
-  [Поле СтолбецПоПолю / ColumnByField](#)
-  [Функция ДобавитьСтолбец / AddColumn](#)
-  [Функция ВставитьСтолбец / InsertColumn](#)
-  [Процедура УдалитьСтолбец / DeleteColumn](#)
-  [Процедура НачатьРедактирование / BeginEdit](#)
-  [Процедура ЗавершитьРедактирование / EndEdit](#)
-  [Процедура Обновить / Update](#)
-  [Поле ПоказыватьИтоги / ShowTotal](#)
-  [Поле МожноДобавлять / CanInsert](#)
-  [Поле МожноУдалять / CanDelete](#)
-  [Поле МожноПеремещать / CanMove](#)
-  [Поле МожноРедактировать / CanEdit](#)

-  [Событие ПриНажатии / OnClick](#)
-  [Событие ПриВставке / OnInsert](#)
-  [Событие ПриУдалении / OnDelete](#)
-  [Событие ПриСменеВыделения / OnChangeSelected](#)
-  [Событие ПриОформлении / OnRearrange](#)
-  [Событие ПриСменеПозиции / OnChangePosition](#)

Подтаблица / Subtable

Описание

Подтаблица : [Подтаблица](#);

Subtable : [Subtable](#);

Назначение

Поле содержит ссылку на объект класса [Подтаблица](#), связанного с данной подтаблицей картотеки. Имя этой подтаблицы можно узнать с помощью свойства [ИмяПодтаблицы](#). Поле доступно только на чтение.

Пример

```
Cardfile.SubCard[1].SubtableName = "Детали";
Cardfile.SubCard[1].SubtableFilter = "Код > 100 AND Код < 200";
-- выводим отладочную информацию о количестве записей
-- в подтаблице (с учетом наложения фильтра)
trace(Cardfile.SubCard[1].Subtable.Count);
```


Описание

Выделенные[Индекс: Целое] : [Структура](#);
Selected[Index: Integer] : [Structure](#);

Аргументы

Индекс - номер структуры. Номер меняется от 1 до числа выделенных структур, которое определяется с помощью свойства [КоличествоВыделенных](#).

Назначение

Поле позволяет по номеру получить ссылку на конкретную структуру из числа выделенных в подтаблице.

Поле доступно только на чтение.

Пример

```
-- цикл по выделенным строкам подтаблицы
for i=1..Cardfile.Subcard[1].SelectedCount do
  -- обработка
  CountUp(Cardfile.Subcard[1].Selected[i].Value);
end;
```

Описание

ИмяПодтаблицы : Строка;
SubtableName : String;

Назначение

Это поле позволяет установить имя подтаблицы (из класса записи, связанного с данной картотекой), которая должна выводиться в окне картотеки (в области для подтаблиц).

В визуальном редакторе шаблонов данное свойство вводится с помощью поля **Подтаблица** в диалоге ["Свойства подтаблицы"](#).

Пример

```
Cardfile.SubCard[1].SubtableName = "Детали";
```

Описание

КоличествоВыделенных : Целое;
SelectedCount : Integer;

Назначение

Поле позволяет определить количество выделенных в данный момент структур. Поле доступно только на чтение.

Пример

```
-- цикл по выделенным строкам подтаблицы
for i=1..Cardfile.Subcard[1].SelectedCount do
  -- обработка
end;
```

Описание

КоличествоСтолбцов : Целое;
ColumnsCount : Integer;

Назначение

Данное свойство содержит количество столбцов в подтаблице. Поле доступно только на чтение.

Для программного добавления/удаления столбцов необходимо пользоваться методами [ДобавитьСтолбец](#), [ВставитьСтолбец](#), [УдалитьСтолбец](#).

Пример

```
-- цикл по всем столбцам подтаблицы
for i=1..Cardfile.SubCard[1].ColumnsCount do
  if Шаблон.ПолеДаты = CardFile.SubCard[1].Column[i].FieldType then
    -- если дата
    CardFile.SubCard[1].Column[i].Format = "dd mmmm yyyy";
  end;
end;
```

Описание

МожноДобавлять :Логическое;
CanInsert :Logical;

Назначение

Поле определяет, можно ли интерактивно добавлять строки в подтаблицу.

В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге "[Свойства подтаблицы](#)" (на странице "Правка").

Пример

```
Cardfile.SubCard[1].CanInsert = CheckBox1.State;
```

Описание

МожноПеремещать :Логическое;

CanMove :Logical;

Назначение

Поле определяет, можно ли интерактивно перемещать строки в подтаблице.

В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге "[Свойства подтаблицы](#)" (на странице "Правка").

Пример

```
Cardfile.SubCard[1].CanMove = CheckBox3.State;
```

Описание

МожноРедактировать :Логическое;
CanEdit :Logical;

Назначение

Поле определяет, можно ли интерактивно редактировать значения в клетках подтаблицы.

В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге ["Свойства подтаблицы"](#) (на странице "Правка").

Пример

```
Cardfile.SubCard[1].CanEdit = CheckBox4.State;
```

Описание

МожноУдалять :Логическое;
CanDelete :Logical;

Назначение

Поле определяет, можно ли интерактивно удалять строки в подтаблице.

В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге "[Свойства подтаблицы](#)" (на странице "Правка").

Пример

```
Cardfile.SubCard[1].CanDelete = CheckBox2.State;
```


Поле Надпись / Caption

Описание

Надпись :Строка;

Caption :String;

Назначение

Данное поле позволяет узнать и изменить заголовок подтаблицы.

Свойство используется при закладочном расположении подтаблиц, по умолчанию совпадает с именем поля.

Пример

```
Cardfile.SubCard[1].Caption = "Детали";
```

Описание

ПозТекущегоСтолбца : Целое;
CurrentColumnPos : Integer;

Назначение

Данное свойство позволяет узнать номер текущего (видимого) столбца подтаблицы картотеки, а также сделать текущим другой видимый столбец. Текущим (выделенным) считается столбец, в котором находится курсор. Свойство доступно на чтение и запись.

Внимание! Свойство **ПозТекущегоСтолбца**=CurrentColumn.Index, только в том случае, когда видимыми являются все столбцы подтаблицы картотеки. В общем случае номер видимого столбца ПозТекущегоСтолбца не совпадает с порядковым номером столбца картотеки CurrentColumn.Index.

Валидность диапазона при установке свойства не проверяется, поэтому, чтобы встать на последний столбец можно написать CurrentColumnPos = 999; или задать общее [количества столбцов](#) (см. пример ниже).

Пример

```
-- выделяем клетку в самом последнем столбце  
Cardfile1.SubCard[1].CurrentColumnPos = Cardfile1.SubCard[1].ColumnsCount;
```

Описание

ПоказыватьИтоги :Логическое;
ShowTotal :Logical;

Назначение

Свойство разрешает или запрещает показывать итоговую строку в подтаблице картотеки. Поле доступно на чтение и запись.

Пример

```
var SC1 :SubCardfile;  
...  
SC1.ShowTotal = True;
```

Описание

Столбец[Индекс: Целое] : [СтолбецПодтаблицы](#)

Column[Index: Integer] : [SubCardfileColumn](#)

Аргументы

Индекс - задает позицию, в которую будет вставлен новый столбец подтаблицы. Индекс может изменяться в пределах от 1 до [количества столбцов](#).

Назначение

Данное поле позволяет получить доступ к конкретному столбцу подтаблицы картотеки по его номеру. Поле доступно только на чтение.

Пример

```
-- цикл по всем столбцам подтаблицы
for i=1..Cardfile.SubCard[1].ColumnsCount do
  if Шаблон.ПолеДаты = CardFile.SubCard[1].Column[i].FieldType then
    -- если дата
    CardFile.SubCard[1].Column[i].Format = "dd mmmm yyyy";
  end;
end;
```

Описание

СтолбецПоПолю[Имя: Строка] : [СтолбецПодтаблицы](#)
ColumnByField[Name: String] : [SubCardfileColumn](#)

Аргументы

Имя - имя столбца подтаблицы картотеки.

Назначение

Данное поле позволяет получить доступ к конкретному столбцу подтаблицы картотеки по имени поля, содержащегося в нем. Поле доступно только на чтение.

Пример

```
proc BoldByName(S:SubCardfile);  
var cc: SubCardFileColumn;  
    cc = S.ColumnByField["Сумма"];  
    cc.Font.Bold = TRUE;  
end;
```

Поле Текущая / Current

Описание

Текущая : [Структура](#);

Current : [Structure](#);

Назначение

Данное поле содержит указатель на текущую [структуру](#) в подтаблице. Изменяя значение поля можно менять текущую строку в подтаблице.

Следует отметить, что текущая строка визуально отличается от остальных строк подтаблицы – либо отдельная клетка строки, либо вся строка (в зависимости от типа выделения) отображается контрастным цветом.

Пример

```
-- выводим название товара из текущей позиции в строку состояния  
Hint(Cardfile.Subcard[1].Current.Товар);
```

Описание

ТекущийСтолбец : [СтолбецПодтаблицы](#)

CurrentColumn : [SubCardfileColumn](#)

Назначение

Поле возвращает ссылку на текущий столбец подтаблицы, содержащий выделенную клетку. Текущим (выделенным) считается столбец, в котором находится курсор. Свойство доступно на чтение и запись.

Внимание! При попытке выделить невидимый столбец или столбец, отсутствующий в подтаблице, текущая позиция *не* меняется.

Пример

```
Cardfile.SubCard[1].CurrentColumn = CardFile.SubCard[1].ColumnByField['Тип'];
```

Описание

ФиксированныхСтолбцов : Целое;
FixedColumns : Integer;

Назначение

Данное свойство позволяет узнать и установить количество фиксированных столбцов в подтаблице картотеки. Фиксированные столбцы остаются у левого края картотеки при ее прокрутке по горизонтали.

Пример

```
proc ButtonClick(B:Button);  
    Картотека.Подтаблица[1].ФиксированныхСтолбцов = 2;  
end;
```


Описание

ФильтрПодтаблицы : Строка;
SubtableFilter : String;

Назначение

Поле **ФильтрПодтаблицы** позволяет установить на отображаемые записи подтаблицы произвольный фильтр – строковое выражение, отвечающее общим правилам написания фильтров (см. раздел [Установка фильтра](#)). В подтаблице будут отображаться только те записи, для которых выражение фильтра принимает истинное значение.

В визуальном редакторе шаблонов данное свойство вводится с помощью поля **Фильтр** в диалоге ["Свойства подтаблицы"](#).

Пример

```
Cardfile.SubCard[1].SubtableFilter = "Код > 100 AND Код < 200";
```

Описание

```
ЗавершитьРедактирование[ (ПринятьИзменения :Логическое) ] ;  
EndEdit[ (AcceptChanges: Logical) ] ;
```

Аргументы

ПринятьИзменения - логический параметр, определяет, следует ли принять или отклонить сделанные изменения. Если параметр **ПринятьИзменения** опущен, он считается равным TRUE.

Назначение

Закрывает встроенный (in-place) редактор, открытый ранее с помощью [BeginEdit](#) или пользователем. Если **ПринятьИзменения** равно TRUE, изменения (если они были) передаются в обработчик [OnVerify](#) (если он установлен) и затем сохраняются во временном представлении текущей записи. Впоследствии, изменения будут окончательно приняты, если будет вызван метод [Post](#), или отменены, если будет вызван метод или если пользователь выполнит соответствующую команду (**Записать**). Изменения не проходят, если будет вызван метод [Cancel](#) для записи, содержащей подтаблицу, или пользователь выполнит соответствующую команду (**Отменить**).

Если **ПринятьИзменения** равно FALSE, изменения не попадают из встроенного редактора во временный буфер, где хранится текущая редактируемая запись.

Пример

```
proc ButtonOKClick(B:Button);  
    Cardfile.SubCard[1].EndEdit;  
end;
```

Описание

НачатьРедактирование;
BeginEdit;

Назначение

Открывает в выделенной клетке встроенный (in-place) редактор, позволяющий изменить значение в ней. При этом как подтаблица, так и вся запись, содержащая данную подтаблицу, переводится в состояние редактирования.

Редактирование возможно только в том случае, если в настройках подтаблицы разрешена ее модификация.

Пример

```
proc B1Click(B:Button);  
    Cardfile.SubCard[1].BeginEdit;  
end;
```

Описание

```
Обновить[(Только1строка:Логическое)];  
Update[(Only1row:Logical)];
```

Аргументы

Только1строка - логическое выражение, которое определяет, следует ли обновить только одну текущую строку или все окно подтаблицы.

Назначение

Иницирует перерисовку окна картотеки, если параметр равен FALSE, или только одной текущей строки картотеки, если параметр равен TRUE. Если параметр опущен, то он по умолчанию принимается равным FALSE, то есть обновляется вся подтаблица.

Данная процедура может быть полезна при выполнении длительных расчетов, влияющих на содержимое картотеки и когда требуется визуально отображать ход процесса. Кроме того, обновление может потребоваться, если внешний вид некоторых вычисляемых полей зависит не только от их значений, но и внешних факторов. Дело в том, что картотека автоматически перерисовывается только при изменении значений в ее полях и не может реагировать на какие-либо другие события вне ее, даже если они по логике программы связаны с картотеккой.

Пример

```
proc OnEditChange(E1:Edit);  
    Cardfile.SubCard[1].Update(TRUE);  
    -- обновляем строку подтаблицы картотеки  
end;
```

Описание

```
УдалитьСтолбец(Номер :Целое);  
DeleteColumn(Position :Integer);
```

Аргументы

Номер - задает номер столбца, который следует удалить. Номер должен лежать в пределах от 1 до [количества столбцов](#).

Назначение

Процедура удаляет указанный столбец из подтаблицы картотеки.

Пример

```
-- код класса формы бланка  
-- ...  
-- удаляем первый столбец из подтаблицы  
Cardfile.SubCard[1].DeleteColumn(1);
```

Описание

ПриВставке : Строка;
OnInsert: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при вставке новой структуры в подтаблицу картотеки, минуя бланк-редактор. Событие происходит после того, как пользователь вызвал команду вставки структуры (например, нажал кнопку *Ins*), но до того, как началось ее редактирование. Событие позволяет проинициализировать требуемые поля структуры или вообще запретить ее создание.

Обработчик

```
func OnInsert(Sender :SubCardfile; Index :Integer ) :Logical;
```

Параметры:

Sender - указатель на объект класса **ПодтаблицаКартотеки**, в котором произошло событие. С помощью этого параметра можно узнать, какие структуры уже имеются в подтаблице и проанализировать их содержимое;

Index - номер добавляемой структуры ([Структура](#)).

Если функция возвращает TRUE, происходит стандартная вставка структуры и позиционирование на нее. Если функция возвращает FALSE, вставка не выполняется.

Пример

```
func CardfileOrder_OnInsert(Tab :SubCardfile;  
    Index :Integer) :Logical;  
    var S :Structure;  
    S = Tab.Subtable.Items[Index];  
    -- заполняем поле структуры значением по умолчанию  
    (S As Документы.ПриходРасход.Позиции).Количество = 1;  
    return true;  
end;
```

Описание

ПриНажатии: Строка;
OnClick: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при одиночном или двойном щелчке мышью на клетке, а также при нажатии клавиши *Enter*.

Если клетка подтаблицы картотеки имеет кнопку выбора и эта кнопка была нажата, то система также генерирует событие **ПриНажатии**, причем это верно в том числе и для клеток, доступных только на чтение.

Обработчик

func **OnClick**(Action : [Template.ClickTypes](#); Column : [SubCardfileColumn](#); Struct: [Structure](#)) :Logical;

Событие возникает при щелчке кнопки мыши на любой клетке подтаблицы картотеки, при нажатии клавиши *Enter* или кнопки обзора (если она есть).

Параметры:

- Action** - одна из предопределенных констант, описанных в типе [Шаблон.ТипыНажатия](#), которая определяет действие пользователя;
- Column** - текущий столбец подтаблицы ([СтолбецПодтаблицы](#));
- Struct** - текущая структура (строка) подтаблицы ([Структура](#)).

Если функция возвращает TRUE, происходит стандартная обработка события, то есть открытие бланка-редактора, открытие встроенного (in-place) редактора, вход в группу или выбор текущего документа, в зависимости от настроек картотеки. Если функция возвращает FALSE, стандартная обработка не выполняется.

Если событие не обрабатывается, то это эквивалентно возврату значения TRUE.

Пример

```
-- обработчик события "щелчок мышью в колонке подтаблицы"
func CardfileSubTable_OnClick(Action :Template.ClickTypes;
  Column:SubCardfileColumn; Struct:Structure):Logical;
  if Action = Template.SingleClick then
    CardFile.SubCard[1].BeginEdit;
    -- приступить к редактированию клетки
  end;
  return Ложь;
end;
```

Описание

ПриОформлении: Строка;
OnRearrange: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при изменении атрибутов столбца пользователем. Системой отслеживаются изменения ширины, положения, видимости и сортировки.

Обработчик

proc **OnRearrange**(Action : [Cardfile.RearrangeActions](#); Column : [SubCardfileColumn](#));

Параметры:

Action - параметр, определяющий, какой именно атрибут столбца был изменен и принимающий одно из значений перечислимого типа [RearrangeActions](#);

Column - указатель на объект класса **СтолбецПодтаблицы**, в котором произошло событие.

Внимание. Если параметр Action=Cardfile.OtherRearrangeActions, параметр Column = nil. В этом случае событие может вызываться, например, при изменении ширины панелей или включении/выключении видимости удаленных записей.

Пример

```
proc SubCard_OnRearrange(Column :SubCardfileColumn; Action :Cardfile.RearrangeActions);  
  -- в случае сортировки по какому-либо столбцу  
  if Action = Cardfile.Sort then  
    -- вызываем прикладную функцию для сортировки  
    -- служебных массивов  
    SortAuxArrays(Column);  
  end;  
end;
```


Описание

ПриСменеВыделения: Строка;
OnChangeSelected: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при изменении выделения нескольких структур в подтаблице.

Обработчик

```
proc OnChangeSelected(Count :Integer);
```

Параметры:

Count - количество выделенных структур. Когда выделение снимается, **Count** равен нулю.

Количество выделенных структур и ссылки на сами структуры можно получить соответственно с помощью свойств [КоличествоВыделенных](#) и [Выделенные](#).

Пример

```
proc CardfileOrder_OnChangeSelected(Count : Integer);  
  hint("Выделено:" + Str(Count));  
end;
```

Описание

ПриСменеПозиции :Строка;
OnChangePosition :String;

Назначение

Данное событие позволяет управлять процессом перемещения строк в подтаблице.

Обработчик

```
func OnChangePosition(Sender :SubCardfile; Index :Integer; MoveUp :Logical) :Logical;
```

Параметры:

Sender - указатель на объект класса **ПодтаблицаКартотеки**, в котором произошло событие;

Index - номер перемещаемой строки. Нумерация строк начинается с 1;

MoveUp - логический параметр, равный значению TRUE, если строка перемещается вверх, и FALSE - в противном случае.

Возвращаемый функцией результат разрешает (TRUE) либо запрещает (FALSE) перемещение. В случае, если обработчик вернул FALSE, перемещение можно осуществить, запрограммировав его вручную (с учетом специфики выполняемой задачи).

Пример

```
func ПодТабл_ПриСменеПозиции(Sender :SubCarfile;  
  N :Integer; Up :Logical): Logical;  
  if Sender = ТаблицаТоварныхПозиций then  
    return TRUE;  
    -- разрешаем перемещение кадра  
  else  
    return FALSE;  
    -- запрещаем перемещение кадра  
  end;  
end;
```

Описание

ПриУдалении : Строка;
OnDelete : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит перед удалением структуры (строки) подтаблицы.

Обработчик

```
func OnDelete(Sender : SubCardfile; Index :Integer) :Logical;
```

Параметры:

Sender - указатель на объект класса **ПодтаблицаКартотеки**, в котором произошло событие;

Index - номер элемента подтаблицы, то есть номер структуры, которая удаляется.

Если функция возвращает TRUE, происходит стандартная обработка события, то есть структура удаляется. Если функция возвращает FALSE, стандартная обработка не выполняется.

Пример

```
func CardfileOrder_OnDelete(Tab :SubCardfile; Index :Integer) :Logical;  
var S :Structure;  
var Товар :Документы.ПриходРасход.Позиции;  
S = Tab.Subtable.Items[Index];  
Товар = S As Документы.ПриходРасход.Позиции;  
if Товар.Количество = 0 then  
    return TRUE;  
end;  
return FALSE;  
end;
```

Описание

ВставитьСтолбец(Номер :Целое) : [СтолбецПодтаблицы](#)
InsertColumn(Position :Integer) : [SubCardfileColumn](#)

Аргументы

Номер - задает позицию, в которую будет вставлен новый столбец. Номер должен лежать в пределах от 1 до [количества столбцов](#).

Назначение

Функция вставляет новый столбец в подтаблицу картотеки. Позиция нового столбца задается аргументом **Номер**.

Пример

```
-- код класса формы бланка

proc PlusColumn1(Fieldname :String);
var Column: SubCardfileColumn;
  -- вставляем новый столбец в первую подтаблицу
  -- новый столбец должен располагаться первым
  Column = Cardfile.SubCard[1].InsertColumn(1);
  -- настраиваем столбец
  Column.FieldName = Fieldname;
  Column.Width = 200;
  -- ...
end;
```

Описание

ДобавитьСтолбец : [СтолбецПодтаблицы](#)

AddColumn : [SubCardfileColumn](#)

Назначение

Функция добавляет новый столбец в подтаблицу картотеки. Столбец становится последним в массиве столбцов.

Пример

```
-- код класса формы бланка












proc PlusColumn(FieldName :String);
var Column: SubCardfileColumn;
  -- добавляем столбец в первую подтаблицу
  Column = Cardfile.SubCard[1].AddColumn;
  -- настраиваем столбец
  Column.FieldName = FieldName;
  Column.Width = 200;
  -- ...
end;
```

Класс *СписокЗаписей* (*RecordList*), производный от класса [Объект](#), является вспомогательным при работе с картотеками. В частности, списки записей применяются для установки фильтра на группы записей в иерархической картотеке.

Внимание! Создать объект класса *СписокЗаписей* напрямую нельзя. Такие объекты создаются только внутри системы и используются как свойства других объектов (класса [Картотека / CardFile](#)).

Программа на ТБ.Скрипт может получить доступ к объекту *СписокЗаписей*, являющемуся свойством картотеки, и использовать его методы и свойства.

Непосредственно в классе *СписокЗаписей* определены следующие свойства и методы:

-  [Количество / Count](#)
-  [Запись / Record](#)
-  [Записи / Records](#)
-  [Корень / Root](#)
-  [Индекс / IndexOf](#)
-  [Добавить / Add](#)
-  [Вставить / Insert](#)
-  [Удалить / Delete](#)
-  [Очистить / Clear](#)
-  [СписокГруппы / GroupList](#)
-  [ФильтрГруппы / GroupFilter](#)

Описание

Записи: Запись [];
Records : Record [];

Назначение

С помощью данного поля можно прочитать или присвоить сразу весь массив записей. Заполнение списка записей таким способом значительно быстрее, чем поэлементная вставка.

Пример

```
Cardfile1: TemplateCardfile;  
-- установить в картотеке шаблона фильтр по выделенным  
-- записям в картотеке-источнике Source  
proc SetFilter(Source: Cardfile);  
var RL: RecordList;  
var i: Integer;  
    -- заполняем список выделенными записями  
    -- цикл от 1 до общего числа выделенных записей  
    for i=1..Source.SelectedCount do  
        RL.Add(Source.Selected[i])  
    end;  
    Cardfile1.GroupFilter = RL.Records;  
end;
```

Описание

Запись[] : Запись;
Record[] : Record;

Назначение

С помощью данного многозначного поля можно по целочисленному индексу записи в списке получить доступ к ней.

Поле доступно не только на чтение, но и на запись, то есть позволяет заменять элемент списка на новую запись. Индекс должен лежать в пределах от 1 до [Count](#).

Пример

```
-- функция заменяет запись в списке на другую
func ListReplace(RL: RecordList; D1: Record; D2: Record): Logical;
var i: Integer;
    for i=1..RL.Count do -- цикл от 1 до общего числа записей в списке
        if RL.Record[i] = D1 then
            RL.Record[i] = D2; -- замена
            return TRUE;
        end;
    end;
    return FALSE;
end;
```


Описание

Количество : Целое;
Count : Integer;

Назначение

Поле хранит количество записей в списке. Поле доступно только на чтение.

Пример

```
-- функция возвращает TRUE, если запись есть в списке
-- вместо такой функции можно воспользоваться методом
-- IndexOf
func IsInList(RL: RecordList; D: Record): Logical;
var i: Integer;
  for i=1..RL.Count do
    -- цикл от 1 до общего числа записей в списке
    if RL.Records[i] = D then
      return TRUE;
    end;
  end;
  return FALSE;
end;
```

Описание

Корень :Запись;
Root :Record;

Назначение

С помощью данного поля можно узнать и изменить запись, определяющую корневую группу для иерархических картотек (объектов классов **Картотека**, **КартотекаШаблона**, **ДеревоШаблона**). Любой картотечный объект отображает свои элементы, отобранные с учетом фильтра, относительно корневого элемента. Выйти из корневой группы на уровень выше нельзя.

Присваиваемая корневая запись должны быть группой и лежать в иерархии не ниже всех элементов, отобранных текущими фильтрами. В противном случае попытка установить некорректный корень приведет к ошибке.

Пример

```
Cardfile1: TemplateCardfile;  
-- установить в картотеке шаблона фильтр по выделенным  
-- записям в картотеке-источнике Source  
proc SetFilter(Source: Cardfile);  
var RL: RecordList;  
var i: Integer;  
    -- заполняем список выделенными записями  
    -- цикл от 1 до общего числа выделенных записей  
for i=1..Source.SelectedCount do  
    RL.Add(Source.Selected[i])  
end;  
Cardfile1.GroupFilter = RL.Records;  
-- после установки группового фильтра система в  
-- качестве корня автоматически выбирает группу,  
-- расположенную выше всех элементов, отобранных по  
-- фильтру, которая может и не быть вершиной иерархии.  
-- Устанавливаем в качестве корня вершину иерархии  
Cardfile1.GroupFilter.Root = nil;  
end;
```

Описание

Вставить(Позиция: Целое; Зап: Запись);
Insert(Position: Integer; Rec: Record);

Аргументы

Позиция - номер позиции (индекс) в списке, куда требуется вставить запись;
Зап - указатель на добавляемую запись.

Назначение

Процедура вставляет в список запись, заданную по указателю, в указанную позицию. Значение позиции должно лежать в пределах от 1 до [Count](#).

Пример

```
-- процедура добавляет запись в список, если там такой еще нет
-- новая запись добавляется в начало списка
proc AddGroupToFilter(RL: RecordList; D: Record);
var i: Integer;
    i = RL.IndexOf(D);
    if i = -1 then
        RL.Insert(1,D);
    end;
end;
```

Описание

```
Добавить(Зап: Запись);  
Add(Rec: Record);
```

Аргументы

Зап - указатель на добавляемую запись.

Назначение

Процедура добавляет запись, заданную по указателю, в конец списка.

Пример

```
-- процедура добавляет запись в список, если там такой еще нет  
proc AddGroupToFilter(RL: RecordList; D: Record);  
var i: Integer;  
    i = RL.IndexOf(D);  
    if i = -1 then  
        RL.Add(D);  
    end;  
end;
```

Описание

Очистить;
Clear;

Назначение

Процедура очищает список.

Пример

```
-- процедура очищает список, если в нем нет указанной записи
proc Exclude(RL: RecordList; D: Record);
var i: Integer;
    i = RL.IndexOf(D);
    if i = -1 then
        RL.Clear;
    end;
end;
```

Процедура Удалить / Delete

Описание

Удалить (Позиция: Целое);
Delete(Position: Integer);

Аргументы

Позиция - индекс записи, которую следует удалить.

Назначение

Процедура удаляет запись из списка по ее индексу, который должен лежать в пределах от 1 до Count.

Пример

```
-- процедура удаляет запись из списка, если такая есть
proc Exclude(RL: RecordList; D: Record);
var i: Integer;
    i = RL.IndexOf(D);
    if i <> -1 then
        RL.Delete(i);
    end;
end;
```

Описание

Индекс(Зап: Запись): Целое;
IndexOf(Rec: Record): Integer;

Аргументы

Зап - указатель на искомую запись.

Назначение

Функция возвращает индекс записи (от 1 до [Count](#)), если запись есть в списке, или -1, если такой записи нет в списке.

Пример

```
-- процедура добавляет запись в список, если там такой еще нет
proc AddGroupToFilter(RL: RecordList; D: Record);
var i: Integer;
    i = RL.IndexOf(D);
    if i = -1 then
        RL.Add(D);
    end;
end;
```

Описание

```
СписокГруппы(Группа :Запись) :Запись[];  
GroupList(TheGroup :Record) :Record[];
```

Аргументы

Группа - ссылка на группу, для которой необходимо получить список входящих в неё групп. Этот аргумент может быть равен **nil** для получения списка групп в корне иерархии, отобранной по текущему фильтру групп ([ФильтрГруппы](#)).

Назначение

Данная функция возвращает для указанной группы, отобранной по текущему фильтру групп, массив принадлежащих ей записей (подгрупп), также подходящих под этот фильтр. Для получения списка групп в корне иерархии необходимо передать в функцию **nil**.

Если переданная в функцию запись не найдена в иерархии, функция возвращает **nil**.

Рассмотрим пример. Пусть в картотеке имеется иерархия групп:

```
Root  
  Group1  
  Group2  
    Group21  
    Group22  
  Group3
```

Пусть на картотеку наложен фильтр групп, представляющий собой следующий список записей: [Group1, Group21].

Тогда функция **СписокГруппы** вернет следующие значения:

```
СписокГруппы(nil)      [Group1,Group2]  
СписокГруппы(Group1)  [ ]  
СписокГруппы(Group2)  [Group21]  
СписокГруппы(Group3)  nil
```

В корне иерархии при указанном фильтре видны две группы: Group1 (она входит в само выражение фильтра) и Group2 (она содержит группу Group21, также указанную в фильтре).

В группе Group1 нет подгрупп, поэтому возвращается пустой массив.

В группе Group2 под фильтром видна только группа Group21.

Группа Group3 вообще отсутствует в иерархии групп, отобранной по фильтру.

Пример

```
-- процедура выводит список всех групп, видимых под  
-- текущим фильтром  
proc Recursive(ЭтаГруппа :Запись);  
var Группы: СписокЗаписей;  
var Записи: Запись [];  
var i,n: integer;  
  Группы = Картотека.ФильтрГрупп;  
  Записи = Группы.СписокГруппы(ЭтаГруппа);  
  n = LengthOfArray(Записи);  
  for i=1..n do  
    trace(Записи[i]);  
  end;  
  for i=1..n do  
    Recursive(Записи[i]);  
  end;  
end;
```


Описание

```
ФильтрГруппы(Группа :Запись) :Строка;  
GroupFilter (Group :Record) :String;
```

Аргументы

Группа - ссылка на группу, для которой необходимо получить выражение фильтра.

Назначение

Данная функция возвращает для указанной группы (в том числе и для корня, если в функцию передано значение nil) строковое выражение, которое должно быть наложено на записи картотеки для получения списка записей этой группы (то есть это фильтр, который картотека автоматически накладывает при входе в эту группу).









Пример

```
proc ApplyGroup(TheRecord :Record);  
  Cardfile.Filter = Cardfile.GroupFilter(TheRecord);  
end;
```

Класс *СтолбецКартотеки* / *CardfileColumn* используется для работы со столбцами картотек Студии и является наследником классов [Объект](#) и [СтолбецТаблицы](#). Программа на ТБ.Скрипт может получить доступ к объекту *СтолбецКартотеки*, являющемуся свойством картотеки, и изменять свойства столбца.

Внимание! Создать объект класса *СтолбецКартотеки* напрямую нельзя. Такие объекты создаются только внутри системы и используются как свойства других объектов (класса [Картотека](#)).

Непосредственно в классе *СтолбецКартотеки* определены следующие свойства:

-  [Событие ПриВходе / OnEnter](#)
-  [Событие ПриПроверке / OnVerify](#)
-  [Событие ПриВыходе / OnExit](#)
-  [Событие ПриВыводе / OnOutput](#)
-  [Событие ПриВводе / OnInput](#)
-  [Событие ПриНаборе / OnType](#)
-  [Событие ПриОбзоре / OnLookup](#)
-  [Событие ПриРисовании / OnDraw](#)

См. также раздел [Последовательность событий в объектах СтолбецКартотеки](#).

Описание

ПриВводе : Строка;
OnInput : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при завершении редактирования клетки картотеки, а именно: после события [OnVerify](#), но перед событием [OnExit](#).

Событие позволяет контролировать запись новых значений в поля документов картотеки.

Обработчик

```
func OnInput(Column :CardfileColumn;Rec :Record; Value :Variant ) :Logical;
```

Параметры:

Column - указатель на объект класса **СтолбецКартотеки**, в котором произошло событие;

Rec - указатель на документ, поле которого вводится;

Value - введенное пользователем значение. В момент вызова события оно еще не сохранено в поле записи (т.е. существует возможность узнать и новое, и старое значение, что иногда необходимо).

Обработчик события должен вернуть TRUE, чтобы разрешить присвоение нового значения полю документа. В случае возврата FALSE введенное в клетку картотеки значение игнорируется (поле документа не меняется).

С помощью этого события можно реализовывать вычисляемые поля. Для этого необходимо задать вычисляемый тип столбца и обрабатывать события [OnOutput](#) и, при необходимости, **OnInput**. Через последнее из них прикладная программа получает информацию о введенном пользователем значении и может его где-либо сохранить, а с помощью первого – реализует вывод на экран.

Тип параметра **Value** в обработчике **OnInput** должен совпадать с типом, возвращаемым обработчиком события **OnOutput**.

Пример

```
func Column_OnInput(Column :CardfileColumn; Doc :Document;  
    Value : Numeric) :Logical;  
    -- заполняем массив значениями из редактировавшихся записей  
    XX[Doc.DocID] = Value;  
    return TRUE;  
end;
```

Событие ПриВходе / OnEnter

Описание

ПриВходе : Строка;
OnEnter : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит каждый раз, когда пользователь начинает вводить что-либо в клетку картотеки (например, с помощью inplace-редактора или по нажатию флага, если столбец содержит поле логического формата). Событие возможно только в столбцах, в которых разрешено редактирование непосредственно в окне картотеки (свойство [CanEdit](#) равно TRUE).

Обработчик

```
func OnEnter(Column :CardfileColumn; Rec :Record) :Logical;
```

Параметры:

Column - указатель на объект класса **СтолбецКартотеки**, в котором произошло событие;

Rec - содержит указатель на документ, поле которого начинает редактироваться.

Обработчик должен вернуть TRUE, если редактирование поля следует разрешить, или FALSE, если редактирование запрещено.

Пример

```
func ColumnData_OnEnter(Column :CardfileColumn; Document :Document) :Logical;  
  if Document.Дата < 01.01.2007 then  
    return FALSE;  
  else  
    return TRUE;  
  fi;  
end;
```

Описание

ПриВыводе : Строка;
OnOutput : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит каждый раз при перерисовке клетки.

Обработчик

func **OnOutput**(Column :CardfileColumn; Rec :Record; Action :[Template.OutputTypes](#); var Format :String) :Variant;

Параметры:

Column - указатель на объект класса **СтолбецКартотеки**, в котором произошло событие;
Rec - указатель на документ, поле которого требует перерисовки;
Action - указывает, что послужило причиной запроса значения для вывода;
Format - параметр, позволяющий изменить текущий формат отображения значения для данного конкретного случая (столбец, документ, действие).

Функция должна вернуть значение, которое требуется отобразить на экране. Тип возвращаемого значения может быть произвольным.

Обработчик позволяет произвольным образом управлять тем, что отображается в клетке. Возвращаемое обработчиком значение не влияет собственно на значение клетки – изменяется лишь значение, предназначенное для вывода.

Если для столбца назначено какое-либо форматное преобразование или трансляция, то данное событие происходит до преобразования и позволяет подменить строку форматного преобразования.

Данное событие не происходит во время редактирования клетки картотеки по месту (с помощью inplace-редактора).

Внимание! Внутри обработчика нельзя вызывать любые функции, выводящие что-либо на экран (например, диалоги), так как это приведет к заикливанию.

Пример

```
-- обработка вычисляемого столбца
func Column_OnOutput(Column :CardfileColumn; Doc :Document; Action :Template.OutputTypes; var Format :String) : Numeric;
  if Column.FieldName = "Итого" then
    return Doc.Сумма*Док.Количество;
  end;
end;
```

Описание

ПриВыходе : Строка;
OnExit : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит каждый раз, когда пользователь закончил ввод нового значения в клетку картотеки и оно уже сохранено в соответствующем поле записи.

Событие возможно только в столбцах, в которых разрешено редактирование непосредственно в окне картотеки (свойство [CanEdit](#) равно TRUE).

Внимание! Событие возникает, только если значение клетки картотеки было изменено.

Обработчик

```
proc OnExit(Column :CardfileColumn; Rec :Record);
```

Параметры:

Column - указатель на объект класса **СтолбецКартотеки**, в котором произошло событие;

Rec - указатель на документ, поле которого было изменено.

Пример

```
Изменено : Целое;  
-- ведем статистику обработки документов  
proc Column_OnExit(Column :CardfileColumn; Rec :Record);  
    Изменено = Изменено + 1;  
end;
```

Описание

ПриНаборе : Строка;
OnType : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при нажатии какой-либо клавиши (пока событие происходит только при нажатии клавиши *Ввод*) во время редактирования значения в клетке картотеки (в inplace-редакторе).

Обработчик позволяет выполнить нестандартные операции по присвоению вспомогательных переменных или ввести в картотеку дополнительные интерактивные возможности.

Обработчик

```
func OnType(Column :CardfileColumn; Rec :Record; Key :String; Value :Variant; var  
NewValue :Variant) :Logical;
```

Параметры:

Column - указатель на объект класса **СтолбецКартотеки**, в котором произошло событие;
Rec - ссылка на документ, в поле которого произошло событие;
Key - сейчас это код 13 (клавиша Enter), в дальнейшем будет содержать код нажатой клавиши;
Value - введенное пользователем значение. Тип **Value** определяется типом адресуемой по ссылке переменной либо непосредственно типом поля, если оно не является ссылочным;
NewValue - ссылка на объект (если поле ссылочное) или новое значение соответствующего типа (если поле не ссылочное), которые следует присвоить полю.

При работе с ссылочными полями возможна ситуация, когда **Value** и **NewValue** имеют различный тип, например, **Value** – строка (разыменованная ссылка вида Контрагент.Название), а **NewValue** – ссылка на документ (карточку товара, контрагента и пр.).

Если функция возвращает TRUE, система выполняет стандартную обработку поля (и, в частности, ссылочного поля с разрешенным ручным вводом). Стандартные действия системы описываются ниже.

Если функция возвращает FALSE, то в поле вводится значение **NewValue**, которое должен установить обработчик. Обратите внимание, что в этом случае созданное внутри клетки поле ввода (inplace-редактор) автоматически не закрывается, и, чтобы его закрыть, необходимо вызвать в обработчике метод: Cardfile.EndEdit(True).

Стандартная обработка набора символов в поле подразумевает, в большинстве случаев, запись текущего значения (**Value**) в клетку картотеки и отображение его на экране (пока без присвоения этого значения полю записи, связанному со столбцом; присвоение выполняется позднее, после проверки допустимости этой операции через обработчик события **ПриПроверке**). Особый случай представляет ссылочное поле. Дело в том, что если в ссылочном поле разрешен ручной ввод, то при входе в него не происходит автоматического открытия картотеки, а появляется inplace-редактор, в котором пользователь может набрать некоторый текст.

При этом, если ссылка содержит лишь имя класса (например, "Контрагент") без уточнения имени свойства этого класса через точку (например, "Контрагент.КорИмя") и в свойствах столбца (в диалоге визуального редактора шаблонов) не установлено свойство "Обзор по полю", то набранный текст интерпретируется как DocID (в формате ":").

Если ссылка содержит имя класса и, через точку, имя свойства данного класса, то набранный текст используется для поиска документа по данному свойству (полю). При точном совпадении ссылка на найденный документ автоматически записывается в поле, в противном случае открывается картотека и позиционируется на ближайший похожий документ.

Если установлено свойство "Обзор по полю", то осуществляются аналогичные действия, но по заданному полю.

Пример

```
func Счет_ПриНаборе(C:CardfileColumn; D:Document; Key:String;  
ИмяКонтрагента:String;  
var NewValue:Справочник.Контрагент) :Logical;
```

```

var x :Справочник.Контрагент;
var s :String; -- фильтр
if Key<>CHR(13) then -- ввод строки еще не завершен
    return Истина;
end;

-- создаем фильтр по контрагентам, название которых
-- начинается с введенных символов
s = "Соотв(Наименование, '"+ ИмяКонтрагента+"*')";
-- открываем картотеку контрагентов
if КартотекаКонтрагентов.ВыполнитьКартотеку(x,s) = cmOk then
    -- присваиваем новое значение
    NewValue = x;
end;


Cardfile.EndEdit(Истина);
-- сообщаем системе, что всю обработку мы выполнили сами
-- и параметр NewValue содержит нужное значение
return Ложь;
end;

```


Описание

ПриОбзоре : Строка;
OnLookup : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при попытке пользователя ввести значение в ссылочное поле или поле ввода с кнопкой обзора .

Если в поле разрешен ручной ввод, то событие возникает при нажатии кнопки Обзор справа от поля (она появляется при активации inplace-редактора в клетке столбца, при условии, что в свойствах столбца в визуальном редакторе была включена соответствующая опция) или *PgUp*.

В ссылочном поле это событие также происходит после нажатия любой клавиши при условии, что ручной ввод в него запрещен и поле в данный момент выделено.

Кроме того, следует иметь в виду, что если поле доступно только на чтение и имеет кнопку обзора, то событие **ПриОбзоре** в нем не возникает, однако программист может узнать о нажатии на кнопку обзора в таком поле с помощью обработчика события [ПриНажатии](#) (для объекта самой картотеки).

Обработчик

```
func OnLookup(Column :CardfileColumn; Rec:Record; Value :Variant; var NewValue :Variant) :Logical;
```

Параметры:

Column - указатель на объект класса **СтолбецКартотеки**, в котором произошло событие;

Rec - ссылка на документ, в поле которого произошло событие;

Value - текущее значение поля. Тип **Value** определяется несколькими факторами. Если поле содержит разыменованную ссылку, то есть выражение вида **<ссылка на запись>.<свойство>**, то тип **Value** соответствует типу адресуемого по ссылке свойства (переменной). Если в поле указана лишь ссылка на запись (то есть там фактически хранится переменная **<ссылка на запись>** без имени свойства), но в диалоге свойств клетки корректно заполнено поле **Обзор по полю**, то тип **Value** тот же, что и поле для обзора. Если же ссылочное поле не содержит разыменования (ни в явном виде, ни в свойствах клетки), то тип **Value** должен быть строковым, поскольку система предполагает указание записи с помощью строкового представления ее **DocID**. Во всех остальных случаях (то есть если поле не является ссылочным) тип **Value** определяется непосредственно типом переменной, отображаемой в поле;

NewValue - ссылка на объект (если поле ссылочное) или новое значение соответствующего типа (если поле не ссылочное), которые следует присвоить полю.

Если функция возвращает TRUE, то поле обрабатывается системой по умолчанию (см. ниже). Если функция возвращает FALSE, то никакой стандартной обработки не производится и функция должна присвоить переменной **NewValue** требуемое значение, которое затем попадает в поле.

Стандартная обработка данного события включает следующие действия:

- открытие календаря (для полей типа "Дата");
- открытие калькулятора (для полей типа "Число");
- открытие картотеки (для ссылочных полей);
- открытие списка выбора (для перечислимых полей).

Пример

```
func Названия_ПриОбзоре(C:CardfileColumn; D:Document;  
  Value :String; var NewValue :String) :Logical;  
  var x :Справочник.Контрагент;  
  
  if КартотекаКонтрагентов.ВыполнитьКартотеку(x) = cmOk then  
    -- присваиваем новое значение  
    NewValue = x.Наименование;  
  end;  
  
  return Ложь;  
end;
```

Описание

ПриПроверке : Строка;
OnVerify : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит каждый раз, когда пользователь закончил ввод нового значения в клетку картотеки, но новое значение еще не присвоено полю записи. При обработке данного события программа может проверить введенное значение на допустимость и, в случае необходимости, откорректировать его.

Событие возможно только в столбцах, в которых разрешено редактирование непосредственно в окне картотеки (свойство [CanEdit](#) равно TRUE). Событие происходит до событий [OnInput](#) и [OnExit](#).

Обработчик

```
func OnVerify(Column :CardfileColumn; Rec :Record; var Value :Variant) :Logical;
```

Параметры:

Column - указатель на объект класса **СтолбецКартотеки**, в котором произошло событие;

Rec - указатель на документ, поле которого было отредактировано;

Value - новое значение, которое требуется проверить на допустимость, прежде чем присваивать его полю записи. Тип параметра **Value** должен совпадать с типом поля записи.

Обработчик может выполнить одно из трех действий:

1. отменить операцию ввода; при этом в клетке картотеки отобразится прежнее значение, которое она содержала до редактирования; в этом случае обработчик должен вернуть значение FALSE;
2. разрешить присвоение нового значения в том виде, как оно было введено пользователем; в этом случае обработчик должен вернуть значение TRUE;
3. откорректировать введенное пользователем значение, сделав его допустимым, и разрешить присвоение этого модифицированного значения; в этом случае обработчик должен присвоить переменной Value корректное значение и вернуть TRUE.

Пример

```
func Column_OnVerify(Column :CardfileColumn; Document :Document; var S: String) :Logical;  
  if Column.FieldName = "Название" then  
    -- приводим название к верхнему регистру  
    S = Up(S);  
  end;  
  return TRUE;  
end;
```

Описание

ПриРисовании : Строка;
OnDraw : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит перед отрисовкой ячейки картотеки.

Обработчик

```
proc OnDraw(Column :CardfileColumn; Rec :Record; Selected :Logical; var Color :Integer; Font :Font);
```

Параметры:

Column - указатель на объект класса **СтолбецКартотеки**, в котором произошло событие;

Rec - указатель на документ, поле которого должно выводиться в текущей клетке;

Selected - логический параметр. Если он равен TRUE, ячейка выделена;

Color - цвет ячейки;

Font - шрифт ячейки.

Параметры **Color** и **Font** можно изменять внутри обработчика, управляя тем самым внешним видом картотеки. Если процедура не меняет цвет и шрифт, для отрисовки будут использованы их стандартные значения по умолчанию.









Пример

```
-- "веселая" картотека в разноцветную крапинку  
proc Column_OnDraw(Column :CardfileColumn; Doc :Document; Selected : Logical; Color : Integer; Font : Font);  
    color = random(255)*256*256 + random(255)*256 + random(255);  
end;
```

Класс *СтолбецПодтаблицы* / *SubCardfileColumn* используется для работы со столбцами подтаблиц картотек и наследует все свойства и методы родительских классов [Объект/Object](#) и [СтолбецТаблицы](#).

Внимание! Создать объект класса *СтолбецПодтаблицы* напрямую нельзя. Такие объекты создаются только внутри системы и используются как свойства других объектов (класса [ПодтаблицаКартотеки](#)).

Программа на ТБ.Скрипт может получить доступ к объекту *СтолбецПодтаблицы*, являющемуся свойством подтаблицы, и изменять параметры этого столбца. Непосредственно в классе *СтолбецКартотеки* определены следующие свойства:

-  [Событие ПриВходе / OnEnter](#)
-  [Событие ПриПроверке / OnVerify](#)
-  [Событие ПриВыходе / OnExit](#)
-  [Событие ПриВыводе / OnOutput](#)
-  [Событие ПриВводе / OnInput](#)
-  [Событие ПриНаборе / OnType](#)
-  [Событие ПриОбзоре / OnLookup](#)
-  [Событие ПриРисовании / OnDraw](#)

Описание

ПриВводе : Строка;
OnInput : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при завершении редактирования клетки подтаблицы, а именно: после события [OnVerify](#), но перед событием [OnExit](#).

Событие позволяет контролировать запись новых значений в поля подтаблицы картотеки.

Обработчик

```
func OnInput(Column :SubCardfileColumn; Struct :Structure; Value :Variant ) :Logical;
```

Параметры:

Column - указатель на объект класса **СтолбецПодтаблицы**, в котором произошло событие;

Struct - указатель на структуру, поле которой вводится;

Value - введенное пользователем значение. В момент вызова события оно еще не сохранено в поле структуры (т.е. существует возможность узнать и новое, и старое значение, что иногда необходимо).

Обработчик события должен вернуть TRUE, чтобы разрешить присвоение нового значения полю структуры. В случае возврата FALSE введенное в клетку структуры значение игнорируется (поле структуры не меняется).

С помощью этого события можно реализовывать вычисляемые поля. Для этого необходимо задать вычисляемый тип столбца и обрабатывать события [OnOutput](#) и, при необходимости, **OnInput**. Через последнее из них прикладная программа получает информацию о введенном пользователем значении и может его где-либо сохранить, а с помощью первого – реализует вывод на экран.

Тип параметра **Value** в обработчике **OnInput** должен совпадать с типом, возвращаемым обработчиком события **OnOutput**.

Пример

```
func Column_OnInput(Column:SubCardfileColumn; Struct:Structure; Value:Numeric) :Logical;  
    КоличествоЗаказанногоТовара[Struct.Code]  
        = КоличествоЗаказанногоТовара[Struct.Code]  
        - Struct.Количество  
        + Value;  
    return TRUE;  
end;
```

Описание

ПриВходе : Строка;
OnEnter : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит каждый раз, когда пользователь начинает вводить что-либо в клетку подтаблицы. Событие возможно только в столбцах, в которых разрешено редактирование непосредственно в подтаблице (свойство [CanEdit](#) равно TRUE).

Обработчик

func **OnEnter**(Column :SubCardfileColumn; Struct :[Structure](#)) :Logical;

Параметры:

Column - указатель на объект класса **СтолбецПодтаблицы**, в котором произошло событие;
Struct - содержит указатель на структуру, поле которой начинает редактироваться.

Обработчик должен вернуть TRUE, если редактирование поля следует разрешить, или FALSE, если редактирование запрещено.

Пример

```
func ColumnData_OnEnter(Column:SubCardfileColumn; Struct:Structure) :Logical;  
  if Struct.Дата < 01.01.2000 then  
    return FALSE;  
  else  
    return TRUE;  
  fi;  
end;
```

Описание

ПриВыводе : Строка;
OnOutput : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит каждый раз при перерисовке клетки.

Обработчик

```
func OnOutput(Column :SubCardfileColumn; Struct :Structure; Action :Template.OutputTypes) :Variant;
```

Параметры:

Column - указатель на объект класса **СтолбецПодтаблицы**, в котором произошло событие;

Struct - указатель на структуру, поле которой требует перерисовки;

Action - параметр, который передает причину события.

Функция должна вернуть значение, которое требуется отобразить на экране. Тип возвращаемого значения может быть произвольным.

Обработчик позволяет произвольным образом управлять тем, что отображается в клетке. Возвращаемое обработчиком значение не влияет собственно на значение клетки - изменяется лишь значение, предназначенное для вывода.

Данное событие не происходит во время редактирования клетки картотеки по месту (с помощью inplace-редактора).

Внимание! Внутри обработчика нельзя вызывать любые функции, выводящие что-либо на экран (например, диалоги), так как это приведет к закливанию.

Пример

```
-- обработка вычисляемого столбца
func Column_OnOutput(Column:SubCardfileColumn; Struct:Structure;
  Action :Template.OutputTypes) :Numeric;
  if Column.FieldName = "Сумма" then
    return Struct.Цена*Struct.Количество;
  end;
end;
```

Описание

ПриВыходе : Строка;
OnExit : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит каждый раз, когда пользователь закончил ввод нового значение в клетку подтаблицы и оно уже сохранено в соответствующем поле записи.

Событие возможно только в столбцах, в которых разрешено редактирование непосредственно в подтаблице (свойство [CanEdit](#) равно TRUE).

Внимание! Событие возникает, только если значение клетки подтаблицы было изменено.

Обработчик

```
proc OnExit(Column :SubCardfileColumn; Struct :Structure);
```

Параметры:

Column - указатель на объект класса **СтолбецПодтаблицы**, в котором произошло событие;
Struct - указатель на структуру, поле которой было изменено.

Пример

```
-- после редактирования позиции выводим код ТМЦ  
-- в строку состояния  
Column_OnExit(Column:SubCardfileColumn; Struct:Structure);  
Hint(Struct.GetField("Наименование")  
+" / "  
+Struct.GetField("Код"));  
end;
```


Описание

ПриНаборе : Строка;
OnType : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при нажатии какой-либо клавиши (пока событие происходит только при нажатии клавиши *Ввод*) во время редактирования значения в клетке подтаблицы картотеки (в inplace-редакторе).

Обработчик позволяет выполнить нестандартные операции по присвоению вспомогательных переменных или ввести в картотеку дополнительные интерактивные возможности. Событие аналогично одноименному событию в объекте [СтолбецКартотеки](#).

Обработчик

```
func OnType(Column :SubCardfileColumn; Struct :Structure; Key :String; Value :Variant; var  
NewValue :Variant) :Logical;
```

Параметры:

Column - указатель на объект класса **СтолбецПодтаблицы**, в котором произошло событие;

Struct - ссылка на структуру, в поле которой произошло событие;

Key - сейчас это код 13 (клавиша Enter), в дальнейшем будет содержать код нажатой клавиши;

Value - введенное пользователем значение. Тип **Value** определяется типом адресуемой по ссылке переменной либо непосредственно типом поля, если оно не является ссылочным;

NewValue - ссылка на объект (если поле ссылочное) или новое значение соответствующего типа (если поле не ссылочное), которые следует присвоить полю.

При работе со ссылочными полями возможна ситуация, когда **Value** и **NewValue** имеют различный тип, например, **Value** – строка (разыменованная ссылка вида Контрагент.Название), а **NewValue** – ссылка на документ (карточку товара, контрагента и пр.).


Если функция возвращает TRUE, система выполняет стандартную обработку поля (и, в частности, ссылочного поля с разрешенным ручным вводом). Стандартные действия системы аналогичны тем, что происходят для события [СтолбецКартотеки.ПриНаборе](#).

Если функция возвращает FALSE, то в поле вводится значение **NewValue**, которое должен установить обработчик. Обратите внимание, что в этом случае созданное внутри клетки поле ввода (inplace-редактор) автоматически не закрывается, и, чтобы его закрыть, необходимо вызвать в обработчике метод: Cardfile.EndEdit(True).

Описание

ПриОбзоре : Строка;
OnLookup : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при попытке пользователя ввести значение в ссылочное поле или поле ввода с кнопкой обзора  в столбце подтаблицы.

Если в поле разрешен ручной ввод, то событие возникает при нажатии кнопки Обзор справа от поля (она появляется при активации inplace-редактора в клетке столбца, при условии, что в свойствах столбца в визуальном редакторе была включена соответствующая опция) или *PgUp*.

В ссылочном поле это событие также происходит после нажатия любой клавиши при условии, что ручной ввод в него запрещен и поле в данный момент выделено.

Кроме того, следует иметь в виду, что если поле доступно только на чтение и имеет кнопку обзора, то событие **ПриОбзоре** в нем не возникает, однако программист может узнать о нажатии на кнопку обзора в таком поле с помощью обработчика события [ПриНажатии](#) (для объекта самой картотеки).

Обработчик

```
func OnLookup(Column :SubCardfileColumn; Struct :Structure; Value :Variant; var  
NewValue :Variant) :Logical;
```

Параметры:

Column - указатель на объект класса **СтолбецПодтаблицы**, в котором произошло событие;

Struct - ссылка на структуру, в поле которой произошло событие;

Value - текущее значение поля. Тип **Value** определяется несколькими факторами. Если поле содержит разыменованную ссылку, то есть выражение вида **<ссылка на запись>.<свойство>**, то тип **Value** соответствует типу адресуемого по ссылке свойства (переменной). Если в поле указана лишь ссылка на запись (то есть там фактически хранится переменная **<ссылка на запись>** без имени свойства), но в диалоге свойств клетки корректно заполнено поле **Обзор по полю**, то тип **Value** тот же, что и поле для обзора. Если же ссылочное поле не содержит разыменования (ни в явном виде, ни в свойствах клетки), то тип **Value** должен быть строковым, поскольку система предполагает указание записи с помощью строкового представления ее **DocID**. Во всех остальных случаях (то есть если поле не является ссылочным) тип **Value** определяется непосредственно типом переменной, отображаемой в поле;

NewValue - ссылка на объект (если поле ссылочное) или новое значение соответствующего типа (если поле не ссылочное), которые следует присвоить полю.

Если функция возвращает TRUE, то поле обрабатывается системой по умолчанию (см. ниже). Если функция возвращает FALSE, то никакой стандартной обработки не производится и функция должна присвоить переменной **NewValue** требуемое значение, которое затем попадает в поле.

Стандартная обработка данного события включает следующие действия:

- открытие календаря (для полей типа "Дата");
- открытие калькулятора (для полей типа "Число");
- открытие картотеки (для ссылочных полей);
- открытие списка выбора (для перечислимых полей).

Событие полностью аналогично одноименному событию в классе [СтолбецКартотеки](#).

Описание

ПриПроверке : Строка;
OnVerify : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит каждый раз, когда пользователь закончил ввод нового значения в клетку подтаблицы, но новое значение еще не присвоено полю записи. При обработке данного события программа может проверить введенное значение на допустимость и, в случае необходимости, откорректировать его.

Событие возможно только в столбцах, в которых разрешено редактирование непосредственно в окне картотеки (свойство [CanEdit](#) равно TRUE). Событие происходит до событий [OnInput](#) и [OnExit](#).

Обработчик

```
func OnVerify(Column :SubCardfileColumn; Struct :Structure; var Value :Variant) :Logical;
```

Параметры:

Column - указатель на объект класса **СтолбецПодтаблицы**, в котором произошло событие;
Struct - указатель на структуру, поле которой было отредактировано;
Value - новое значение, которое требуется проверить на допустимость, прежде чем присваивать его полю записи. Тип параметра **Value** должен совпадать с типом поля записи.

Обработчик может выполнить одно из трех действий:

1. отменить операцию ввода; при этом в клетке картотеки отобразится прежнее значение, которое она содержала до редактирования. В этом случае обработчик должен вернуть значение FALSE;
2. разрешить присвоение нового значения в том виде, как оно было введено пользователем. В этом случае обработчик должен вернуть значение TRUE;
3. откорректировать введенное пользователем значение, сделав его допустимым, и разрешить присвоение этого модифицированного значения. В этом случае обработчик должен присвоить переменной **Value** корректное значение и вернуть TRUE.

Пример

```
func Column_OnVerify(Column:SubCardfileColumn; Struct:Structure; var S:String) :Logical;  
  if Column.FieldName = "Наименование" then  
    -- приводим название к верхнему регистру  
    S = Up(S);  
  end;  
  return TRUE;  
end;
```

Описание

ПриРисовании : Строка;
OnDraw : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит перед отрисовкой ячейки подтаблицы.

Обработчик

```
proc OnDraw(Column :SubCardfileColumn; Struct :Structure; Selected :Logical; var Color :Integer;  
Font :Font);
```

Параметры:

Column - указатель на объект класса **СтолбецПодтаблицы**, в котором произошло событие;
Struct - указатель на структуру, поле которой должно выводиться в текущей клетке;
Selected - логический параметр. Если он равен TRUE, ячейка выделена;
Color - цвет ячейки;
Font - шрифт ячейки.

Параметры **Color** и **Font** можно изменять внутри обработчика, управляя тем самым внешним видом картотеки. Если процедура не меняет цвет и шрифт, для отрисовки будут использованы их стандартные значения по умолчанию.

Пример

```
proc Column_OnDraw(Column:SubCardfileColumn; Struct:Structure;  
  Selected:Logical; var Color:Integer; Font:Font);  
  -- выделяем жирным шрифтом крупные суммы  
  if Column.FieldName = "Сумма" then  
    if Struct.Сумма > 1000.0 then  
      Font.Bold = true;  
    end;  
  end;  
end;  
end;
```

Класс *СтолбецТаблицы/TableColumn*, являющийся наследником класса [Объект](#), используется для работы со столбцами таблиц картотек.

Внимание! Создать объекты класса *СтолбецТаблицы* напрямую нельзя, они создаются только внутри системы и используются как свойства других объектов класса [Картотека](#). Однако, программный интерфейс позволяет получить доступ к объекту *СтолбецТаблицы*, который является свойством картотеки, и изменять свойства столбца.

Непосредственно в классе *СтолбецТаблицы* определены следующие свойства:

- [Поле ИмяПоля / FieldName](#)
- [Поле Надпись / Caption](#)
- [Поле Подсказка / Hint](#)
- [Поле КонтекстПомощи / HelpContext](#)
- [Поле Ширина / Width](#)
- [Поле МинШирина / MinWidth](#)
- [Поле Виден / Visible](#)
- [Поле Выравнивание / Alignment](#)
- [Поле Цвет / Color](#)
- [Поле Шрифт / Font](#)
- [Поле ТипПоля / FieldType](#)
- [Поле ТипКолонки / ColumnType](#)
- [Поле ФорматКолонки / ColumnFormat](#)
- [Поле ФорматМаскиПоиска / FindMaskFormat](#)
- [Поле Формат / Format](#)
- [Поле АвтоОбзор / AutoLookup](#)
- [Поле ПолеОбзора / LookupField](#)
- [Поле ФильтрОбзора / LookupFilter](#)
- [Поле ИмяКартотеки / CardfileName](#)
- [Поле Кнопка / Button](#)
- [Поле Список / List](#)
- [Поле МожноРедактировать / CanEdit](#)
- [Поле МожноУпорядочить / CanSort](#)
- [Поле МожноСуммировать / CanSummary](#)
- [Поле МожноРазыменовывать / CanDesignate](#)
- [Поле Индекс / Index](#)

Производными от класса *СтолбецТаблицы* являются классы [СтолбецКартотеки / CardfileColumn](#) и [СтолбецПодтаблицы / SubCardfileColumn](#).

Описание

АвтоОбзор : Логическое;
AutoLookup : Logical;

Назначение

Включает/отключает автоматический обзор. Если в клетке включен автоматический обзор (**АвтоОбзор** = ИСТИНА), то попытка ввести в клетку какой-либо текст будет приводить к открытию картотеки (см. [ИмяКартотеки](#)) для выбора требуемой записи.

Если автоматический обзор отключен, то в клетку можно ввести текст, который интерпретируется системой как значение поля обзора (см. [ПолеОбзора](#)), и непосредственно обзор данной картотеки по полю производится затем клавишей *PgUp*.

Примечание. Данное свойство имеет смысл только для столбцов, имеющих ссылочный [тип поля](#) (**СсылочноеПоле / ReferenceField**).

Пример

```
Column.AutoLookup = false;
```

Описание

Виден : Логическое;
Visible : Logical;

Назначение

Позволяет узнать, видим ли столбец, и управлять его видимостью. Когда поле имеет значение TRUE, столбец виден.

Пример

```
proc ButOkClick(S:String);
var cc:TableColumn;

cc = CardFile.СтолбецПоПолю[ "Корресп.Название" ];
if cc = nil then
  message("Ошибка определения столбца Контрагентов");
else
  -- меняем атрибуты
  cc.Width = 200;
  cc.Visible = TRUE;
  cc.Caption = "Контрагент";
  cc.Alignment = 0; -- прижать текст влево
end;
end;
```

Описание

Выравнивание: Шаблон.[ТипыВыравнивания](#);

Alignment: Template.[AlignmentTypes](#);

Назначение

Позволяет узнать и изменить выравнивание текста в столбце. Поле может принимать одно из следующих значений, определенных в типе [Шаблон.ТипыВыравнивания](#):

ВыравниватьВлево - выравнивание по левому краю;

ВыравниватьВправо - выравнивание по правому краю;

ВыравниватьПоЦентру - центрирование.

Пример

```
proc ButOkClick(S:String);
var cc:TableColumn;

cc = CardFile.СтолбецПоПолю["Корресп.Название"];
if cc = nil then
  message("Ошибка определения столбца Контрагентов");
else
  -- меняем атрибуты
  cc.Width = 200;
  cc.Visible = TRUE;
  cc.Caption = "Контрагент";
  cc.Alignment = Kernel.Template.LeftAlign; -- прижать текст влево
end;
end;
```


Описание

ИмяКартотеки : Строка;
CardfileName : String;

Назначение

Свойство позволяет узнать и изменить имя картотеки, используемой для заполнения клетки (т.е. ввода ссылки на конкретную записи того типа, что соответствует типу ссылочного поля).

Свойство может быть изменено не только программно, но и на стадии проектирования – в диалоге свойств столбца.

Примечание. Данное свойство имеет смысл только для столбцов, имеющих ссылочный [тип поля](#) (**СсылочноеПоле / ReferenceField**).

См. также [Поле ПолеОбзора](#).

Пример

```
func Column_OnEnter(Column :TableColumn; Document :Document):Logical;  
  -- конструируем имя картотеки по примеру:  
  -- из поля "Customer" получаем "card_Customers"  
  Column.CardfileName = "card_"+Column.FieldName+"s";  
  return TRUE;  
end;
```

Описание

ИмяПоля : Строка;
FieldName : String;

Назначение

Позволяет узнать имя поля, которое выводится в столбце. Поле доступно только на чтение.

Пример

```
-- симулируем метод ColumnByField
func FieldByName(Name: String):TableColumn;
var i: integer;
  for i=1..CardFile.ColumnsCount do
    if Name = CardFile.Column[i].FieldName then
      return CardFile.Column[i];
    end;
  end;
  return nil;
end;
```

Описание

Индекс :Целое;
Index :Integer;

Назначение

Поле содержит порядковый номер столбца. **Индекс** может изменяться в результате перетаскивания столбца на новую позицию. В нумерации участвуют как видимые, так и скрытые столбцы.

Поле доступно не только на чтение, но и на запись, что дает возможность изменять порядок следования столбцов программным способом.


Пример

```
proc ButtonClick(B1 :Button);  
    -- по нажатию кнопки последний столбец становится первым  
    CardFile.Column[CardFile.ColumnsCount].Index = 1;  
end;
```

Описание

Кнопка : Логическое;
Button : Logical;

Назначение

Позволяет узнать и изменить свойство столбца, определяющее наличие в его клетках кнопок обзора  у правого края.

Когда значение поля **Кнопка** равно ИСТИНА, клетки столбца имеют кнопки обзора.

Пример

```
proc DisableBrowse;
var i, n :integer;
  n = Cardfile.ColumnsCount;
  -- проход по всем столбцам картотеки
  for i=1..n do
    -- если столбец имеет кнопку выбора
    if Cardfile.Column[i].Button then
      -- запрещаем работу с ним
      Cardfile.Column[i].CanEdit = false;
    end;
  end;
end;
```

Описание

```
КонтекстПомощи : Строка;  
HelpContext : String;
```

Назначение

Свойство позволяет определить и задать путь к файлу с текстом помощи, поясняющим назначение текущего столбца таблицы.

Этот текст будет появляться в окне справке по прикладным системам, если выделен столбец и нажата клавиша **F1**.

Пример

```
CardFile.Column[1].HelpContext="C:\ПолныйПуть\ИмяФайла.htm";  
-- ПолныйПуть - полный путь к файлу ИмяФайла.htm с перечислением  
-- всех папок, начиная от папки, в которую установлена программа
```

Описание

МинШирина : Целое;
MinWidth : Integer;

Назначение

Позволяет получить и установить минимально допустимую ширину столбца картотеки (в пикселях). Программа не дает пользователю возможности сузить столбец больше указанного здесь значения.

Пример

```
proc ButMinClick(B :Button);  
  
  if State then  
    -- включаем ограничение на ширину  
    CardFile.Column[1].MinWidth = 100;  
  else  
    -- отключаем ограничение на ширину  
    CardFile.Column[1].MinWidth = 0;  
  end;  
  
end;
```

Описание

`МожноРазыменовывать` : Логическое;
`CanDesignate` : Logical;

Назначение

Данное поле управляет формированием фильтра и поиском по столбцу картотеки. При выполнении этих операций по ссылочным полям система использует фильтр, который при необходимости может быть как **разыменованным**, так и **неразыменованным**.

Предположим, в столбце картотеки выводится поле "Товар.Имя", где *Товар* – ссылочное поле, а *Имя* – одно из его свойств. В целях повышения эффективности поиска по этому столбцу имеет смысл формировать фильтр не по имени товара, а по самому товару (объекту/документу). **Такой фильтр называется неразыменованным**. Именно такой фильтр генерируется системой по умолчанию.

В тех случаях, когда требуется поиск по **разыменованному фильтру**, т.е. точно по содержимому столбца, необходимо присвоить свойству **МожноРазыменовывать** значение TRUE. Пример разыменованного фильтра: "Контрагент.ИНН".

Если свойство **МожноРазыменовывать** имеет значение FALSE, система просматривает всю цепочку разыменований в содержимом столбца справа налево и отбрасывает справа все разыменования по одному, до тех пор пока в качестве свойства разыменования не встретится ссылочный тип. Например, "Товар.Поставщик.Регион", где *Товар* и *Поставщик* – объекты, а *Регион* – строка, будет сокращен в фильтре до "Товар.Поставщик".

Доступ к данному свойству только программный. По умолчанию свойство равно FALSE.

Пример

```
Column.CanDesignate = TRUE;
```

Описание

МожноРедактировать : Логическое;
CanEdit : Logical;

Назначение

Позволяет узнать и изменить свойство столбца, определяющее, доступен ли столбец на редактирование. Если – да, то значение поля равно TRUE, иначе – FALSE. Начальное значение задается прикладным программистом в визуальном редакторе шаблонов картотеки.

Пример

```
proc ButtonClick(S:String);
var i: integer;
  for i=1..CardFile.ColumnsCount do
    if NOT CardFile.Column[i].CanEdit then
      -- показываем/скрываем все колонки, доступные
      -- только на чтение
      CardFile.Column[i].Visible = NOT CardFile.Column[i].Visible;
    end;
  end;
end;
```


Описание

МожноСуммировать : Логическое;
CanSummary : Logical;

Назначение

Позволяет узнать и изменить свойство столбца, определяющее, разрешено ли по нему суммирование. Суммировать можно только значения типа **Число** или **Целое**.

Поле лишь разрешает суммирование. Установка его в TRUE не означает автоматическое суммирование. Для выполнения суммирования пользователь должен выполнить команду [Картотека|Показывать суммы](#).

Суммарные значения отображаются в специальной строке у нижнего края таблицы.

Пример

```
МожноСуммировать
CanSummary
Назначение
Пример
proc ButtonClick(B1 :Button);
var i: integer;
var b: logical;
    b = FALSE;
    for i=1..CardFile.ColumnsCount do
        if NOT CardFile.Column[i].CanSummary then
            b = TRUE;
            -- как минимум одна колонка может быть просуммирована
            break;
        end;
    end;
    if b then
        ExecuteCommand("Картотеки","Показывать суммы");
    end;
end;
```

Описание

МожноУпорядочить : Логическое;
CanSort : Logical;

Назначение

Позволяет узнать и изменить свойство столбца, определяющее, разрешена ли сортировка по данному столбцу (то есть, может ли пользователь нажать на шапку столбца, инициировав сортировку).

Если значение поля равно TRUE, столбец можно сортировать, иначе – нельзя. Установка данного поля в TRUE не означает, что столбец будет немедленно отсортирован.

Начальное значение задается прикладным программистом в визуальном редакторе шаблонов картотеки.

Пример

```
proc ButtonClick(S:String);  
var i: integer;  
  for i=1..CardFile.ColumnsCount do  
    if NOT CardFile.Column[i].CanSort then  
      -- все несортируемые колонки делаем желтыми  
      CardFile.Column[i].Color = 256*256*0+240*256+240;  
    end;  
  end;  
end;
```

Описание

Надпись : Строка;
Caption : String;

Назначение

Позволяет узнать и изменить заголовок столбца.

Пример

```
proc ButOkClick(S:String);
var cc:TableColumn;

cc = CardFile.СтолбецПоПолю["Корресп.Название"];
if cc = nil then
  message("Ошибка определения столбца Контрагентов");
else
  -- меняем атрибуты
  cc.Width = 200;
  cc.Visible = TRUE;
  cc.Caption = "Контрагент";
  cc.Alignment = 0; -- прижать текст влево
end;
end;
```

Описание

```
Подсказка : Строка;  
Hint : String;
```

Назначение

Позволяет узнать и изменить строку контекстной подсказки, которая выводится во всплывающем окне, при наведении курсора на заголовок столбца картотеки.

Пример

```
CardFile.Column[1].Hint = "Назначение столбца";
```

Описание

ПолеОбзора : Строка;
LookupField : String;

Назначение

Свойство позволяет назначить и/или узнать, какое из полей класса записи, адресуемого по ссылке, будет использоваться для быстрого поиска и фильтрации документов при заполнении клетки картотеки.

При вводе значений в клетку они интерпретируются как значения указанного поля обзора и позволяют системе "на лету" искать документы с требуемым значением.

В том случае, если несколько записей картотеки, используемой для заполнения ссылочного поля, содержат в поле обзора фрагмент строки, введенной пользователем, система производит фильтрацию картотеки по запрашиваемому фрагменту и открывает картотеку с отфильтрованными записями.

Свойство может быть изменено не только программно, но и на стадии проектирования – в диалоге свойств столбца.

См. также [ИмяКартотеки](#).

Примечание. Данное свойство имеет смысл только для столбцов, имеющих ссылочный [тип поля](#) (**СсылочноеПоле / ReferenceField**). В этом случае в клетках столбца хранятся ссылки на объекты указанного класса записи.

Пример

```
func Column_OnEnter(Column :TableColumn; Document :Document):Logical;  
  -- если поле обзора не назначено, ...  
  if Column.LookupField = "" then  
    -- используем общеупотребительное поле "Имя"  
    -- (оно должно быть заведено в классе записи)  
    Column.LookupField = "Имя";  
    return TRUE;  
  end;  
end;
```

Описание

Список : [СписокСтрок](#);
List : [StringList](#);

Назначение

Поле обеспечивает программный доступ к объекту **СписокСтрок**, который содержит список строк трансляции. Это позволяет изменять, добавлять, удалять строки, управляющие отображением содержимого столбца.

Строки задаются в формате "отображение | значение", то есть все величины, совпадающие с конкретным "значением" будут заменены при выводе на "отображаемую" строку. Изменения, сделанные программно, не влияют на [настройки столбца картотеки](#), сделанные в дизайн-режиме.

Поле доступно только на чтение.

Пример

```
proc КнопкаИзменитьОтображение(Column :TableColumn; Display: String);  
    Column.Список.Добавить(Display);  
end;
```

Описание

ТипКолонки : [ТипыКолонки](#);

ColumnType : [ColumnTypes](#);

Назначение

Свойство позволяет установить или узнать формат типа колонки и содержит одно из значений перечислимого типа [ТипыКолонки](#).

Поле доступно на чтение и запись.

Пример

```
func FieldByName(Name: String):TableColumn;  
var i: integer;  
  for i=1..CardFile.ColumnsCount do  
    if Name = CardFile.Column[i].FieldName then  
      CardFile.Column[i].ColumnType = CardFile.FieldColumn;  
      return CardFile.Column[i];  
    end;  
  end;  
  return nil;  
end;
```

Описание

ТипПоля : [Шаблон.ТипыПоля](#);
FieldType : [Template.FieldTypes](#);

Назначение

Позволяет узнать и изменить тип поля столбца. Поле может принимать одно из следующих константных значений, определенных в типе [Шаблон.ТипыПоля](#):

СтатическийТекст - статический текст;
ОбщееПоле - поле ввода/вывода;
СтроковоеПоле - поле строки;
ЧисловоеПоле - поле числа;
ПолеДаты - поле даты;
ЛогическоеПоле - логическое поле;
ПеречислимоеПоле - перечислимое поле;
СсылочноеПоле - ссылочное поле;
ВычисляемоеПоле - вычисляемое поле.

Пример

```
proc SetDates;  
var i: integer;  
  for i=1..CardFile.ColumnsCount do  
    if CardFile.Column[i].FieldType = Kernel.Template.DateField then  
      -- если дата  
      CardFile.Column[i].Format = "dd mmmm yyyy";  
    end;  
  end;  
end;
```


Описание

ФильтрОбзора : Строка;
LookupFilter : String;

Назначение

Свойство позволяет узнать и изменить фильтр для ограничения подмножества записей, из которых можно выбирать запись для ввода значения в данную клетку. Если клетка связана с картотекой, то этот же фильтр накладывается и на записи картотеки перед её открытием.

Свойство может быть изменено не только программно, но и на стадии проектирования – в диалоге свойств столбца.

Примечание. Данное свойство имеет смысл только для столбцов, имеющих [ссылочный тип поля](#) (**СсылочноеПоле / ReferenceField**).

Пример

```
-- при обращении к столбцу "Контрагент"
func Column_OnEnter(Column :TableColumn; Document :Document):Logical;
var Сделка : ЗаписьСделка;
    -- при редактировании поля с информацией о контрагенте
    -- в документе-сделке устанавливаем фильтр на все записи
    -- за исключением того контрагента, который уже выбран
    Сделка = Document as ЗаписьСделка;
    Column.LookupFilter = "DocID <> " + Str(Сделка.Контрагент.DocID);
    return TRUE;
end;
```

Описание

Формат : Строка;
Format : String;

Назначение

Позволяет узнать и изменить форматное преобразование, используемое при выводе значения переменной, связанной со столбцом картотеки. Синтаксис и семантика форматных преобразований описаны в разделе [Преобразования формата](#).

Пример

```
proc SetDates;
var i: integer;
  for i=1..CardFile.ColumnsCount do
    if CardFile.Column[i].FieldType = Kernel.Template.DateField then
      -- если дата
      CardFile.Column[i].Format = "dd mmmm yyyy";
    end;
  end;
end;
```

Описание

ФорматКолонки : [ФорматыКолонки](#);

ColumnFormat : [ColumnFormats](#);

Назначение

Свойство позволяет установить или узнать формат колонки таблицы в картотеках и содержит одно из значений перечислимого типа [ФорматыКолонки](#).

Поле доступно на чтение и запись.

Пример

```
func FieldByName(Name: String):TableColumn;
var i: integer;
  for i=1..CardFile.ColumnsCount do
    if Name = CardFile.Column[i].FieldName then
      CardFile.Column[i].ColumnFormat = CardFile.StringFormat;
      return CardFile.Column[i];
    end;
  end;
  return nil;
end;
```

Описание

ФорматМаскиПоиска : [ФорматыМаскиПоиска](#);
FindMaskFormat : [FindMaskFormats](#);

Назначение

Свойство управляет тем, как будет формироваться маска для поиска/фильтра строки по умолчанию. Свойство доступно только программно и может содержать только одно из значений перечислимого типа [ФорматыМаскиПоиска](#).

По умолчанию, а также при отсутствии соответствующего столбца в мастере фильтров формат считается равным FindWithin.

Поле доступно на чтение и запись.

Свойство также учитывается при поиске ссылочных значений как в диалогах фильтрации/поиска, так и при inplace редактировании в картотеке.

В связи с появлением этого свойства изменилось стандартное поведение поиска ссылок при inplace редактировании. Раньше для строковых Lookup-полей (полей обзора) введенная маска безусловно обрамлялась звездочками для поиска по вхождению. Теперь, если пользователь сам ввел звездочку в состав маски, то система их добавлять не будет. Если пользователь не вводил звездочек, то они добавятся автоматически в соответствии с данным свойством FindMaskFormat.

Пример

```
func FieldByName(Name: String):TableColumn;
var i: integer;
  for i=1..CardFile.ColumnsCount do
    if Name = CardFile.Column[i].FieldName then
      CardFile.Column[i].FindMaskFormat = CardFile.FindWithin;
      return CardFile.Column[i];
    end;
  end;
  return nil;
end;
```

Описание

Цвет : Целое;
Color : Integer;

Назначение

Позволяет узнать и изменить цвет фона столбца.

Пример

```
proc SetColors;  
var i: integer;  
  for i=1..CardFile.ColumnsCount do  
    if CardFile.Column[i].FieldType = Kernel.Template.DateField then  
      -- все колонки с датами делаем розовыми  
      CardFile.Column[i].Color = 128*256*256 + 128*256 + 240;  
      -- и жирным шрифтом  
      CardFile.Column[i].Font.Bold = TRUE;  
    end;  
  end;  
end;  
end;
```

Описание

Ширина : Целое;
Width : Integer;

Назначение

Позволяет получить и установить ширину столбца картотеки (в пикселях). При записи в поле значения 0 столбец переходит в режим автоматического [подбора ширины](#).

Пример

```
proc ButOkClick(S:String);
var cc:TableColumn;

cc = CardFile.СтолбецПоПолю[ "Корресп.Название" ];
if cc = nil then
  message("Ошибка определения столбца Контрагентов");
else
  -- меняем атрибуты
  cc.Width = 200;
  cc.Visible = TRUE;
  cc.Caption = "Контрагент";
  cc.Alignment = 0; -- прижать текст влево
end;
end;
```

Описание

Шрифт : [Шрифт](#);

Font : [Font](#);

Назначение

Позволяет узнать и изменить параметры шрифта столбца.

Пример

```
proc SetColors;
var i: integer;
  for i=1..CardFile.ColumnsCount do
    if CardFile.Column[i].FieldType = Kernel.Template.DateField then
      -- все колонки с датами делаем розовыми
      CardFile.Column[i].Color = 128*256*256 + 128*256 + 240;
      -- и жирным шрифтом
      CardFile.Column[i].Font.Bold = TRUE;
    end;
  end;
end;
```

Все классы, производные от родительского класса [Объект](#), сгруппированы по своему назначению на группы.








В группу классов, предназначенных для работы с внутренними отчетами Студии, входят следующие:

- [Отчет / Report](#)
- [ГрафикОтчета / ReportChart](#)
- [ПоказательОтчета / ReportIndicator](#)
- [РазбиениеОтчета / ReportSplit](#)
- [НастройкиГрафика / ChartSettings](#)

Класс *ГрафикОтчета/ReportChart* предназначен для графического отображения результатов отчетов. Объекты данного класса входят в состав объектов класса [ФормаОтчета/ReportForm](#) и используют в качестве исходных данных данные из объекта *Отчет*, связанного с той же формой отчета.

Класс *ГрафикОтчета* является производным от класса [Объект/Object](#) и наследует от родительского класса *Объект* все свойства.

Непосредственно в классе *ГрафикОтчета / ReportChart* определены следующие свойства и методы:

-  [Поле Настройки / Settings](#)
-  [Поле ОстаткиОбороты / SumKind](#)
-  [Поле АктивнаяТаблица / ActiveTable](#)
-  [Поле ЧислоСтраниц / PageCount](#)
-  [Поле АктивнаяСтраница / ActivePage](#)
-  [Поле Значение / Value](#)
-  [Событие ПриНажатии / OnClick](#)

Создать объект класса *ГрафикОтчета* из программы на ТБ.Скрипт нельзя. Объекты данного класса создаются самой системой в контексте объектов класса *ФормаОтчета* и доступны из последних через соответствующее ссылочное поле. Разыменовывая такую ссылку, программист имеет возможность обращаться к свойствам графического отчета.

Предупреждение. Графический отчет не может быть иерархическим.

Описание

АктивнаяСтраница :Целое;
ActivePage :Integer;

Назначение

Поле позволяет узнать и изменить номер текущей страницы графика. Номер может лежать в пределах от 1 до общего числа страниц. Количество страниц выбирается системой автоматически, исходя из настроек графика и размера окна.

Пример

```
-- фрагмент кода класса-наследника формы отчета
proc ShowNextPage;
  if ReportInGraphics.ActivePage < ReportInGraphics.PageCount then
    ReportInGraphics.ActivePage = ReportInGraphics.ActivePage + 1;
  end;
end;
```

Описание

АктивнаяТаблица :Целое;
ActiveTable :Integer;

Назначение

Поле позволяет узнать и изменить номер таблицы из результатов отчета, данные из которой в текущий момент отображаются на графике. Номер может лежать в пределах от 1 до числа таблиц в объекте класса [Отчет/Report](#), связанном с объектом класса [ФормаОтчета/ReportForm](#), которому в свою очередь принадлежит описываемый объект класса [ГрафикОтчета](#). В зависимости от параметров отчета, поставляющего данные форме отчета, количество таблиц может меняться.

Иными словами, объект класса **ФормаОтчета** поддерживает встроенный объект класса **Отчет** (источник данных) и встроенный объект рассматриваемого класса **ГрафикОтчета**, отображающий эти данные.

Пример

```
-- фрагмент кода класса-наследника формы отчета,  
-- в нем определены свойства Отчет/Report  
-- и ОтчетВГрафике/ReportInGraphics,  
-- для доступа к объектам классов Отчет  
-- и ГрафикОтчета, соответственно  
proc ShowTable(x :Integer);  
  if x <= Report.TableCount then  
    ReportInGraphics.ActiveTable = x;  
  end;  
end;
```

Описание

Значение [НомерСерии :Целое, НомерЗначения :Целое] :Число;
Value [SeriesNo :Integer, ValueNo :Integer] :Numeric;

Назначение

Поле предоставляет доступ к исходным данным, по которым строится графический отчет. Получаемые значения фактически берутся из результирующих таблиц отчета, доступного через свойство [Отчет/Report](#) объекта класса **ФормаОтчета**, которому принадлежит объект рассматриваемого класса [ГрафикОтчета](#).

Поле доступно только на чтение.

В качестве номера серии выступает номер колонки активной таблицы отчета. В качестве номера значения - номер строки таблицы. Соответственно, эти номера могут изменяться от 1 до общего числа колонок и строк в таблице.

Пример

```
-- фрагмент кода класса-наследника формы отчета
func Count1 :Numeric;
var i,n :Integer;
var sum :Numeric;
    sum = 0;
    Report.CurTable = 1;
    n = Report.RowCount; -- n - число значений в 1-ой таблице
    for i=1..n do
        sum = sum + ReportInGraphics.Value[1,i];
    end;
    return sum;
end;
```

Описание

Настройки : [НастройкиГрафика](#);
Settings : [ChartSettings](#);

Назначение

Данное свойство предоставляет доступ к настройкам графического отчета, которые хранятся в виде внутреннего объекта класса [НастройкиГрафика](#).

Поле доступно не только на чтение (что дает возможность изменять свойства объекта **НастройкиГрафика**), но и на запись (то есть объект с настройками можно целиком копировать из одного графического отчета в другой или из объектов класса [График/Chart](#)).

Пример

```
ReportChart2.Settings = ReportChart1.Settings;  
ReportChart2.Settings.View3D = false;
```

Описание

ОстаткиОбороты : [Отчет.ВидыОстОбор](#);
SumKind : [Report.SumKind](#);

Назначение

Поле позволяет узнать или изменить показатели, отображающиеся в графическом отчете. Поле может содержать одно из значений перечислимого типа [Отчет.ВидыОстОбор](#). В зависимости от выбранной константы в отчете будут выводиться начальные остатки (skBegSaldo), обороты (skTurn) или конечные остатки (skEndSaldo).

В одном объекте [ГрафикОтчета](#) одновременно могут отображаться значения лишь по одному показателю.

Пример

```
ReportChart1.SumKind = Report.skTurn;  
-- выводим обороты
```

Описание

ЧислоСтраниц :Целое;
PageCount :Integer;

Назначение

Поле позволяет узнать число страниц, на которые разделен графический отчет. Количество страниц выбирается системой автоматически, исходя из настроек графика и размера окна.

Пример

```
-- фрагмент кода класса-наследника формы отчета
proc ShowNextPage;
  if ReportInGraphics.CurPage < ReportInGraphics.PageCount then
    ReportInGraphics.CurPage = ReportInGraphics.CurPage + 1;
  end;
end;
```

Описание

ПриНажатии :Строка;
OnClick :String;

Назначение

Поле позволяет узнать и изменить имя процедуры-обработчика события, которое возникает по щелчку кнопки мыши на графике отчета.

Обработчик

```
proc OnClick (Sender :ReportChart; ClickedArea :Chart.ClickedArea; SeriesNo :Integer, ValueNo :Integer);
```

Параметры:

Sender - объект, в котором произошло событие;

ClickedArea - область, в которой произошло событие. Коды областей задаются в перечислимом типе [Chart.ClickedArea](#);

SeriesNo - номер серии, к которой относится элемент графика, на котором произведен щелчок. Номер изменяется от 1 до числа серий;

ValueNo - номер значения, которому соответствует элемент графика, на котором произведен щелчок. Номер изменяется от 1 до числа значений.

Если щелчок был выполнен вне какого-либо рабочего элемента графика, параметры **SeriesNo** и **ValueNo** равны нулю.

Пример

```
-- фрагмент кода класса-наследника формы отчета  
proc OnClick(Sender :ReportChart; ClickedArea :Chart.ClickedArea;  
  SeriesNo :Integer, ValueNo :Integer);  
  if ClickedArea = Chart.Series then  
    Message(Sender.Value[SeriesNo, ValueNo]);  
  end;  
end;
```



НастройкиГрафика / ChartSettings

Класс *НастройкиГрафика / ChartSettings*, производный от родительского класса [Объект / Object](#), используется для указания свойств объектов других классов, обеспечивающих графическое представление данных, таких как [График / Chart](#) и [ГрафикОтчета / ReportChart](#).

Непосредственно в классе *НастройкиГрафика / ChartSettings* определены следующие свойства и методы:

- [Поле ВидСерий / SeriesKind](#)
- [Поле ТрехмерныйВид / View3D](#)
- [Поле ПоказыватьПодсказки / ShowMarks](#)
- [Поле Штриховка / Pattern](#)
- [Поле Перевернутый / Inverted](#)
- [Поле ПоказыватьЛегенду / ShowLegend](#)
- [Поле РазмещениеЛегенды / LegendPos](#)
- [Поле ГоризонтальнаяСетка / HGrid](#)
- [Поле ВертикальнаяСетка / VGrid](#)
- [Поле ВидГистограммы / HystogramKind](#)
- [Поле ФормаГистограммы / HystogramForm](#)
- [Поле СодержаниеПодсказок / MarkContents](#)

При определении свойств графика в классе *НастройкиГрафика / ChartSettings* используются следующие перечислимые типы:

-  [Тип ВидыСерий / SeriesKinds](#)
-  [Тип ВидыГистограммы / HystogramKinds](#)
-  [Тип ФормыГистограммы / HystogramForms](#)
-  [Тип ВидыПодсказок / MarkKinds](#)
-  [Тип ВидыРазмещенияЛегенды / LegendPosKinds](#)

Создать объект класса *НастройкиГрафика* из программы на ТБ.Скрипт нельзя. Объекты данного класса создаются самой системой в контексте объектов других классов и доступны из последних через соответствующее ссылочное поле. Разыменовывая такую ссылку, программист имеет возможность читать и устанавливать настройки графиков.

Константы [ВидыГистограммы](#) / [HystogramKinds](#)

Перечислимый тип **ВидыГистограммы / HystogramKinds** определяет несколько констант, которые задают взаимное расположение столбиков гистограммы.

В данном типе определены следующие константы:

- **ДругЗаДругом / None** - столбики отображаются рядом друг с другом (как бы в плоскости);
- **Рядом / Side** - столбики отображаются друг за другом (как бы в глубину);
- **Соединенные / Stacked** - столбики серий будут объединены в один общий столбец по каждому аргументу;
- **Соединенные100 / Stacked100** - аналогично предыдущему варианту, но столбцы растягиваются на полную высоту (или ширину) графика.

Константы, определенные в данном типе, используются в свойстве [ВидГистограммы / HystogramKind](#).

С помощью перечислимого типа **ВидыПодсказок / MarkKinds** определено несколько констант, задающих суть подсказок, выводимых для элементов графика. Значения данного типа присваиваются свойству [СодержаниеПодсказок/ MarkContents](#).

В данном типе определены следующие константы:

- **Величина / YValue** - значение ;
- **Процент / Percent** - процент от суммы значений в серии;
- **Аргумент / Argument** - аргумент;
- **АргументПроцент / ArgumentPercent** - аргумент и процент от суммы значений в серии;
- **АргументВеличина / ArgumentYValue** - аргумент и значение;
- **ВеличинаАргумент / YValueArgument** - значение и аргумент;
- **ПроцентВсего / PercentTotal** - процент от суммы значений по данной серии, значение самой суммы;
- **АргументПроцентВсего / ArgumentPercentTotal** - аргумент, процент от суммы значений по данной серии, значение самой суммы.

Перечислимый тип **ВидыРазмещенияЛегенды** / **LegendPosKinds**, определенный в классе **НастройкиГрафика**, задает способ размещения легенды на графике.

В данном типе определены следующие константы:

- РазместитьСверху / AtTop;
- РазместитьСнизу / AtBottom;
- РазместитьСправа / AtRight;
- РазместитьСлева / AtLeft.

Указанные константы используются в свойстве [РазмещениеЛегенды / LegendPos](#).

Перечислимый тип **ВидыСерий / SeriesKinds** определяет несколько констант, которые задают внешний вид отображения серий данных. Тип используется в свойстве [ВидСерий](#).

В типе определены следующие константы:

- **ВертикальнаяГистограмма / Bars** - вертикальная гистограмма;
- **ГоризонтальнаяГистограмма / HorizBars** - горизонтальная гистограмма;
- **Заполненная / Areas** - заполненная диаграмма;
- **График / Lines** - линейная диаграмма;
- **Точечная / Points** - точечная диаграмма;
- **Круговая / Pie** - круговая диаграмма;
- **Стрелочная / Arrows** - стрелочная диаграмма;
- **Пузырьковая / Bubbles** - пузырьковая диаграмма.

Перечислимый тип **ФормыГистограммы / HystogramForms** определяет несколько констант, которые задают возможную форму столбиков гистограммы.

В данном типе определены следующие константы:

- Параллелепипед / Rectangle;
- Пирамида / Pyramid;
- ПеревернутаяПирамида / InvertedPyramid;
- Цилиндр / Cylinder;
- Эллипс / Ellipse;
- Стрелка / Arrow;
- ПараллелепипедСПереходомЦвета / GradientRectangle.

Константы, определенные в данном типе, используются в свойстве [ФормаГистограммы / HystogramForm](#).

Описание

ВертикальнаяСетка :Логическое;
VGrid :Logical;

Назначение

Поле позволяет включать/отключать режим отображения вертикальной сетки.

Пример

```
-- используем флаг для отображения/удаления  
-- вертикальной сетки  
proc OnCheckBox2(C1: CheckBox);  
    Chart1.Settings.VGrid = C1.State;  
end;
```

Описание

ВидГистограммы : [НастройкиГрафика.ВидыГистограммы](#)
HystogramKind : [ChartSettings.HystogramKinds](#)

Назначение

Поле позволяет устанавливать один из возможных [видов гистограмм](#) в графике.

Пример

```
proc OnCheckBox(C1: CheckBox);  
  if C1.State then  
    Chart1.HystogramKind = ChartSettings.Stacked;  
  else  
    Chart1.HystogramKind = ChartSettings.Stacked100;  
  end;  
end;
```


Описание

ВидСерий : [НастройкиГрафика.ВидыСерий](#);
SeriesKind : [ChartSettings.SeriesKinds](#);

Назначение

С помощью данного свойства можно узнать и изменить внешний вид серий данных. Поле может принимать одно из предопределенных значений, описанных в типе [НастройкиГрафика.ВидыСерий](#).

Пример

```
-- процедура для переключения вида графика
-- в зависимости от состояния элементов управления
proc OnRadioButtonGroup(R1: RadioButton);
  if R1.Name="Линейная" AND R1.State then
    -- устанавливаем вид серии
    Chart1.Settings.ВидСерий =
      НастройкиГрафика.График;
  elseif R1.Name = "Точечная" AND R1.State then
    -- устанавливаем вид серии
    Chart1.Settings.ВидСерий =
      НастройкиГрафика.Точечная;
  end;
end;
```

Описание

ГоризонтальнаяСетка :Логическое;
HGrid :Logical;

Назначение

Поле позволяет включать/отключать режим отображения горизонтальной сетки.

Пример

```
-- используем флаг для отображения/удаления
-- горизонтальной сетки
proc OnCheckBox2(C1: CheckBox);
    Chart1.Settings.HGrid = C1.State;
end;
```

Описание

Перевернутый :Логическое;
Inverted :Logical;

Назначение

Поле позволяет включать и отключать режим, когда график отображается перевернутым на 180 градусов.

В зависимости от вида серий изображение переворачивается либо по вертикали (верх и низ меняются местами), либо по горизонтали (левый и правый край меняются местами).

Следует иметь в виду, что некоторые виды серий, например, круговая диаграмма, не могут быть перевернуты.

Пример

```
proc F1(Sender:Button);  
    if Chart1.Settings.Inverted then  
        Message("График перевернут.");  
    end;  
end;
```

Описание

ПоказыватьЛегенду :Логическое;
ShowLegend :Logical;

Назначение

Поле определяет, нужно ли показывать выносную область с описанием обозначений (цветов) на графике.

Когда поле равно TRUE, легенда выводится, причем ее положение определяет поле [РазмещениеЛегенды / LegendPos](#).

Пример

```
-- используем флаг для вывода/отключения легенды  
proc OnCheckBox2(C1: CheckBox);  
    Chart1.Settings.ShowLegend = C1.State;  
end;
```

Описание

ПоказыватьПодсказки :Логическое;
ShowMarks :Logical;

Назначение

Поле позволяет узнать, выводятся ли на график подсказки (отметки вдоль координатных осей), а также изменить данный режим. Когда поле равно TRUE, подсказки выводятся.

Содержание подсказок определяется свойством [СодержаниеПодсказок](#).

Пример

```
-- используем флаг для вывода/удаления подсказки
proc OnCheckBox3(C1: CheckBox);
    Chart1.Settings.ShowMarks = C1.State;
end;
```

Описание

РазмещениеЛегенды : [НастройкиГрафика.ВидыРазмещенияЛегенды](#);

LegendPos : [ChartSettings.LegendPosKinds](#);

Назначение

Поле позволяет задать размещение легенды на графике. Возможные положения описываются с помощью констант перечислимого типа [ВидыРазмещенияЛегенды](#).

Видимостью легенды управляет поле [ПоказыватьЛегенду](#).

Пример

```
proc OnCheckBox2(C1: CheckBox);  
  Chart1.Settings.ShowLegend = C1.State;  
  if Chart1.Settings.ShowLegend then  
    Chart1.Settings.LegendPos=ChartSettings.AtRight;  
  end;  
end;
```

Описание

СодержаниеПодсказок [НастройкиГрафика.ВидыПодсказок](#);
MarkContents : [ChartSettings.MarkKinds](#);

Назначение

Позволяет узнать, какого рода информация выводится в подсказках к элементам графика, а также изменить данную установку.

Поле может принимать значение типа [НастройкиГрафика.ВидыПодсказок](#).

Пример

```
Chart1.MarkContents = ChartSettings.YValue;
```

Описание

ТрехмерныйВид :Логическое;
View3D :Logical;

Назначение

Поле позволяет узнать и изменить внешнее представление элементов графика. Когда поле равно FALSE элементы представлены в двумерном виде, иначе (TRUE) - в трехмерном виде.

Пример

```
-- используем флаг для переключения вида графика
proc OnCheckBox1(C1: CheckBox);
    Chart1.Settings.View3D = C1.State;
end;
```


Описание

ФормаГистограммы [НастройкиГрафика.ФормыГистограммы](#);
HystogramForm : [ChartSettings.HystogramForms](#);

Назначение

Поле позволяет устанавливать одну из возможных [форм гистограмм](#) в графике.

Пример

```
proc OnCheckBox(C1: CheckBox);  
  if C1.State then  
    Chart1.HystogramKind = ChartSettings.Pyramid;  
  else  
    Chart1.HystogramKind = ChartSettings.InvertedPyramid;  
  end;  
end;
```

Описание

Штриховка :Логическое;
Pattern :Logical;

Назначение

Поле позволяет определить режим использования штриховки на графике. Когда поле равно TRUE, штриховка выводится.

Пример

```
proc F1(Sender:Button);  
  if Chart1.Settings.Pattern then  
    Message("Используется штриховка.");  
  end;  
end;
```

Класс *Отчет / Report*, производный от класса [Объект](#), используется для работы с внутренними отчетами. Он представляет собой программный интерфейс к встроенным отчетам, которые доступны пользователям посредством команды [Учет|Отчеты \(Alt+Q\)](#) при наличии соответствующих прав.

Более подробные сведения о работе с объектами и свойствами класса *Отчет* изложены в теме [Общие принципы использования свойств класса Отчет](#).

В связи с тем, класс *Отчет* содержит большое количество свойств и методов, [список перечислимых типов](#), приведен в отдельной теме. Свойства и типы, помеченные в списке свойств и типов значком &, доступны не только в классах языка ТБ.Скрипт, но и в языке типовых операций (т.е. на сервере расчетов, а не на клиентском компьютере). Эти свойства относятся к настройке, построению и анализу внутренней структуры отчета, что иногда полезно выполнять непосредственно на сервере расчетов (для оптимизации быстродействия). Остальные свойства, связанные с визуализацией результатов построения отчета, не могут использоваться на сервере.

Непосредственно в классе *Отчет* определены следующие свойства и методы:

-  [Функция Создать / Create](#) &
-  [Функция СоздатьПоИмени / CreateName](#)
-  [Функция СоздатьНовый / CreateNew](#)
-  [Поле Имя / Name](#)
-  [Поле Название / Caption](#)
-  [Поле ТипОтчета / ReportType](#)
-  [Поле Формат / Format](#)
-  [Поле НачальнаяДата / BegDat](#) &
-  [Поле КонечнаяДата / EndDate](#) &
-  [Поле НачальнаяДатаОстатка / SaldoBegDat](#)
-  [Поле ПланСчетов / AccountPlan](#) &
-  [Поле ФильтрСчетов / AccountFilter](#)
-  [Поле ФильтрПараметров / ParameterFilter](#)
-  [Поле ФильтрСправочника / ReferenceFilter](#)
-  [Поле ВключатьУдаленные / IncludeDeleted](#)
-  [Поле ТолькоИспользованные / OnlyUsed](#)
-  [Поле Разбиение / Split](#)
-  [Поле ФорматОстОбор / SaldoTurnFmt](#) &
-  [Поле КоличествоПоказателей / IndicatorCount](#)
-  [Поле Показатель / Indicator](#)
-  [Поле ПоказательПоИмени / IndicatorByName](#)
-  [Функция ДобавитьПоказатель / AddIndicator](#)
-  [Функция ВставитьПоказатель / InsertIndicator](#)
-  [Процедура УдалитьПоказатель / DeleteIndicator](#)
-  [Поле СчетчикПроводок / IncludeTransCount](#) &
-  [Поле ПоказыватьПрочие / ShowBeyondLimits](#)
-  [Поле СклеиваниеТаблиц / LinkTables](#)
-  [Поле ИтоговаяСтрокаПоОтчету / ShowReportTotalRow](#)
-  [Поле ИнтерактивноеУточнение / InteractivePrecision](#)
-  [Поле ПоказыватьДебет / ShowDebet](#)
-  [Поле ПоказыватьКредит / ShowCredit](#)
-  [Поле ПоказыватьПривязку / ShowJurLink](#)
-  [СводныеПроводки / ConsolidatedTrans](#)
-  [СводитьСУчетом / ConsolidateWith](#)
-  [Поле ПриУточненииПроводок / PreciseTransBy](#)
-  [Поле БазовыйКласс / BaseClass](#)
-  [ВыделятьСтрокиПо / StripeRowsBy](#)
-  [Поле ИспользоватьАккумуляторы / UseAccumulators](#)
-  [Поле ИспользоватьИнвСпискиПоПараметрам / UseParamIndexes](#)

-  [Поле ИспользоватьИндСпискиПоСчетам / UseAccsIndexes](#)
-  [Поле ПоказыватьПодсказку / ShowHint](#)
-  [Поле ПодставитьЗаголовок / SubstCaption](#)
-  [Поле БазовыйУточняющего / PreciseBaseClass](#)
-  [Поле ЯвляетсяУточняющим / ServeAsPrecise](#)
-  [Процедура Сохранить / Save](#)
-  [Процедура Обновить / Reinit](#)
-  [Процедура Построить / Build](#) &
-  [Поле КоличествоТаблиц / TableCount](#) &
-  [Поле КоличествоСтрок / RowCount](#) &
-  [Поле КоличествоКолонок / ColumnCount](#) &
-  [Поле ТекущаяТаблица / CurTable](#)
-  [Поле ТекущаяСтрока / CurRow](#)
-  [Поле ТекущаяКолонка / CurColumn](#)
-  [Поле ЗначениеРазбиения / SplitValue](#) &
-  [Поле МеткаРазбиения / SplitTag](#)
-  [Поле ЗначениеПоказателя / IndicatorValue](#)
-  [ФорматЗначенияПоказателя / IndicatorValueFormat](#)
-  [Поле МеткаЗначенияПоказателя / IndicatorValueTag](#)
-  [Функция Оборот / Turn](#) &
-  [Функция НачальныйОстаток / BegSaldo](#) &
-  [Функция КонечныйОстаток / EndSaldo](#) &
-  [Функция КоличествоПроводок / TransCount](#) &
-  [Поле ЭтоГруппа / IsGroup](#)
-  [Процедура ПостроитьГруппу / |BuildGroup](#)
-  [Поле ГруппаОткрыта / GroupIsOpen](#)
-  [Поле МожноВойтиВГруппу / CanEnterGroup](#)
-  [Процедура ВойтиВГруппу / EnterGroup](#)
-  [Процедура ВыйтиИзГруппы / LeaveGroup](#)
-  [ВнутриГруппы / InGroup](#)
-  [Поле Группа / Group](#) &
-  [Поле КоличествоУточняющихОтчетов / PreciseReportCount](#)
-  [Поле УточняющийОтчет / PreciseReport](#)
-  [Функция ВыбратьУточняющийОтчет / SelectPreciseReport](#)
-  [Процедура ПостроитьУточнение / BuildPrecision](#)
-  [Поле УточнениеОткрыто / PrecisionIsOpen](#)
-  [Поле МожноВойтиВУточнение / CanEnterPrecision](#)
-  [Процедура ВойтиВУточнение / EnterPrecision](#)
-  [Процедура ВыйтиИзУточнения / LeavePrecision](#)
-  [Поле ВнутриУточнения / InPrecision](#)
-  [Процедура Удалить / Delete](#)
-  [Процедура ПереупорядочитьПо / ResortBy](#)
-  [Процедура ОграничитьКоличествоСтрок / LimitRowCount](#)

Описание

ВыделятьСтрокиПо : [ТипыВыделенияСтрок](#);
StripeRowsBy : [RowStripeTypes](#);

Назначение

Данное свойство позволяет установить или узнать стиль, используемый для выделения четных и нечетных строк, или стили, принадлежащие различным документам. Значением поля может быть одна из констант **stOdd**, **stDocum** или **stNone** перечислимого типа **ТипыВыделенияСтрок**.

По умолчанию значение данного свойства равно **stOdd** для сохранения работы уже настроенных отчетов по оборотам.

Устанавливая значение свойства StripeRowsBy=stDocum, можно управлять выделением строк в отчетах по оборотам и проводкам при изменении значения параметра первого отсортированного разбиения. Если сортировка не задана, то по значению первого параметра разбиения.

Переключить режимы выделения строк можно, не перестраивая отчет на сервере, с помощью вызова метода формы отчета [Update](#).

Внимание. Для срабатывания данного свойства при выборе значений stOdd, stDocum необходимо в шаблон отчета в таблицу по проводкам добавить строку "ПроводкаЧет", а также в таблицу по оборотам вставить строки "СтрокаЧет_Элемент" и "СтрокаЧет_СвернутаяГруппа" при наличии строки "Строка_СвернутаяГруппа".

Пример

```
var Rep :Report;  
....  
Rep.StripeRowsBy = Report.stDocum;
```

Первоначальная настройка отчетов может осуществляться разработчиком в редакторе проекта с помощью диалогов в ветви "Внутренние отчеты" иерархии проекта. Данный класс не обладает визуальным интерфейсом и лишь предоставляет возможность программно получать и анализировать данные. Разработчик прикладных систем на ТБ.Скрипте может произвольным образом отобразить результирующую информацию в наглядном для пользователя виде.

Отчет представляет собой одну или несколько таблиц (если нет данных, то число таблиц [TableCount](#) равно нулю и попытки обратиться к результатам отчета будут приводить к ошибке на стадии выполнения программы). Каждая таблица состоит из строк, причем каждая строка содержит информацию по одному объекту хозяйственной деятельности (или ресурсу), заданному в настройках отчета с помощью поля [ТипРазбиения](#).

Если нет разбиения на колонки (это задается с помощью константы **None** по измерению **rdCol** для свойства **ТипРазбиения**), то в результирующей структуре данных формируются только служебные колонки с суммами начального сальдо, оборота и конечного сальдо, которые содержат обобщенные показатели по объектам учета, упомянутым в соответствующих строках. Каждая из этих колонок имеет деление на дебетовую и кредитовую составляющие. Для их получения используются функции [НачальныйОстаток](#), [Оборот](#) и [Функция КонечныйОстаток](#) соответственно. При их вызове через первый параметр с помощью констант **ByDeb**, **ByCre** и **Roll** задается составляющая показателя: дебетовая, кредитовая или свернутая форма. Второй аргумент используется для указания измерителя, в котором необходимо получить показатели.

Следует иметь в виду, что прежде чем обращаться к функциям **НачальныйОстаток**, **Оборот** и **КонечныйОстаток** необходимо задать свойства **ТекущаяТаблица**, **ТекущаяСтрока** и **ТекущаяКолонка**, однозначно определяющие, какой именно показатель требуется извлечь из результатов отчета. Установка свойств **ТекущаяТаблица**, **ТекущаяСтрока** и **ТекущаяКолонка** должна производиться именно в указанном порядке. Таблицы, строки и колонки нумеруются от 1. Если какой-либо из этих индексов установить в 0, то станут доступны итоги, соответственно, по таблице, строке или колонке (разумеется, при условии, что было задано разбиение). Общие итоги по всему отчету задаются в свойстве [ИтоговаяСтрокаПоОтчету](#).

Описание

`МожноВойтиВУточнение` :Логическое;
`CanEnterPrecision` :Logical;

Назначение

Свойство используется для интерактивного уточнения и возвращает значение "Истина", если уточнение было [построено](#).

Пример

См. в теме [Пример работы с уточняющими отчетами](#).

Описание

БазовыйКласс :Строка;
BaseClass :String;

Назначение

Данное свойство используется для настройки базового отчета. Оно применимо только для пользовательских отчетов и для отчетов, созданных *не в рамках объекта "Форма отчета"*.

Пример

```
proc UpdateReport(const AReportName :String);  
  var Rep :Report;  
  Rep = Report.CreateName(AReportName);  
  Rep.BaseClass="Валюта";  
  ...  
end;
```


Описание

БазовыйУточняющего :Строка;
PreciseBaseClass :String;

Назначение

Поле позволяет узнать и изменить имя класса отчета (наследника класса **ФормаОтчета**), который будет использоваться в качестве уточняющего для данного отчета.

Поле наследуется производными классами отчетов.

В режиме проектирования значение данного поля задается в диалоге настройки свойств отчета на странице "Формат".

Пример

```
proc P1(Sender :Button);  
  var Rep :Report;  
  Rep = Report.CreateName("ПоОборотам");  
  Rep.БазовыйУточняющего = "ОборотыПоКонтрагентам";  
  Rep.Построить;  
end;
```

Описание

ВключатьУдаленные : Логическое;
IncludeDeleted : Logical;

Назначение

Свойство позволяет включать в отчет удаленные элементы справочника, по умолчанию значение свойства равно False, т.е. в отчет не попадают ранее удаленные элементы справочника.

Внимание. При удалении элементов справочника не происходит их физического удаления, они только помечаются как удаленные (потерянные), что позволяет их, при необходимости, включать в отчет.

Добавление в отчет ранее удаленных элементов справочника происходит при условии, что значение текущего свойства равно True (IncludeDeleted = True), а свойство [ТолькоИспользованные](#) равно False.

Пример

```
var R:Report;  
R = Report.Create("ОтчетПоКонтрагентам");  
R.Build;  
R.OnlyUsed = False;  
R.IncludeDeleted = True;
```

Описание

ВнутриГруппы[Измерение: [ОтчетReport.ИзмеренияОтчета](#)] : Логическое;
InGroup[Dimension : [Report.RepDimension](#)] :Logical;

Аргументы

Измерение - вид измерения, заданный одной из констант перечислимого типа [ИзмеренияОтчета/RepDimension](#). Данный аргумент определяет способ разбиения отчета: по таблицам, строкам, столбцам или интерактивное разбиение.

Назначение

Для заданного измерения свойство позволяет определить, находится ли текущая позиция внутри группы. Свойство используется при наличии в отчетах иерархии по параметрам разбиения.

Поле доступно на только чтение.

Описание

ВнутриУточнения :Логическое;
InPrecision :Logical;

Назначение

Свойство устанавливается в True, если текущая строка таблицы относится к интерактивному уточнению.

Причем, настройки разбиения по строкам после построения отчета могут быть разными, в зависимости от того:

- 1)** - принадлежит, ли текущее положение внутренней структуры отчета уточнению или основному отчету;
- 2)** - какое именно уточнение было выполнено (зависит от строки).

Замечание. Данное свойство введено в программный интерфейс класса [Отчет](#) для выполнения действий внутри обработчика [ПриПодготовкеОтчета](#) при интерактивном уточнении.

Пример

См. в теме [Пример работы с уточняющими отчетами](#).

Описание

Группа [Измерение : [Отчет.ИзмеренияОтчета](#) {; Индекс :Целое}] :Вариант;
Group [Dimension : [Report.RepDimensions](#) {; Index :Integer}] :Variant;

Аргументы

Измерение - [измерение](#), по которому происходит переход в требуемую группу;

Индекс - номер параметра разбиения. В обычной иерархии по умолчанию необязательный параметр равен 1. В иерархии *по параметрам разбиения* он по умолчанию равен номеру параметра, соответствующему уровню текущей группы в иерархии разбиения. Посмотреть остальные значения группы, лежащие выше по иерархии, и которые вместе позволяют однозначно идентифицировать группу, можно устанавливая соответствующий номер.

Назначение

В иерархическом отчете поле позволяет осуществить быстрый переход в требуемую группу по указанному измерению, для чего её идентификатор следует присвоить полю. В зависимости от типа разбиения в присваиваемом значении типа **Вариант** может находиться счет или аналитический признак. Кроме того, полю можно присвоить значение **nil**, в результате чего происходит переход на верхний уровень иерархии иерархического отчета.

Если полю присваивается значение не попавшего в отчет счета или признака (группа хотя и может существовать в иерархии счетов или признаков, но отсутствовать в таблице отчета), генерируется исключительная ситуация.

После установки группы не определены поля **CurRow** и **CurColumn**. Кроме того, если группа является таблицей, также не определено и поле **CurTable**.

Если иерархический отчет строился с разбиением на таблицы по верхнему уровню групповой иерархии, то присваивание данному полю значения **nil** приводит к тому, что все координаты не определены: **CurTable**, **CurRow** и **CurColumn** равны **nil**. Если присваиваемая группа находится в другой таблице, то автоматически будет корректно изменено и поле **CurTable**.

Если отчет строился с разбиением на таблицы, отличным от разбиения по верхнему уровню групповой иерархии, то установка новой группы действует в контексте текущей таблицы. В связи с этим могут возникать ситуации, когда запрашиваемая группа, вообще говоря, упоминается в отчете, но в другой таблице - тогда все равно будет возникать ошибка.

Свойство доступно только на чтение.

Пример

```
var Отч:Отчет;  
Отч = Отчет.СоздатьПоИмени( "ПоОборотам" );  
Отч.Построить;  
-- определяем внутренние обороты  
Отч.Group = Контрагенты.Мы;
```

Описание

ГруппаОткрыта(Измерение : [Отчет.ИзмеренияОтчета](#)) :Логическое;
GroupIsOpen(Dimension : [Report.RepDimensions](#)) :Logical;

Аргументы

Измерение - предопределенная [константа](#), которая определяет вид разбиения отчета: по таблицам, строкам, столбцам или интерактивное разбиение. для которой требуется определить, является ли группа открытой.

Назначение

Свойство используется для интерактивного уточнения иерархии и доступно на чтение и запись. При чтении возвращает значение "Истина", если для заданного измерения группа является открытой, иначе - "Ложь".

При установке делает группу открытой и видимой (форматирует и выводит в шаблон) или закрытой и невидимой.

Замечание 1. Действия будут выполнены, даже если шаблон отчета еще не был построен, и при последующем обновлении все, что было построено и открыто, будет отображаться в шаблоне отчета.

Замечание. Данное свойство введено в программный интерфейс класса [Отчет](#) для выполнения действий внутри обработчика [ПриПодготовкеОтчета](#) при интерактивном уточнении иерархии.

Описание

ЗначениеПоказателя [Индекс :Целое; ОстОбор : [Отчет.ВидыОстОбор](#); ДебКре : [Отчет.ВыводОстОбор](#)] :Вариант;
IndicatorValue [Index :Integer; SumKind : [Report.SumKinds](#); DebCre : [Report.SumDebCre](#)] :Variant;

Аргументы

Индекс - номер типа показателя;

ОстОбор - тип показателя, задаваемый одной из предопределенных констант типа [ВидыОстОбор](#);

ДебКре - одна из предопределенных [констант](#), задающая режим вывода показателей, разделенных по дебету|кредиту или в свернутом виде.

Назначение

Поле позволяет получить или установить значение показателя для заданного номера, типа и режима вывода результатов показателя. Поле может содержать как показатель стандартного вида (целое, число, строка, дата и др.), так и измеритель (числовое значение и единица измерения) или число и пара валют (например, для функций деления).

Поле доступно на чтение и запись.

Пример

```
var локИзмеритель :Unit;  
const аОстОбор :SumKinds;  
const аДебКре :SumDebCre;  
.....  
if аДебКре = Roll then  
  if ЭтоДебетовоеЗначение(локИзмеритель) then  
    Отчет.IndicatorValue[аНомерПоказателя, аОстОбор, ByDeb] = локИзмеритель;  
    Отчет.IndicatorValue[аНомерПоказателя, аОстОбор, ByCre] = nil as Unit;  
  else  
    ....  
  end;  
end;  
end;
```

Описание

ЗначениеРазбиения [(Измерение : [Отчет.ИзмеренияОтчета](#) {;Индекс :Integer})] :Вариант;
SplitValue [(Dimension : [Report.RepDimensions](#) {;Index :Integer})] :Variant;

Аргументы

Измерение - измерение, по которому требуется получить значение параметра разбиения;

Индекс - номер параметра разбиения. В обычной иерархии по умолчанию необязательный параметр равен 1. Для иерархии по параметрам разбиения необязательный параметр по умолчанию устанавливается равным номеру параметра, соответствующему текущему уровню иерархии разбиения.

Назначение

Поле позволяет получить значение разбиения для заданного измерения и индекса. Поле может использоваться как для обычной иерархии (Индекс=1), так и для иерархии по параметрам разбиения.

В зависимости от типа разбиения, значением может быть число, измеритель, идентификатор счета или аналитического признака, строка или дата. Например, при разбиении на таблицы по датам через данное свойство можно получить дату, которой соответствует текущая таблица.

Пример

```
var R      :Report;  
var локСтавкаНДС :String;  
....  
R = Report.CreateName('БазовыйСПараметрами', Report.ByTurn);  
R.BegDate      = 1.1.1900;  
R.EndDate      = ДатаКон;  
R.AccountPlan  = "Технологические";  
R.AccountFilter = "НДС";  
R.Split[Report.RdTab].SplitType = Report.ByParam;  
R.InsertIndicator(2,"Сумма");  
локСтавкаНДС = R.SplitValue[Report.rdRow,2];
```


Описание

Имя :Строка;
Name :String;

Назначение

Позволяет получить или установить имя отчета.

Пример

```
proc CreateNewReport(const AReportName, ABaseClassName :String; AShared :Logical);
    var Rep :Report;

    Rep = Report.CreateNew(AReportName, Report.byTurn, AShared);
    Rep.BaseClass = ABaseClassName;
    Rep.Имя = 'Отчет по проводкам';
    Rep.AccountPlan = 'Склад';
    ....
    Rep.Save;
end;
```

Описание

ИнтерактивноеУточнение :Логический;
InteractivePrecision :Logical;

Назначение

В отчетах по оборотам данное свойство позволяет программным способом выполнить интерактивное уточнение. При значении, равным True, в таблице уточняемого отчета появляются новые строки с уточняющими результатами, при значении False уточняющий отчет строится в отдельном окне, а не в уже существующей таблице отчета.

Поле доступно на чтение и запись. Действие данного свойства аналогично флагу **Интерактивное уточнение** на странице [Дополнительно](#) диалога "Внутренние отчеты". Значение True соответствует включенному флагу.

Пример

```
var Rep :Report;  
  
proc Fl(Sender:Button);  
    if Rep.InteractivePrecision then  
        Message("Уточнение производится непосредственно в отчете.");  
    end;  
end;
```

Описание

ИспользоватьАккумуляторы :Логический;
UseAccumulators :Logical;

Назначение

Позволяет программно установить режим построения отчетов при котором разрешается использовать аккумуляторы. Если значение поля равно "Истина", то аккумуляторы разрешается использовать, что соответствует установке флага **"Использовать аккумуляторы при построении отчетов"** на странице "Отчеты" раздела "Журналы и отчеты" диалога "Настройка учета", который открывается командой **Сервис | Настройки программы**. По умолчанию флаг выключен.

Пример

```
var Rep :Report;  
  
proc Fl(Sender:Button);  
  if Rep.UseAccumulators then  
    Message("Аккумуляторы разрешается использовать.");  
  end;  
end;
```

Описание

ИспользоватьИнвСпискиПоПараметрам :Логический;
UseParamIndexes :Logical;

Назначение

При построении отчета свойство позволяет программно отключать использование инвертированных списков (False) при отборе проводок по параметрам. Если свойство равно True, инвертированные списки используются.

Поле доступно на чтение и запись.

Пример

```
var Rep :Report;  
  
proc F1(Sender:Button);  
    if Not Rep.UseParamIndexes then  
        Message("Инвертированные списки по параметрам не используются.");  
    end;  
    ...  
end;
```

Описание

ИспользоватьИнвСпискиПоСчетам :Логический;
UseAccsIndexes :Logical;

Назначение

При построении отчета свойство позволяет программно управлять использованием инвертированных списков при отборе проводок по счетам. Если свойство равно True, инвертированные списки используются, иначе - отключаются (False).

Поле доступно на чтение и запись.

Пример

```
proc Fl(Sender:Button);  
  if Rep.UseAccsIndexes then  
    Message("Инвертированные списки по счетам используются.");  
  end;  
  ...  
end;
```

Описание

ИтоговаяСтрокаПоОтчету :Логическое;
ShowReportTotalRow :Logical;

Назначение

Данное свойство позволяет в отчетах по оборотам со [склеенными таблицами](#) (при отсутствии итоговой таблицы) отображать (свойство равно True) итоговую строку по отчету ("Итого по отчету").

Причем, вывод итоговой строки по отчету не зависит от состояния флага **Итоговая строка по отчету**, расположенного в группе флагов **Прочие настройки** на странице ["Дополнительно"](#) диалога "Внутренние отчеты".

Пример

```
var Rep :Report;  
....  
-- разрешен вывод итоговой строки  
Rep.ShowReportTotalRow = true;
```

Описание

КоличествоКолонок :Целое;

ColumnCount :Integer;

Назначение

Возвращает число колонок в отчете, если он строился с разбиением по колонкам. В противном случае генерируется исключительная ситуация.

Внимание! Перед обращением к данному полю необходимо построить отчет с помощью процедуры [Построить / Build](#).

Пример

Описание

КоличествоПоказателей : Целое;
IndicatorCount : Integer;

Назначение

Позволяет определить или изменить количество показателей, которые перечислены в поле **Имя** на [странице "Показатели"](#) диалога "Внутренние отчеты".

Поле доступно на чтение и запись.

Пример

Описание

КоличествоСтрок :Целое;
RowCount :Integer;

Назначение

Возвращает количество строк в текущей таблице. В иерархическом отчете возвращает число строк в текущей открытой группе.

В случае склеенных таблиц при отсутствии итоговой таблицы, но при наличии итоговой строки по отчету, свойство возвращает значение 0.

Внимание! Перед обращением к данному полю необходимо построить отчет с помощью процедуры [Построить / Build](#).

Описание

`КоличествоТаблиц` :Целое;

`TableCount` :Integer;

Назначение

Возвращает количество таблиц в отчете.

Внимание! Перед обращением к данному полю необходимо построить отчет с помощью процедуры [Построить / Build](#).

Пример

Описание

`КоличествоУточняющихОтчетов` : Целое;
`PreciseReportCount` : Integer;

Назначение

Свойство используется для интерактивного уточнения и выдает количество отчетов в списке отчетов, заданных для интерактивного уточнения текущей строки.

Замечание. Данное свойство введено в программный интерфейс класса [Отчет](#) для выполнения действий внутри обработчика [ПриПодготовкеОтчета](#) при интерактивном уточнении.

Пример

См. в теме [Пример работы с уточняющими отчетами](#).

Описание

КонечнаяДата : Дата;
EndDate : Date;

Назначение

Позволяет получить и установить конечную дату отчетного периода.

Пример

```
proc UpdateReport(const AReportName :String);  
  var Rep :Report;  
  Rep = Report.CreateName(AReportName);  
  Rep.AccountPlan = 'Склад';  
  Rep.НачальнаяДата=01.01.2006;  
  Rep.КонечнаяДата=01.04.2006;  
  Rep.Save;  
end;
```

Описание

МеткаЗначенияПоказателя (Индекс :Целое; ОстОбор :Отчет.ВидыОстОбор;
ДебКре :Отчет.ВыводОстОбор] :Вариант;
IndicatorValueTag (Index :Integer; SumKind :SumKinds; DebCre :SumDebCre) :Variant;

Аргументы

Индекс - номер типа показателя;

ОстОбор - тип показателя, задаваемый одной из предопределенных констант типа [ВидыОстОбор](#);

ДебКре - одна из предопределенных [констант](#), задающая режим вывода показателей, разделенных по дебету|кредиту или в свернутом виде.

Назначение

Данное свойство позволяет с каждым значением показателя ассоциировать значение произвольного типа.

Пример

Описание

МеткаРазбиения (Измерение : [ИзмеренияОтчета](#)) :Вариант;
SplitTag (Dimension :RepDimensions) :Variant;

Аргументы

Измерение - измерение, по которому требуется получить значение параметра разбиения.

Назначение

Свойство позволяет установить метку (произвольные данные) соответственно для текущей таблицы, строки или колонки отчета. В частности, свойство предполагается использовать для хранения информации о цвете строки отчета.

Поле доступно на чтение и запись.

Описание

МожноВойтиВГруппу (Измерение :[Отчет.ИзмеренияОтчета](#)) :Логическое;
CanEnterGroup (Dimension :RepDimensions) :Logical;

Аргументы

Измерение - [измерение](#), по которому требуется определить, можно ли войти в группу.

Назначение

Возвращает значение "Истина", если группа была [построена](#), (есть данные), при этом она может быть и свернутой.

Замечание 2. Данное свойство введено в программный интерфейс класса [Отчет](#) для выполнения действий внутри обработчика [ПриПодготовкеОтчета](#) при интерактивном раскрытии иерархии.

Пример

Описание

Название : Строка;
Caption : String;

Назначение

Позволяет получить и установить заголовок отчета. Если отчет создавался на базе некоторого предварительно настроенного шаблона из иерархии объектов проекта, то его название изначально (после создания объекта) соответствует той строке, которая была введена в поле **Название** диалога настройки отчета. В противном случае название - пустая строка.

Пример

```
proc CreateNewReport(const AReportName, ABaseClassName :String; AShared :Logical);  
    var Rep :Report;  
  
    Rep = Report.CreateNew(AReportName, Report.byTurn, AShared);  
    Rep.BaseClass = ABaseClassName;  
    Rep.Caption = 'Caption';  
    Rep.AccountPlan = 'Склад';  
    ....  
    Rep.Save;  
end;
```


Описание

НачальнаяДата : Дата;
BegDate : Date;

Назначение

Позволяет получить и установить начальную дату отчетного периода.

Пример

```
proc UpdateReport(const AReportName :String);  
  var Rep :Report;  
  Rep = Report.CreateName(AReportName);  
  Rep.AccountPlan = 'Склад';  
  Rep.НачальнаяДата=01.01.2006;  
  Rep.КонечнаяДата=01.04.2006;  
  Rep.Save;  
end;
```

Описание

НачальнаяДатаОстатка : Дата;
SaldoBegDate : Date;

Назначение

Позволяет получить и установить начальную дату остатка. Свойство используется в отчетах по проводкам и оборотам, когда при построении отчета требуется указывать, с какой даты считать начальное сальдо. Например, чтобы построить налоговые отчеты, в которых не нужно учитывать входящее сальдо на начало года.

Пример

```
proc UpdateReport(const AReportName :String);  
  var Rep :Report;  
  Rep = Report.CreateName(AReportName);  
  Rep.AccountPlan = 'Склад';  
  Rep.НачальнаяДатаОстатка=01.01.2006;  
  Rep.Save;  
end;
```

Описание

ФильтрПараметров : Строка;
ParameterFilter : String;

Назначение

Позволяет считать или установить фильтр (строковое условие отбора проводок для отчета по параметрам, например, аналитическим признакам, валютам, единицам измерения).

Пример

```
proc UpdateReport(const AReportName :String);  
  var Rep :Report;  
  Rep = Report.CreateName(AReportName);  
  Rep.AccountPlan = 'Склад';  
  Rep.НачальнаяДата=01.01.2006;  
  Rep.КонечнаяДата=01.04.2006;  
  Rep.ParameterFilter"";  
  Rep.Save;  
end;
```

Описание

ПланСчетов : Строка;
AccountPlan : String;

Назначение

Позволяет узнать и изменить план счетов для отчета.

Если необходимо получить проводки по всем планам счетов, в данное поле следует записать пустую строку.

Пример

```
proc UpdateReport(const AReportName :String);  
  var Rep :Report;  
  
  Rep = Report.CreateName(AReportName);  
  Rep.AccountPlan = 'Склад';  
  ....  
  Rep.Save;  
end;
```

Описание

ПодставитьЗаголовок[Элемент : [Отчет.ЭлементыПодстановки](#)] :Строка;
SubstCaption[Item : [Report.SubstItems](#)] :String;

Аргументы

Элемент - одна из констант перечислимого типа [ЭлементыПодстановки](#), которая определяет в отчетах по оборотам тип элемента (итоговая таблица, итоговая колонка, аналитика), заголовок которого требуется изменить или узнать.

Назначение

Позволяет узнать и программно изменить в отчетах по оборотам заголовок элемента, заданного в первом параметре.

Поле доступно на чтение и запись. Пустая строка в качестве значения поля означает значение по умолчанию.

Пример

```
var Rep :Report;  
....  
Rep.SubstCaption[Report.sbColTotals] = "Всего";
```

Описание

Показатель (Индекс: Целое) : ПоказательОтчета;
Indicator (Index: Integer) :ReportIndicator;

Аргументы

Индекс - индекс показателя, который изменяется от 1 до общего числа показателей в отчете, указанных в поле **Имя** на [странице "Показатели"](#) диалога "Внутренние отчеты". Общее число показателей определяется с помощью свойства [КоличествоПоказателей / IndicatorCount](#).

Назначение

Позволяет по заданному индексу определить ссылку на показатель. В случае отсутствия показателя возвращается пустая ссылка Nil.

Пример

Описание

```
ПоказательПоИмени (Имя : Строка) : ПоказательОтчета;  
IndicatorByName (Name :String) :ReportIndicator;
```

Аргументы

Имя - имя показателя, которое может быть указано в поле **Имя** на [странице "Показатели"](#) диалога "Внутренние отчеты".

Назначение

Возвращается ссылку на показатель из класса **ПоказательОтчета** по его имени, если он существует, иначе - nil.

Пример

Описание

ПоказыватьДебет :Логический;

ShowDebet :Logical;

Назначение

Поле управляет видимостью колонки с дебетом в отчетах по проводкам.

По умолчанию значение поля считается равным TRUE, что соответствует установке флага **Дебет проводки** на странице [Дополнительно](#) диалога "Внутренние отчеты".

Пример

Описание

ПоказыватьКредит :Логический;
ShowCredit :Logical;

Назначение

Позволяет определить или изменить состояние флага **Кредит проводки** на странице [Дополнительно](#) диалога "Внутренние отчеты".

При установленном флаге (значение поля "Истина")отчетах по проводкам будут показываться колонки с кредитом.

Пример

Описание

ПоказыватьПодсказку :Логический;
ShowHint :Logical;

Назначение

Поле позволяет включать/отключать вывод подсказок в строке состояния при построении отчета. Если значение поля равно TRUE, подсказки выводятся, в противном случае - нет.

Пример

```
var Отч:Отчет;  
Отч = Отчет.СоздатьПоИмени( "ПоПроводкам" );  
Отч.ПоказыватьПодсказку = false;
```

Описание

ПоказыватьПривязку :Логический;
ShowJurLink :Logical;

Назначение

Поле управляет видимостью колонки с указанием привязки к журналу (название журнала и номер проводки/типовой операции в нем), поясняющей источник проводки в отчете по проводкам.

По умолчанию значение поля считается равным TRUE.

Пример

```
var Отч:Отчет;  
Отч = Отчет.СоздатьПоИмени( "ПоПроводкам" );  
Отч.ПоказыватьПривязку = false;
```

Описание

ПоказыватьПрочие :Логический;
ShowBeyondLimits :Logical;

Назначение

Поле позволяет включать/отключать строку *Прочие* в отчетах по оборотам. В строку *Прочие* выводятся суммы, которые не удовлетворяют условиям на ограничения по остаткам и оборотам.

Если значение поля равно False, то строка не отображается, что соответствует выключенному флагу **Прочие** на странице [Дополнительно](#) диалога "Внутренние отчеты".

Пример

```
var Отч:Отчет;  
Отч = Отчет.СоздатьПоИмени( "ПоОборотам" );  
Отч.ПоказыватьПрочие = false;
```

Описание

ПриУточненииПроводок : [Отчет.ТипыУточненияПроводок](#);
PreciseTransBy : [Report.TransPreciseTypes](#);

Назначение

Поле позволяет узнать и изменить способ интерактивного уточнения показателей в результатах отчета: либо с помощью документа, породившего проводку, либо с помощью журнала проводок.

Изменение данного свойства возможно как до, так и после построения отчета.

Пример

```
var Отч:Отчет;  
Отч = Отчет.Создать( "ПоОборотам" );  
Отч.Построить;  
Отч.ПриУточненииПроводок = Отчет.ВуOpenJur;
```

Описание

Разбиение [Измерение :ИзмеренияОтчета]: РазбиениеОтчета;
Split [Dimension:RepDimensions] :ReportSplit;

Аргументы

Измерение - [измерение отчета](#), т.е. способ расположения показателей в построенных отчетах (по таблицам, строкам, столбцам).

Назначение

Возвращает ссылку на объект из класса **РазбиениеОтчета / ReportSplit** для заданного вида измерения отчета.

Поле доступно только на чтение.

Следует отметить, что отчет по проводкам (разбиение на строки по проводкам) не имеет внутренней структуры данных, доступной из ТБ.Скрипт, хотя его и можно построить. Все попытки получить доступ к результирующим таблицам построенного отчета по проводкам будут завершаться ошибками.

Пример

```
proc UpdateReport(const AReportName :String);
var Rep :Report;
Rep = Report.CreateName("ПоПродажам");
-- отчет с таблицами
Rep.Save;
....

if Rep.Разбиение[Report.rdTab]<>nil:
    Message("В отчете используется разбиение по таблицам");
end;
....
end;
```

Описание

СклеиваниеТаблиц: Логическое;
LinkTables: Logical;

Назначение

Позволяет выводить таблицы в склеенном виде. Однако, присвоение полю значения True не гарантирует появления склеенных таблиц.

Внимание! Для склеивания таблиц должны быть выполнены и другие условия, например, такие как:

- наличие разбиения на таблицы при отсутствии иерархии по таблицам;
- включение флага **Таблицы по верхнему уровню** в иерархии по строкам на странице Иерархия диалога "Внутренние отчеты";
- в отчете должно быть более одной таблицы.

Пример

```
proc Fl(Sender:Button);  
  if not Rep.LinkTables then  
    Message("В отчете нет склеенных таблиц.");  
  end;  
end;
```

Описание

СчетчикПроводок :Логический;
IncludeTransCount :Logical;

Назначение

Определяет, нужно ли включать счетчик проводок при построении отчета и выводить их количество в результаты отчета.

Пример

```
var Rep :Report;  
Rep = Report.CreateName("ПоОборотам");  
Rep.СчетчикПроводок = true;
```


Описание

ТекущаяКолонка :Целое;
CurColumn :Integer;

Назначение

Позволяет узнать и изменить номер текущей колонки (в текущей таблице) в результатах отчета, из которой извлекаются показатели. Нумерация производится, начиная с 1. Если номер текущей колонки задан равным нулю, то функции получения показателей ([Оборот](#), [НачальныйОстаток](#), [КонечныйОстаток](#), [КоличествоПроводок](#)) возвращают итоги по всей колонке. Если и текущая строка, и текущая колонка равны нулю - получаются итоги по таблице.

Данное свойство следует устанавливать только после установки полей [ТекущаяТаблица](#) и [ТекущаяСтрока](#).

Пример

```
var i: Integer;  
var sum: Numeric;  
Отч = Отчет.СоздатьПоИмени("ПоОборотам");  
Отч.Построить;  
sum = 0;  
try  
    Отч.ТекущаяТаблица = 1;  
    Отч.ТекущаяСтрока = 1;  
    if Отч.ГруппаОткрыта then  
        Отч.ВойтиВГруппу;  
        for i=1..Отч.ЧислоСтрок do  
            Отч.ТекущаяСтрока = i;  
            Отч.ТекущаяКолонка = 1;  
            sum = sum + Отч.Оборот(Отчет.ВыводОстОбор.Roll,1);  
            ...  
        end;  
    end;  
    ...  
except  
    ...  
end;
```

Описание

ТекущаяСтрока : Целое;
CurRow : Integer;

Назначение

Позволяет узнать и изменить номер текущей строки (в текущей таблице) в результатах отчета, из которой извлекаются показатели. Нумерация производится, начиная с 1. Если номер текущей строки задан равным нулю, то функции получения показателей ([Оборот](#), [НачальныйОстаток](#), [КонечныйОстаток](#), [КоличествоПроводок](#)) возвращают итоги по всей строке. Если и текущая строка, и текущая колонка равны нулю - получаются итоги по таблице.

Данное свойство следует устанавливать только после установки поля [ТекущаяТаблица](#).

Пример

```
var i: Integer;  
var sum: Numeric;  
Отч = Отчет.СоздатьПоИмени("ПоОборотам");  
Отч.Построить;  
sum = 0;  
try  
    Отч.ТекущаяТаблица = 1;  
    Отч.ТекущаяСтрока = 1;  
    if Отч.ГруппаОткрыта then  
        Отч.ВойтиВГруппу;  
        for i=1..Отч.ЧислоСтрок do  
            Отч.ТекущаяСтрока = i;  
            Отч.ТекущаяКолонка = 1;  
            sum = sum + Отч.Оборот(Отчет.ВыводОстОбор.Roll,1);  
            ...  
        end;  
    end;  
    ...  
except  
    ...  
end;
```

Описание

ТекущаяТаблица :Целое;

CurTable :Integer;

Назначение

Позволяет узнать и изменить номер текущей таблицы в результатах отчета, из которой извлекаются показатели. Нумерация производится, начиная с 1. Если номер текущей таблицы задан равным нулю и номера строки и столбца тоже равны нулю, то функции получения показателей ([Оборот](#), [НачальныйОстаток](#), [КонечныйОстаток](#), [КоличествоПроводок](#)) возвращают итоги по всему отчету.

Внимание. При установке Report.CurTable = 0 и при отсутствии итоговой таблицы и неслинкованных таблицах выдается сообщение об ошибке. При отсутствии итоговой таблицы, но при слинкованных таблицах, шкала по колонкам одна для всех таблиц, и можно работать с итоговой строкой по отчету и разбиением на столбцы.

Пример

```
var i: Integer;
var sum: Numeric;
Отч = Отчет.СоздатьПоИмени( "ПоОборотам" );
Отч.Построить;
sum = 0;
try
    Отч.ТекущаяТаблица = 1;
    Отч.ТекущаяСтрока = 1;
    if Отч.ГруппаОткрыта then
        Отч.ВойтиВГрупппу;
        for i=1..Отч.ЧислоСтрок do
            Отч.ТекущаяСтрока = i;
            Отч.ТекущаяКолонка = 1;
            sum = sum + Отч.Оборот(Отчет.ВыводОстОбор.Roll,1);
            ...
        end;
    end;
    ...
except
    ...
end;
```

Описание

ТипОтчета : [Отчет.ТипыОтчета](#)

ReportType : [Report.RepTypes](#)

Назначение

Позволяет узнать тип требуемого отчета. В программе можно создать отчеты следующих типов по проводкам, по остаткам и оборотам, по справочнику или неопределенного типа.

Поле доступно только на чтение. Тип отчета задается при [создании](#) нового пользовательского отчета программно.

Пример

```
Rep :Report;
....
proc CreateNewReport(const AReportName,
    ABaseClassName :String; AShared :Logical);

    Rep = Report.CreateNew(AReportName,
        Report.byTurn, AShared);
    Rep.BaseClass = ABaseClassName;
    Rep.Caption = 'Caption';
    Rep.AccountPlan = 'Склад';
    ....
    Rep.Save;
end;
.....
Rep.ReportType = Report.byTurn;
-- отчет по остаткам и оборотам.
```

Описание

ТолькоИспользованные : Логическое;
OnlyUsed : Logical;

Назначение

Позволяет узнать и изменить режим отбора элементов справочника в отчет; действует только для отчета по справочнику. Данное поле аналогично полю [ТолькоИспользованные](#) из класса ЗапросАналитики.

Пример

```
proc P1(Sender :Button);  
  var R : Report;  
  R = Report.Create(byReference);  
  -- отчет по справочнику  
  R.ФильтрСправочника="Скидка > 10";  
  R.OnlyUsed=true;  
  
end;
```

Описание

УточнениеОткрыто :Логическое;
PrecisionIsOpen :Logical;

Назначение

Свойство используется для интерактивного уточнения и доступно на чтение и запись. При запросе свойство выдает значение True, если уточнение открыто, иначе - False.

При установке значения поля в True делает уточнение открытым и видимым (форматирует и выводит в шаблон). В этом случае уточнение сразу же отображается в построенном отчете. При установке в False уточнение закрывается и не показывается в шаблоне.

Замечание 1. Действия будут выполнены, даже если шаблон отчета еще не был построен, и при последующем обновлении все, что было построено и открыто, будет отображаться в шаблоне отчета. Так например, можно построить уточнения всех строк еще до появления отчета на экране.

Замечание 2. Данное свойство введено в программный интерфейс класса [Отчет](#) для выполнения действий внутри обработчика [ПриПодготовкеОтчета](#) при интерактивном уточнении.

Пример

См. в теме [Пример работы с уточняющими отчетами](#).

Описание

УточняющийОтчет (Индекс:Целое) :Строка;
PreciseReport (Index :Integer) :String;

Аргументы

Индекс - номер уточняющего отчета в списке отчетов.

Назначение

Выдает имя отчета по индексу из списка отчетов, заданных для интерактивного уточнения. Количество уточняющих отчетов вычисляется с помощью свойства [PreciseReportCount](#).

Замечание. Данное свойство введено в программный интерфейс класса [Отчет](#) для выполнения действий внутри обработчика [ПриПодготовкеОтчета](#) при интерактивном уточнении.

Пример

См. в теме [Пример работы с уточняющими отчетами](#).

Описание

ФильтрСправочника : Строка;
ReferenceFilter : String;

Назначение

Позволяет узнать и установить фильтр в отчете по оборотам с разбиением по справочнику. Фильтр накладывается на те элементы, которые попадают в отчет из справочника. Целесообразно фильтр и условие на параметры отчета задавать так, чтобы они были согласованными.

Пример

```
proc P1(Sender :Button);  
  var R : Report;  
  R = Report.Create(byReference);  
  -- отчет по справочнику  
  R.ФильтрСправочника="Скидка > 10";  
end;
```


Описание

ФильтрСчетов : Строка;
AccountFilter : String;

Назначение

Позволяет получить и установить условие отбора проводок по счетам для отчета, которое не должно противоречить условию отбора, заданному в режиме проектирования. Аналогичный фильтр можно задать в поле **Счета** диалога ["Внутренние отчеты"](#).

Пример

```
proc UpdateReport(const AReportName :String);  
  var Rep :Report;  
  Rep = Report.CreateName(AReportName);  
  Rep.AccountPlan = 'Склад';  
  Rep.НачальнаяДата=01.01.2006;  
  Rep.КонечнаяДата=01.04.2006;  
  Rep.ФильтрСчетов="+50/51";  
  
  Rep.Save;  
end;
```

Описание

Формат : [Отчет.ФорматыОтчета](#);

Format : [Report.RepFormats](#);

Назначение

Позволяет узнать формат отчета. Программным способом внутренние отчеты можно создать в различных форматах, который задается константами перечислимого типа [ФорматыОтчета](#).

Поле доступно только на чтение.

Пример

```
proc F1(Sender:Button);  
  if Rep.Format=Report.fmtTemplate then  
    Message("Отчет построен с помощью шаблона");  
  end;  
end;
```

Описание

ФорматОстОбор [ОстОбор : [ВидыОстОбор](#)] : [Отчет.ФорматыОстОбор](#);
SaldoTurnFmt [SumKind : [Report.SumKinds](#)] : [Report.SumFormats](#);

Аргументы

ОстОбор - тип показателя, задаваемый одной из predefined констант типа [ВидыОстОбор](#).

Назначение

С помощью данного поля для каждого типа показателей, указанного в параметре **ОстОбор** устанавливается формат вывода, задаваемый одной из predefined констант типа [ФорматыОстОбор / SumFormats](#).

Пример

```
var Отч:Отчет;  
Отч = Отчет.СоздатьПоИмени("ПоОборотам");  
-- устанавливаем формат "обороты в свернутом виде"  
Отч.ФорматОстОбор[Отчет.skTurn] = Отчет.sfRoll;
```

Описание

ЭтоГруппа(Измерение : [Отчет.ИзмеренияОтчета](#)) :Логическое;
IsGroup(Dimension : [Report.RepDimensions](#)) :Logical;

Аргументы

Измерение - [измерение](#), по которому требуется определить, является ли текущий элемент группой.

Назначение

Свойство используется при построении иерархических отчетов для выполнения действий внутри обработчика в случае интерактивного раскрытия. Оно позволяет определить, является ли текущий элемент в указанном измерении (строка или таблица) группой объектов.

Свойство возвращает значение "Истина" для группы объектов, независимо от того, является ли группа открытой или свернутой, и "Ложь" для негрупповых элементов.

Замечание. Данное свойство введено в программный интерфейс класса [Отчет](#) для выполнения действий внутри обработчика [ПриПодготовкеОтчета](#).

Описание

ЯвляетсяУточняющим :Логический;
ServeAsPrecise :Logical;

Назначение

Свойство определяет, что отчет вызван как уточняющий, и доступно только на чтение.

Для уточняющих отчетов, построенных по умолчанию и хранящихся в папке "Уточняющие отчеты" поле **ЯвляетсяУточняющим** всегда равно True, независимо от способа вызова.

Если настройки отчета, построенного как уточняющий, были программно изменены [процедурой Reinit](#), то происходит обновление опций отчета, но поверх них накладываются условия уточнения.

Пример

```
proc UpdateReport(const AReportName :String);
  var Rep :Report;  -- объект из класса Отчет

  Rep = Report.CreateName(AReportName);
  Rep.Save;
  if Rep.ServeAsPrecise then
    Message("Отчет является уточняющим");
  end;
  ....
end;
```

Описание

```
ВойтиВГруппу (Измерение :Отчет.ИзмеренияОтчета);  
EnterGroup (Dimension :RepDimensions);
```

Аргументы

Измерение - [измерение](#), по которому требуется открыть текущий групповой элемент (строку или таблицу).

Назначение

Процедура осуществляет вход в группу (раскрывает ее) в результатах иерархического отчета. Текущий элемент (группа), который раскрывается, может относиться к одному из двух измерений отчета: строка или таблица (измерение столбцов не поддерживает иерархию). Перед вызовом процедуры необходимо присвоить корректные значения полям, задающим координаты по измерениям (например, **CurTable** и **CurRow**), так, чтобы текущий элемент (в данном случае – строка) был группой. Также перед вызовом имеет смысл проверить значение поля **ГруппаОткрыта**, поскольку даже для группового элемента раскрытие может быть невозможно, если уже достигнута максимальная для отчета глубина иерархии.

Выполнение процедуры эквивалентно замене текущей таблицы отчета на подтаблицу с элементами указанной группы.

После выполнения процедуры **ВойтиВГруппу** поля **CurRow** и **CurColumn** не определены. Если был выполнен вход в группу по измерению таблиц, то также не определено и поле **CurTable**.

Для навигации по группе необходимо использовать поля **CurRow** и **CurColumn**, что полностью аналогично перемещению по обычному отчету. Если группа является таблицей, то также необходимо установить поле **CurTable**, так как после входа в таблицу её номер сбрасывается.

Для возвращения на один уровень вверх (в родительскую группу/таблицу) необходимо выполнить процедуру [ВыйтиИзГруппы](#).

Альтернативный способ открыть группу заключается в том, чтобы присвоить идентификатор группы полю [Группа](#).

Замечание 2. Данное свойство введено в программный интерфейс класса [Отчет](#) для выполнения действий внутри обработчика [ПриПодготовкеОтчета](#) при интерактивном раскрытии иерархии.

Пример

```
var i: Integer;  
var sum: Numeric;  
Отч = Отчет.СоздатьПоИмени( "ПоОборотам" );  
Отч.Построить;  
sum = 0;  
try  
  Отч.ТекущаяТаблица = 1;  
  Отч.ТекущаяСтрока = 1;  
  if Отч.ГруппаОткрыта then  
    Отч.ВойтиВГруппу;  
    for i=1..Отч.ЧислоСтрок do  
      Отч.ТекущаяСтрока = i;  
      Отч.ТекущаяКолонка = 1;  
      sum = sum + Отч.Оборот(Отчет.ВыводОстОбор.Roll,1);  
      ...  
    end;  
  end;  
  ...  
except  
  ...  
end;
```

Описание

```
ВойтиВУточнение;  
EnterPrecision;
```

Назначение

Процедура осуществляет вход в уточнение, т.е. обеспечивает доступ к его данным. Если при этом была вызвана процедура [УточнениеОткрыто](#), то уточнение сразу же отображается в построенном отчете.

Для выхода из уточнения необходимо выполнить процедуру [ВыйтиИзУточнения](#).

Замечание. Данное свойство введено в программный интерфейс класса [Отчет](#) для выполнения действий внутри обработчика [ПриПодготовкеОтчета](#) при интерактивном уточнении.

Пример

См. в теме [Пример работы с уточняющими отчетами](#).

Описание

ВыйтиИзГруппы (Измерение : [Отчет.ИзмеренияОтчета](#));
LeaveGroup (Dimension :RepDimensions);

Аргументы

Измерение - [измерение](#), по которому требуется закрыть текущий групповой элемент, т.е. выйти из текущей строки или таблицы.

Назначение

Процедура осуществляет выход из группы (на предыдущий уровень иерархии) в результатах иерархического отчета. Измерение, по которому осуществляется выход, задается параметром **Измерение**.

Если закрытие группы производилось по измерению строк, то после выполнения процедуры поле **CurRow** получает такое же значение, какое оно имело до входа в группу, то есть текущим элементом на более высоком уровне иерархии становится строка с консолидированными данными той группы, которая была закрыта.

Если закрытие группы производилось по измерению таблиц, то аналогичным образом восстанавливается значение поля **CurTable** (текущей становится таблица, из которой только что сделан выход).

Поле **CurColumn** после выполнения процедуры не определено.

Если иерархический отчет строился без разбиения на таблицы по верхнему уровню групповой иерархии, то вызов процедуры из корневой группы будет приводить к генерации ошибки.

Если иерархический отчет строился с разбиением на таблицы по верхнему уровню групповой иерархии, то вызов процедуры из таблицы (группы) верхнего уровня устанавливает **CurTable**, **CurRow** и **CurColumn** в **nil** (то есть все координаты не определены).

Замечание 2. Данное свойство введено в программный интерфейс класса [Отчет](#) для выполнения действий внутри обработчика [ПриПодготовкеОтчета](#) при интерактивном раскрытии иерархии.

Пример

```
var Отч:Отчет;
var i,j: Integer;
var sum: Numeric;
Отч = Отчет.СоздатьПоИмени("ПоОборотам");
Отч.Построить;
sum = 0;
try
  Отч.ТекущаяТаблица = 1;
  for j=1..Отч.ЧислоСтрок do -- перебираем строки в таблице 1
    Отч.ТекущаяСтрока = j;
    if Отч.ГруппаОткрыта then
      Отч.ВойтиВГруппу;
      -- обработка элементов группы
      ...
      for i=1..Отч.ЧислоСтрок do -- перебираем строка в группе i
        Отч.ТекущаяСтрока = i;
        Отч.ТекущаяКолонка = 1;
        sum = sum + Отч.Оборот(Отчет.ВыводОстОбор.Roll,1);
        ...
      end;
      Отч.ВыйтиИзГруппы;
      -- Отч.ТекущаяСтрока автоматически равна j
    else
      sum = sum + Отч.Оборот(Отчет.ВыводОстОбор.Roll,1);
    end;
  end;
  ...
except
```



```
...  
end;
```

Описание

```
ВыйтиИзУточнения;  
LeavePrecision;
```

Назначение

Процедура осуществляет выход из уточнения, т.е. закрывает доступ к его данным. Для входа в уточнения используется процедура [ВойтиВУточнение](#).

Замечание. Данное свойство введено в программный интерфейс класса [Отчет](#) для выполнения действий внутри обработчика [ПриПодготовкеОтчета](#) при интерактивном уточнении.

Пример

См. в теме [Пример работы с уточняющими отчетами](#).

Описание

Обновить;
Reinit;

Назначение

Процедура Обновить|Reinit применяется для обновления настроек отчета и сброса его внутренней структуры (если он был построен).

Внимание! Добавление и удаление показателей (кроме пользовательских) после построения отчета, а также изменение некоторых их параметров **запрещено**.

Обновление отчета, построенного как уточняющий, (см. свойство [ЯвляетсяУточняющим / ServeAsPrecise](#) сделано "мягким", т.е. происходит обновление опций отчета, но поверх них накладываются условия уточнения.

Описание

```
ОграничитьКоличествоСтрок (КоличествоСтрок :Целое);  
LimitRowCount (RowCount :Integer);
```

Аргументы

КоличествоСтрок - заданное число строк.

Назначение

Процедура позволяет выводить в таблицах отчета по оборотам верхние строки, количество которых задано в первом параметре.

Процедура выполняется только для отчетов, в которых в момент ее вызова для всех таблиц была произведена [сортировка](#) по показателю /показателям.

Внимание! В отчет включаются только заданное количество строк с максимальными (минимальными) значениями ключа сортировки, остальные попадают в строку "Прочие".

Пример

Описание

```
ПереупорядочитьПо (Список :Строка [; ДляВсехТаблиц:Логическое]);
ResortBy (List :String [; ForAllTables : Logical]);
```

Аргументы

Список - список описаний сумм для сортировки, заданный в соответствии с описанным ниже синтаксисом;

ДляВсехТаблиц - необязательный параметр, по умолчанию равный True. Если он установлен равным False, то процедура применяется к конкретной таблице, и требуется, чтобы был установлен Report.CurTable. При равенстве True - установка таблицы не требуется, а при отсутствии в какой-либо таблице заданного номера столбца, сортировка производится не по сумме, а по вторичным ключам (то, что было задано в строке сортировки разбиения по строкам или по содержимому разбиения). При наличии столбцов сортировка также производится и по вторичным ключам.

Синтаксис списка описаний сумм

```
<Список>                = <Описание суммы> {", " <Описание суммы> }
<Описание суммы>        = [<Признак по возрастанию/убыванию>] "@" <Номер показателя>
                           [ "." <Вид суммы> [ "." <Формат суммы> [ "." <Номер столбца> ] ] ]

<Признак по
  возрастанию/убыванию> = "+" | "-"
<Номер показателя>      = целое (в строковом виде)
<Вид суммы>              = "НачОстаток" | "Оборот" | "КонОстаток"
<Формат суммы>          = "Дебет" | "Кредит" | "Свернутый"
<Номер столбца>         = целое (в строковом виде)
```

<Признак по возрастанию/убыванию> - необязательное значение, при отсутствии считается равным "+".
 <Вид суммы>, <Формат суммы>, <Номер столбца> - необязательные значения. Если часть строки не указана, то считаются заданными: Оборот.Дебет.1

Примеры:

```
1) '-@1.КонОстаток.Свернутый.2'
2) '+@1.Оборот.Дебет.1'
3) '@1.Оборот.Дебет' - равно 2)
4) '@1.Оборот' - равно 2), 3)
5) '@1' - равно 2), 3), 4)
6) '-@1.Оборот.Свернутый.1, @1.КонОстаток.Свернутый.1'
7) '' - отмена сортировки по суммам
```

Назначение

Процедура позволяет произвести пересортировку строк в таблицах в уже построенном отчете по сумме, которая определяется путем задания номера показателя, вида суммы и номера колонки.

Внимание! Пересортировка работает, если изначально была задана сортировка по строкам, и если разбиение по строкам - не по времени.

Пример

```
var Отч:Отчет;
Отч = Отчет.СоздатьПоИмени("ПоОборотам");
Отч.Построить;
...
Отч.ПереупорядочитьПо("+@1.Оборот.Свернутый", False);
```

Описание

Построить;
Build;

Назначение

Процедура предназначена для построения отчета. После ее вызова становятся доступными результаты отчета.

Пример

```
var Отч:Отчет;  
Отч = Отчет.СоздатьПоИмени( "ПоКонтрагентам" );  
Отч.Построить ;
```

См. также [Пример работы с объектом Отчет](#)

```

func РаскрытьИнтерактивноеУточнение(const аИмяУточняющегоОтчета :String) :Logical;
var локОтчетНайден :Logical;
var локНомерОтчета :Integer;
var локИмяУточняющегоОтчета :String;

Result = False;

if аИмяУточняющегоОтчета <> '' then
    -- проверяем, что отчет с таким именем есть...
    локОтчетНайден = False;
    for локНомерОтчета = 1 .. Отчет.PreciseReportCount do
        if Lo(аИмяУточняющегоОтчета) = Lo(Отчет.PreciseReport[локНомерОтчета]) then
            локОтчетНайден = True;
            Break;
        end;
    end;
    Assert(локОтчетНайден, Format('Отчет с именем "%s" не найден среди '+
        'интерактивно уточняющих отчетов', [аИмяУточняющегоОтчета]));
end;

локИмяУточняющегоОтчета = аИмяУточняющегоОтчета;
if Отчет.SelectPreciseReport(локИмяУточняющегоОтчета) then
    Result = True;

    Отчет.BuildPrecision;
    Отчет.EnterPrecision;

    -- Процессор.ИнициализироватьВременныеДанные;

    try
        -- ПодготовитьИерархию;

    finally
        -- Процессор.ОчиститьВременныеДанные;
        Отчет.LeavePrecision;
    end;
end;
end;

-- Событие ПриПодготовкеОтчета

func Шаблон_ПриПодготовкеОтчета(
    Action :ReportForm.ExpandActions;
    Report :Report
) :Logical;

Result = True;
ЗапрещенаПерерисовка = True;

try
    if Action = ReportForm.ExpandPrecision then
        if Report.PrecisionIsOpen then
            -- nothing...
        else
            -- находимся не в уточнении, и уточнение по текущей строке
            -- не сформировано...
            if not Report.InPrecision and not Report.CanEnterPrecision then
                -- если в диалоге выбора уточняющего отчета пользователь
                -- нажал кнопку Отмена, дальнейшая обработка не нужна...
                Result = Построитель.РаскрытьИнтерактивноеУточнение('');
            end;
        end;
    end;

    elseif Action = ReportForm.ExpandHierarchy then

```

```
    if Report.GroupIsOpen[rdRow] then
        -- nothing...
    else
        if not Report.CanEnterGroup[rdRow] then
            Построитель.РаскрытьГруппу;
        end;
    end;

    else
        -- Action = ReportForm.FullExpandHierarchy
        Построитель.РаскрытьИерархию;
    end;

    finally
        ЗапрещенаПерерисовка = False;
    end;
end;
```


Описание

```
ПостроитьГруппу (Измерение : Отчет.ИзмеренияОтчета [; Полностью :Логическое]);  
BuildGroup (Dimension : Report.RepDimensions [; Full :Logical]);
```

Аргументы

Измерение - predetermined [константа](#), измерения, по которой требуется построить группу.

Полностью - необязательный логический параметр, при значении "Истина" обеспечивается полное раскрытие иерархии в параметрических отчетах (для заполнения пользовательских показателей).

Назначение

Создает группу в иерархических отчетах при установке текущей строки.

Замечание. Данное свойство введено в программный интерфейс класса [Отчет](#) для выполнения действий внутри обработчика [ПриПодготовкеОтчета](#).

Описание

ПостроитьУточнение;
BuildPrecision;

Назначение

Создает уточнение в отчетах для текущей строки. Используется для интерактивного уточнения текущей строки.

Замечание. Данное свойство введено в программный интерфейс класса [Отчет](#) для выполнения действий внутри обработчика [ПриПодготовкеОтчета](#) при интерактивном уточнении.

Пример

См. в теме [Пример работы с уточняющими отчетами](#).

Описание

Сохранить;
Save;

Назначение

Метод Сохранить|Save применяется исключительно для пользовательских отчетов. Он позволяет сохранить настройки отчета, созданного при помощи конструктора [СоздатьПоИмени / CreateName](#) или [СоздатьНовый / CreateNew](#)

Пример

```
proc CreateNewReport(const AReportName, ABaseClassName :String; AShared :Logical);
    var Rep :Report;

    Rep = Report.CreateNew(AReportName, Report.byTurn, AShared);
    Rep.BaseClass = ABaseClassName;
    Rep.Caption = 'Caption';
    Rep.AccountPlan = 'Склад';
    ....
    Rep.Save;
end;

proc UpdateReport(const AReportName :String);
    var Rep :Report;

    Rep = Report.CreateName(AReportName);
    Rep.AccountPlan = 'Склад';
    ....
    Rep.Save;
end;
```

Описание

Удалить (Измерение : [Отчет.ИзмеренияОтчета](#));
Delete (Dimension :RepDimensions);

Аргументы

Измерение - [измерение](#) (таблица, строка или столбец), которое требуется удалить.

Назначение

Процедура удаляет текущую таблицу, строку или столбец - в зависимости от значения **Измерения**. При этом текущим становится следующий элемент, если он есть, если нет - текущее положение не определено. Если удалялась таблица, то становятся неопределенными внутренние координаты: **ТекущаяСтрока** и **ТекущаяКолонка**, в случае удаления строки - **ТекущаяКолонка**.

В иерархическом отчете при удалении группы переход осуществляется на следующую строку предыдущего уровня.

Внимание! При разбиении по времени на строки, столбцы или таблицы нельзя удалить соответственно строку, столбец или таблицу, в которой выведен ненулевой свёрнутый оборот или разделённый оборот. Таким образом, при разбиении по времени удалять можно лишь элементы, содержащие нулевой оборот, представленный в свёрнутом виде.

Пример

```
var R:Report;  
R = Report.CreateName("ОтчетПоКонтрагентам");  
-- построение отчета  
R.Build;  
R.CurTable = 1;  
R.CurRow = 1;  
R.CurColumn = 3;  
-- удаляем 3-й столбец  
R.Delete(Report.rdCol);  
-- анализируем прореженную структуру данных  
-- ...
```

Процедура УдалитьПоказатель / DeleteIndicator

Описание

```
УдалитьПоказатель (Индекс :Целое);  
DeleteIndicator (Index :Integer);
```

Аргументы

Индекс - порядковый номер показателя в списке показателей в поле **Имя** на [странице "Показатели"](#) диалога "Внутренние отчеты".

Назначение

Удаляет показатель из списка по заданному номеру.

Пример

Описание

```
СводитьСУчетом :ТипыСведенияПроводок[];  
ConsolidateWith :ConsolidationParams[];
```

Назначение

Сводные проводки могут формироваться по одному или нескольким параметрам сведения, заданным в [перечислимом типе](#) **ТипыСведенияПроводок / TransConsolidationTypes**.

Данное свойство используется для задания или просмотра множества параметров сведения, с учетом которых будут формироваться сводные проводки. Если ни один параметр не задан, то сводные проводки не формируются.

Пример

Описание

СводныеПроводки :Логический;
ConsolidatedTrans :Logical;

Назначение

Данное свойство позволяет управлять процессом формирования сводных проводок. Если свойство равно значению "Истина", то в отчете формируются сводные проводки, иначе - не формируются.

Пример

Описание

ФорматЗначенияПоказателя (Индекс :Целое; ОстОбор :Отчет.ВидыОстОбор;
ДебКре :Отчет.ВыводОстОбор) :Строка;
IndicatorValueFormat (Index :Integer; SumKind :SumKinds; DebCre :SumDebCre) :String;

Аргументы

Индекс - номер типа показателя;

ОстОбор - тип показателя, задаваемый одной из predefined констант типа [ВидыОстОбор](#);

ДебКре - одна из predefined [констант](#), задающая режим вывода показателей, разделенных по дебету|кредиту или в свернутом виде.

Назначение

Данное свойство позволяет через программный интерфейс к отчету связывать с каждым значением показателя отчета свой формат вывода.

Внимание! При использовании макросов в строке форматирования необходимо в шаблоне отчета в соответствующих клетках включить опцию "Использовать макросы".

Пример

Описание

```
ВставитьПоказатель (Индекс :Целое; Имя :Строка [; ПользовательскийОтчет :Logical] ) :  
ПоказательОтчета;  
InsertIndicator (Index :Integer; Name :String [;Custom :Logical]) : ReportIndicator;
```

Аргументы

Индекс - порядковый номер показателя в списке показателей, по которому вставляется новый показатель;

Имя - имя показателя, описанное в структуре учета, которое следует добавить в список показателей в поле **Имя** на [странице "Показатели"](#) диалога "Внутренние отчеты";

ПользовательскийОтчет - необязательный логический параметр, определяющий, является ли отчет пользовательским.

Назначение

По заданному имени и номеру показателя из класса **ПоказательОтчета** добавляет его в список показателей и возвращает ссылку на него.

Изменить имя показателя при необходимости можно с помощью свойства [Имя/Name](#) класса **ПоказательОтчета**.

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
var RepInd : ReportIndicator;  
....  
Rep = Report.CreateName("ПоОборотам");  
RepSplit = Rep.Split[Report.rdRow];  
RepInd = Rep.InsertIndicator(1, 'Остаток', True);
```

Описание

```
ВыбратьУточняющийОтчет (var ИмяОтчета :Строка) :Логическое;  
SelectPreciseReport (var Name :String) :Logical;
```

Аргументы

ИмяОтчета - имя уточняющего отчета.

Назначение

Данная функция устанавливает уточняющий отчет для текущей строки.

Если имя отчета не задано (пустая строка), выбор осуществляется стандартным способом из списка отчетов диалога "Внутренние отчеты" или путем выбора разбиения в случае отсутствия уточняющего отчета.

Если функция не была вызвана, разбиение по строкам для уточнения считается равным разбиению по счетам.

Замечание. Данное свойство введено в программный интерфейс класса [Отчет](#) для выполнения действий внутри обработчика [ПриПодготовкеОтчета](#) при интерактивном уточнении.

Пример

См. в теме [Пример работы с уточняющими отчетами](#).

Описание

```
ДобавитьПоказатель (Имя :Строка [; ПользовательскийОтчет :Logical] ) : ПоказательОтчета;  
AddIndicator (Name :String [;Custom :Logical]) : ReportIndicator;
```

Аргументы

Имя - имя показателя, которое следует добавить в список показателей в поле **Имя** на [странице "Показатели"](#) диалога "Внутренние отчеты";

ПользовательскийОтчет - необязательный логический параметр, определяющий, является ли отчет пользовательским.

Назначение

По заданному имени добавляет показатель из класса **ПоказательОтчета** в список показателей и возвращает ссылку на него.

Изменить имя показателя при необходимости можно с помощью свойства [Имя/Name](#) класса **ПоказательОтчета**.

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
var RepInd : ReportIndicator;  
....  
Rep = Report.CreateName("ПоОборотам");  
RepSplit = Rep.Split[Report.rdRow];  
RepInd = Rep.AddIndicator('Сумма', True);
```

Описание

КоличествоПроводок (ДебКре :Отчет.ВыводОстОбор):Целое;
TransCount (DebCre :SumDebCre) :Integer;

Аргументы

ДебКре - одна из predetermined [констант](#), задающая режим вывода показателей, разделенных по дебету|кредиту или в свернутом виде.

Назначение

Возвращает количество проводок, сформировавших текущий показатель (идентифицируемый тремя координатами с номерами таблицы, строки и колонки, заданными с помощью свойств [ТекущаяТаблица](#), [ТекущаяСтрока](#), [ТекущаяКолонка](#)).

Пример

```
var R:Report;  
R = Report.Create("ОтчетПоКонтрагентам");  
-- задание входных параметров...  
R.Build;  
R.CurTable = 1;  
R.CurRow = 1;  
R.CurColumn = 3;  
-- выводим количество операций по контрагенту  
trace(R.TransCount(Roll));
```

Описание

КонечныйОстаток (ДебКре: [ВыводОстОбор](#); ИндексПоказателя: Целое) :Измеритель;
EndSaldo (DebCre :SumDebCre; IndicatorIndex :Integer) :Измеритель;

Аргументы

ДебКре - одна из predetermined [констант](#), задающая режим вывода показателей (в разделенном или свернутом виде);

ИндексПоказателя - определяет, в каком показателе, из числа доступных (измерители, число+пара валют) для отчета, запрашивается остаток. Измерители нумеруются, начиная с 1.

Назначение

Возвращает остаток на конец отчетного периода для показателя из [текущей строки](#), [текущей колонки](#), [текущей таблицы](#) отчета, заданных в виде измерителей, иначе - выдает ошибку. Для показателей, заданных как число и пара валют, функция возвращает число и первую валюту.

Прежде чем вызывать данную функцию, необходимо предварительно построить отчет с помощью процедуры [Построить / Build](#).

Пример

См. также [Пример работы с объектом Отчет](#)

Описание

НачальныйОстаток (ДебКре : [ВыводОстОбор](#); ИндексПоказателя : Целое) : Измеритель;
BegSaldo (DebCre : [SumDebCre](#); IndicatorIndex : Integer) : Измеритель;

Аргументы

ДебКре - одна из предопределенных [констант](#), задающая режим вывода показателей (в разделенном или свернутом виде);

ИндексПоказателя - определяет, в каком показателе, из числа доступных (измерители, число+пара валют) для отчета, запрашивается остаток. Измерители нумеруются, начиная с 1.

Назначение

Функция возвращает остаток на начало отчетного периода для показателя из [текущей строки](#), [текущей колонки](#), [текущей таблицы](#) отчета, заданных в виде измерителей, иначе - выдает ошибку. Для показателей, заданных как число и пара валют, функция возвращает число и первую валюту.

Прежде чем вызывать данную функцию, необходимо предварительно построить отчет с помощью процедуры [Построить / Build](#).

Пример

См. также [Пример работы с объектом Отчет](#)

Описание

Оборот (ДебКре : [ВыводОстОбор](#) ; ИндексПоказателя : Целое) : Измеритель ;
Turn (DebCre : SumDebCre ; IndicatorIndex : Integer) : Измеритель ;

Аргументы

ДебКре - одна из predetermined [констант](#), задающая режим вывода оборотов, разделенных по дебету|кредиту или в свернутом виде;

ИндексПоказателя - показатель, по которому вычисляется оборот. Показатели нумеруются, начиная с 1.

Назначение

Возвращает оборот по показателю из [текущей строки](#), [текущей колонки](#), [текущей таблицы](#) отчета, заданного в виде измерителя, иначе - выдает ошибку. Для показателей, заданных как число и пара валют, функция возвращает число и первую валюту.

Прежде чем вызывать данную функцию, необходимо предварительно построить отчет с помощью процедуры [Построить / Build](#).

Пример

См. также [Пример работы с объектом Отчет](#)

Описание

```
Создать(Тип :ТипыОтчета): Отчет;  
Create (Type :RepTypes) : Report;
```

Аргументы

Тип - обязательный параметр, задающий [тип](#) создаваемого отчета. По умолчанию, при создании отчета по оборотам **Тип** = byTurn, разбиение на строки устанавливается = byAcc, по справочнику **Тип** = byReference - byParam, в остальных случаях - none.

Назначение

Создает объект **Отчет** с параметрами, установленными в соответствии с параметрами указанного внутреннего отчета, тип которого определяется в первом параметре, и возвращает ссылку на него. Сам отчет при этом не строится, для создания воспользуйтесь процедурой [Построить / Build](#).

Следует отметить, что не все настройки отчета, заданные ранее с помощью диалоговой формы в редакторе проекта, переносятся в создаваемый объект **Отчет**, поскольку часть опций управляет не внутренними структурами данных, а лишь их отображением. Так, установки, управляющие тем, как следует выводить остатки и обороты (не показывать вообще, показывать в свернутом виде или раздельно), задаются в объекте **Отчет** через первый аргумент функций получения оборотов и остатков. Фактически это означает, что и обороты, и остатки всегда доступны в объекте **Отчет** в любом представлении и тем самым оставляют за программистом решение вопроса, что нужно, а что не нужно показывать пользователю.

После создания отчета и до вызова метода **Построить** программист имеет возможность поменять параметры отчета.

Кроме того, существует несколько "запрещенных" комбинаций параметров, при которых построение внутренних отчетов программным путем (с помощью объекта **Отчет**) невозможно. При этом объект **Отчет** будет создаваться, но последующий вызов процедуры **Построить** не формирует отчет и попытка прочитать результаты приводит к генерации исключительной ситуации. Такая ситуация возникает, например, при разбиении отчета на строки по проводкам. Для получения отчетов по проводкам необходимо пользоваться запросом полупроводок (см. класс [ЗапросПолупроводок](#)).

В зависимости от того, является ли отчет иерархическим или нет, программа по-разному выводит информацию об объектах хозяйственного учета и операциях над ними. Например, если аналитический справочник имеет несколько групп и отчет создается с признаком "иерархический" и с разбиением на строки по аналитике, то в результирующей таблице отчета наряду со строками по объектам аналитического учета будут добавляться строки с суммарными показателями по группам объектов.

Пример

```
proc P1(Sender :Button);  
  var R : Report;  
  R = Report.Create(byReference);  
  -- отчет по справочнику  
end;
```


Описание

СоздатьНовый(Имя :Строка; Тип : [Отчет.ТипОтчета](#) [; Общий :Логическое]) : [Отчет](#);
CreateNew(Name :String; Type : [Report.RepTypes](#) [; Shared :Logical]) : [Report](#);

Аргументы

Имя - имя пользовательского (общего) отчета;

Тип - обязательный параметр, задающий [тип](#) создаваемого отчета;

Общий - необязательный параметр, по умолчанию он равен TRUE и используется совместно с методом [Сохранить / Save](#).

Назначение

Функция создает только пользовательский отчет программным способом.

Опции отчета, созданного данной функцией, после настройки можно сохранить при помощи метода [Сохранить / Save](#).

Пример

```
proc CreateNewReport(const AReportName,  
    ABaseClassName :String; AShared :Logical);  
    var Rep :Report;  
  
    Rep = Report.CreateNew(AReportName,  
        Report.byTurn, AShared);  
    Rep.BaseClass = ABaseClassName;  
    Rep.Caption = 'Caption';  
    Rep.AccountPlan = 'Склад';  
    ....  
    Rep.Save;  
end;
```

Функция СоздатьПоИмени / CreateName

Описание

```
СоздатьПоИмени(Имя:Строка [;Тип :ТипыОтчета ]) : Отчет;  
CreateName (Name:String [; Type :RepTypes ]) : Report;
```

Аргументы

Имя - имя проектного или пользовательского отчета;

Тип - необязательный параметр, задающий [тип](#) создаваемого отчета. Он используется для отчетов неопределенного типа.

Назначение

Создает как проектные, так и пользовательские (общие) отчеты программным способом.

Опции пользовательского отчета, созданного при помощи данной функции, после настройки можно сохранить при помощи метода [Сохранить / Save](#). Для сохранения проектных отчетов воспользуйтесь методом [Построить / Build](#).

Пример

```
proc UpdateReport(const AReportName :String);  
    var Rep :Report;  
  
    Rep = Report.CreateName(AReportName);  
    Rep.AccountPlan = 'Склад';  
    ....  
    Rep.Save;  
end;
```

Перечислимые типы класса Отчет

Во многих свойствах и функциях класса [Отчет/Report](#) используются константы следующих перечислимых типов, определенных в классе **Отчет**:

-  [ТипыОтчета / RepTypes](#) &
-  [ФорматыОтчета / RepFormats](#) &
-  [ИзмеренияОтчета / RepDimension](#) &
-  [ТипыРазбиения / SplitTypes](#) &
-  [ТипыРазбиенияПоВремени / DateSplitTypes](#) &
-  [ДиапазоныРазбиения / SplitRangeTypes](#) &
-  [ВидыОстОборSumKinds](#) &
-  [ВыводОстОбор / SumDebCre](#) &
-  [ФорматыОстОбор / SumFormats](#) &
-  [ТипыПоказателей / IndicatorTypes](#) &
-  [ФорматыОстОборПоказателя / IndicatorSumFormats](#) &
-  [ФорматыПоказаПоказателя / IndicatorShowFormats](#) &
-  [ВидыОграниченияПоказателя / IndicatorLimitKinds](#) &
-  [ЧастиПоказателя / IndicatorParts](#) &
-  [ОпцииПоказателя / IndicatorOptions](#) &
-  [ФорматыРазделенияПоказателя / IndicatorSplitFormats](#) &
-  [ФорматыВыводаПоказателя / IndicatorOutputFormats](#) &
-  [МетодыАгрегированияПоказателя / IndicatorAggregateMethods](#) &
-  [ТипыВыделенияСтрок / RowStripeTypes](#) &
-  [ТипыСведенияПроводок / TransConsolidationTypes](#) &
-  [ТипыУточненияПроводок / TransPreciseTypes](#) &
-  [ЭлементыПодстановки / SubstItems](#) &

Константы видов ограничения показателя **ВидыОграниченияПоказателя** / **IndicatorLimitKinds**

Перечислимый тип **ВидыОграниченияПоказателя** / **IndicatorLimitKinds**, определенный в классе **Отчет/Report**, содержит следующие предопределенные константы, для программного задания способа отсекающих показателей по их величине:

- **огрБолееАбс / cutAboveAbs** - отсекающие показатели, превышающих указанную величину по модулю;
- **огрМенееАбс / cutLessAbs** - отсекающие показатели, меньше указанной величины по модулю;
- **огрБолее / cutAbove** - отсекающие показатели, превышающих указанную величину;
- **огрМенее / cutLess** - отсекающие показатели, меньше указанной величины.

Эти константы применяются в свойстве [ВидОграничения/LimitKind](#) класса **ПоказательОтчета/ReportIndicator**.

Перечислимый тип **ВыводОстОбор / SumDebCre** предназначен для указания требуемого режима вывода результатов, например, вывод остатков и оборотов только по дебету, только кредиту или по дебету и кредиту в свернутом виде. Данный перечислимый тип включает набор следующих констант:

- **byDeb** - дебетовая часть показателя;
- **byCre** - кредитовая часть показателя;
- **roll** - показатель в свернутом виде.

Эти константы применяются при вызове функций класса **Отчет / Report**:

- [Оборот](#);
- [НачальныйОстаток](#);
- [КонечныйОстаток](#);
- [КоличествоПроводок](#);
- [ЗначениеПоказателя](#).

Константы также применяются в свойствах класса **ПоказательОтчета/ReportIndicator**:

- [ЗначениеОграничения / LimitValue](#) (класс **ПоказательОтчет**);
- [Ограничить / LimitValue](#) (класс **ПоказательОтчет**);
- [ВидОграничения / LimitKind](#) (класс **ПоказательОтчет**).

Константы диапазонов разбиения ДиапазоныРазбиения / SplitRangeTypes

Перечислимый тип **ДиапазоныРазбиения / SplitRangeTypes**, определенный в классе **Отчет/Report**, используется для задания способа разбиения отчета (по маске, по перечню, по шкале).

Данный тип содержит следующие predefined константы:

- **byMatch** - по маске;
- **byEnum** - по перечню;
- **byScale** - по шкале.

Константы данного типа используются при определении свойства [ДиапазонРазбиения / SplitRange](#) класса **РазбиениеОтчета / ReportSplit**.

Для выбора типа показателя применяется специальный перечисляемый тип **ВидыОстОбор / SumKinds**, включающий следующий набор констант для вывода оборотов, начальных и конечных остатков:

- **skBegSaldo** - начальный остаток;
- **skTurn** - оборот;
- **skEndSaldo** - конечный остаток.

Эти константы применяются в свойствах классов **Отчет/Report**, **ГрафикОтчета/ReportChart** и **ПоказательОтчета/ReportIndicator**:

- [ЗначениеПоказателя / IndicatorValue](#) (класс **Отчет**);
- [ОстаткиОбороты / SumKind](#) (класс **ГрафикОтчета**);
- [ФорматОстОбор / SumFmt](#) (класс **ПоказательОтчет**);
- [ЗначениеОграничения / LimitValue](#) (класс **ПоказательОтчет**);
- [Ограничить / LimitValue](#) (класс **ПоказательОтчет**);
- [ВидОграничения / LimitKind](#) (класс **ПоказательОтчет**).

Измерениями отчета являются таблицы, строки и колонки, которые однозначно определяют способ расположения показателей в построенных отчетах. Все они перечисляются в типе **ИзмеренияОтчета / RepDimensions**, который включает следующие константы:

- **rdTab** - разбиение по таблицам;
- **rdRow** - разбиение по строкам;
- **rdCol** - разбиение по столбцам;
- **rdInline** - интерактивное разбиение.

Константа **rdInline** позволяет до построения уточнения изменять его настройки. Т.е. у текущей строки отчета можно [установить](#), запросить или изменить настройки ее уточнения (те же свойства, которые относятся к разбиению по строкам). Находясь внутри уточнения эти настройки запрашиваются уже из разбиения rdRow, при этом они могут быть несколько скорректированы для совместимости с основным отчетом (если не совпадает вывод иерархии по строкам).

Константы определения измерений используются в свойствах и методах классов **Отчет** и **РазбиенияОтчета**:

- [Разбиение / Split](#);
- [ЗначениеРазбиения / SplitValue](#);
- [ЭтоГруппа / IsGroup](#);
- [Удалить / Delete](#);
- [ВнутриГруппы / InGroup](#);
- [МеткаРазбиения / SplitTag](#)
- [Измерение / Dimension](#).

Отчеты могут строиться в разрезе счетов, корреспондируемых счетов, параметров (аналитики), дат, или же без разбиения.

Константы типов разбиения отчета **ТипыРазбиения / SplitTypes** используются для указания логического разреза, в котором должны рассматриваться показатели отчета в каждом из измерений (таблица, строка, столбец):

- **None** - нет разбиения;
- **ByAcc** - по счетам;
- **ByCorAcc** - по корреспондируемым счетам;
- **ByParam** - по параметрам;
- **ByCorParam** - по корреспондируемым параметрам;
- **ByDate** - по дате;
- **ByDocum** - по документу;
- **byDebCre** - по дебету и кредиту.

Константы разбиений применяются при определении свойства [ТипРазбиения](#).

Константы разбиений по времени ТипыРазбиенияПоВремени / DateSplitTypes

Перечислимый тип **ТипыРазбиенияПоВремени / DateSplitTypes** используются для детализации разбиения по времени, включая агрегирование показателей по дням, неделям, месяцам, кварталам или годам. Для этих целей используются следующие константы разбиений по времени:

- **ByDay** - по дням;
- **ByWeek** - по неделям;
- **ByMonth** - по месяцам;
- **ByQuart** - по кварталам;
- **ByYear** - по годам;
- **byHour** - по часам;
- **byMinute** - по минутам;
- **bySecond** - по секундам.

Константы разбиений по времени применяются при определении свойств класса **РазбиениеОтчета**:

- [Период](#);
- [ПериодДиапазона](#).

Внутренние отчеты могут быть построены в различных форматах, например, в формате *.rtf или *.csv, для экспорта их в программу Word или Excel.

Для задания формата отчета используется перечислимый тип **`ФорматыОтчета / RepFormats`**, включающий следующие константы:

- **`fmtТекст/fmtText`** - текстовый формат;
- **`fmtШаблон/fmtTemplate`** - формат шаблона;
- **`fmtHTML/fmtHTML`** - HTML формат;
- **`fmtRTF/fmtRTF`** - RTF формат;
- **`fmtCSV/fmtCSV`** - CSV формат;
- **`fmtГрафик/fmtGraph`** - графический формат;
- **`fmtDBF/fmtDBF`** - DBF формат.

Перечислимый тип *МетодыАгрегированияПоказателя/IndicatorAggregateMethods*, определенный в классе *Отчет*, включает следующие константы, описывающие операции агрегирования показателей:

- **agrNone** - операция агрегирования показателей отключена;
- **agrSum** - показатели будут просуммированы;
- **agrMin** - вычисление минимального значения показателя;
- **agrMax** - вычисление максимального значения показателя.

Метод агрегирования можно устанавливать для [пользовательских и простых](#) показателей.

Константы, определенные в данном типе, используются в свойстве [МетодАгрегирования](#) класса *ПоказательОтчета*.

Константы способов вывода показателей ОпцииПоказателя / IndicatorOptions

Программный интерфейс класса *Отчет* предоставляет пользователям набор возможностей для управления выводом показателей в отчетах по проводкам. Каждая такая возможность (опция) описывается соответствующей константой в перечислимом типе **ОпцииПоказателя / IndicatorOptions**.

Константы, определенные в данном типе, используются в свойстве **Опция** класса *ПоказательОтчета*.

Причем не все опции доступны для всех видов показателей, т.к. часть из них имеет смысл *только для измерителей*, но не для показателя какого-либо другого вида, такого как *аналитический признак* или *число*.

В данном перечислимом типе определены следующие константы, позволяющие:

- **opShowSeparate** - выводить значения отдельным столбцом;
- **opShowSaldo** - показать остаток (для измерителей). Остаток выводится до и после таблиц с проводками;
- **opShowRowTotal** - показывать нарастающий итог по строке в отчетах по проводкам (для измерителей);
- **opShowColTotal** - показывать итоги по колонке (для измерителей);
- **opShowSplitInTwoLines** - расщепленные параметры выводить в одном столбце в две строки, т.е. кредит под дебетом. Данная опция доступна только через программный интерфейс;
- **opCallEvent** - вызывать события. Опцию можно задать только программно;
- **opVisible** - обеспечить программный интерфейс к флагу видимости показателя. В диалоге внутренних отчетов на странице "[Показатели](#)" флаг ☒ располагается справа от имени каждого показателя, перечисленного в поле **Имя**.
- **opSwapNegValue** - в *оборотных отчетах* и *отчетах по проводкам* программно перемещать отрицательный дебетовый оборот в кредит, если в свойстве **Опция** класса *ПоказательОтчета* установить

RepInd.Option[Report.SwapNegValue] = True;

Константа обеспечивает программный доступ к свойству показателя отчета (флаг **Отрицательный дебет в кредит** в группе флагов **Выводить**) в диалоге "[Расширенная настройка показателя](#)".

Константы типов выделения строк **ТипыВыделенияСтрок / RowStripeTypes**

В функциях класса **Отчет / Report** разрешается управлять выделением строк как для отчетов по проводкам, так и для отчетов по оборотам. Можно задавать различные стили для выделения строк, например, четных и нечетных строк или строк, принадлежащих различным документам.

Эта возможность реализована с помощью перечислимого типа **ТипыВыделенияСтрок / RowStripeTypes**, включающего следующие константы:

- **stOdd** - четные/нечетные строки;
- **stDocum** - по документу (только для отчетов по проводкам);
- **stNone** - не выделять.

Указанные константы применяются при определении свойства [ВыделятьСтрокиПо / StripeRowsBy](#).

Каждый отчет имеет так называемый тип - описательную характеристику, влияющую на область применения отчета и правила работы с ним. Поддерживаемые системой типы отчетов обозначаются на уровне программирования с помощью перечислимого типа **ТипыОтчета / RepTypes**, в который входят следующие константы:

- **byTrans** - отчет по проводкам;
- **byTurn** - отчет по остаткам и оборотам;
- **byReference** - отчет по справочнику;
- **byNone** - отчет неопределенного типа.

Данные константы используются в свойствах и методах класса **Отчет/Report**:

[Создать/Create](#);

[СоздатьНовый/CreateNew](#);

[СоздатьПоИмени/CreateName](#);

[ТипОтчета / ReportType](#).

Константы типов показателей **ТипыПоказателей** / **IndicatorTypes**

Для определения типа показателя отчета как пользовательского, так и обычного (непользовательского) используются следующие константы, описанные в перечислимом типе **ТипыПоказателей** / **IndicatorTypes**:

- **simple** / **простой** - простой тип;
- **calc** / **вычисляемый** - вычисляемый тип;
- **custom** / **пользовательский** - пользовательский тип.

Константы типов показателей применяются при определении свойства [ТипПоказателя](#) / [IndicatorType](#) класса **ПоказательОтчета**.

Константы формирования сводных проводок ТипыСведенияПроводок / TransConsolidationTypes

Перечислимый тип **ТипыСведенияПроводок / TransConsolidationTypes** используется в функциях класса **Отчет / Report** для управления формированием сводных проводок в отчетах по проводкам для объединения нескольких проводок, имеющих одинаковую аналитику или одинаковые счета дебета и кредита, по дате, по аналитике, по документу.

Данный перечислимый тип включает следующие константы:

- **withDate** - по дате;
- **withAnalytics** - по аналитике;
- **withDocument** - по документу.

Указанные константы применяются при определении свойств:

- [СводныеПроводки / ConsolidatedTrans](#);
- [СводитьСУчетом / ConsolidateWith.htm](#).

Константы уточнения проводок **ТипыУточненияПроводок / TransPreciseTypes**

Перечислимый тип **ТипыУточненияПроводок / TransPreciseTypes**, определенный в классе **Отчет/Report**, содержит следующие predefined константы, описывающие способ интерактивного уточнения показателей в результатах отчета:

- **byOpenDoc** - уточнение с помощью документа, породившего проводку;
- **byOpenJur** - уточнение с помощью журнала проводок.

Указанные константы применяются при определении свойства [ПриУточненииПроводок / PreciseTransBy](#).

Константы способа вывода показателей

ФорматыВыводаПоказателя / IndicatorOutputFormats

Перечислимый тип **ФорматыВыводаПоказателя / IndicatorOutputFormats**, определенный в классе **Отчет / Report**, предназначен для определения способа вывода показателей по дебету и/или кредиту.

В данном типе определены следующие константы:

- **ofSplit** - дебет и кредит показываются в отдельных столбцах;
- **ofDeb** - выводится только колонка с дебетовой составляющей показателя;
- **ofCre** - выводится только колонка с кредитовой составляющей показателя;
- **ofBoth** - показатель показывается в одном столбце, разделенном на два подстолбца, в которых выводятся соответственно дебетовая и кредитовая составляющие показателя. Шапка заголовка к столбцу состоит из двух строк, в верхней - выводится заголовок столбца, а в нижней - Дебет и Кредит.

Константы данного типа используются при работе с полем [ФорматВывода/OutputFmt](#).

Константы форматов вывода остатков и оборотов ФорматыОстОбор / SumFormats

Перечислимый тип **ФорматыОстОбор / SumFormats** класса **Отчет / Report** содержит набор констант, которые используются для задания формата вывода остатков и оборотов:

- **sfNone** - не выводить;
- **sfRoll** - выводить в свернутом виде;
- **sfFull** - выводить в разделенном виде;
- **sfTotalRoll** - выводить итоговые остатки в свернутом виде;
- **sfTotalFull** - выводить итоговые остатки в разделенном виде;
- **sfRollAsFull** - формат остатков "Свернутый отдельно".

Указанные константы применяются при определении свойства [ФорматОстОбор / SaldoTurnFmt](#)

Константы настройки видимости показателя ФорматыОстОборПоказателя / IndicatorSumFormats

Перечислимый тип **ФорматыОстОборПоказателя / IndicatorSumFormats** класса **Отчет / Report** используется в отчетах по оборотам для настройки видимости показателя при выводе оборотов, начальных и конечных остатков.

Данный перечислимый тип содержит следующий набор констант:

- **msfDebCre** - выводится дебет и кредит, если показатель разделенный, или разница дебет-кредит, если показатель свёрнутый;
- **msfDeb** - выводится только дебет, если показатель разделенный или дебетовая часть, если показатель свёрнутый;
- **msfCre** - выводится только кредит, если показатель разделенный или кредитовая часть, если показатель свёрнутый;
- **msfNone** - не выводится ни один показатель.

Замечание. По умолчанию берется значение - msfDebCre.

Эти константы применяются при определении свойства [ФорматОстОбор / SumFmt](#).

Константы форматов вывода колонок

ФорматыПоказаПоказателя / IndicatorShowFormats

В функциях класса **Отчет / Report** перечислимый тип **ФорматыПоказаПоказателя / IndicatorShowFormats** определяет константы, описывающие, в каких колонках должен выводиться показатель:

- **mshAll** - в колонках разбиения и в итоговой колонке (по умолчанию);
- **mshSplit** - в колонках разбиения;
- **mshTotal** - в итоговой колонке;
- **mshNone** - не выводится нигде.

Указанные константы применяются при определении свойства [ФорматПоказа / ShowFmt](#).

Константы форматов разделения показателей ФорматыРазделенияПоказателя / IndicatorSplitFormats

Для тех параметров проводок (счетов), которые имеются как у счета дебета, так и у счета кредита, система позволяет выводить показатели отдельно или только один из них (либо общее значение, либо дебетовое).

Настройка этих режимов производится с помощью констант перечислимого типа **ФорматыРазделенияПоказателя / IndicatorSplitFormats** класса **Отчет / Report**:

- **spAuto** - автоматически (в зависимости от наличия расщепления в каждой конкретной проводке);
- **spAlways** - всегда расщеплять (независимо от наличия расщепления в проводке);
- **spNone** - не расщеплять (независимо от наличия расщепления).

В последнем случае, если значения расщепленного (на дебетовую и кредитовую часть) в проводке параметра отличаются друг от друга, то в колонке результирующей таблицы (для данного параметра) будет выводиться дебетовое значение, а кредитовое попадет лишь в общую колонку, где в виде строки перечислены все параметры проводки.

Вышеприведенные константы используются при работе с полем [ФорматРазделения / SplitFmt](#).

Константы вывода показателя по частям ЧастиПоказателя / IndicatorParts

Перечислимый тип **ЧастиПоказателя / IndicatorParts**, определенный в классе **Отчет/Report**, позволяет выводить не весь показатель полностью, а отдельную его часть, например, значение показателя или единицу измерения.

Данный тип содержит следующие предопределенные константы:

- **чсЗначение / ptValue** - выводится значение показателя;
- **чсЕдИзм / ptUnit** - выводится единица измерения.

Константы, определенные в данном типе, используются в событии [ПриОбновленииОтчета](#) класса **Шаблон**.

Перечислимый тип *ЭлементыПодстановки / SubstItems*, определенный в классе *Отчет*, включает набор элементов, для которых в отчетах по оборотам разрешается программно изменять имена заголовков.

В данном типе определены следующие константы, которые позволяют изменить:

- **sbColTotals** - заголовок итоговой колонки;
- **sbTabTotals** - заголовок итоговой таблицы;
- **sbNilAnalit** - надпись для значения аналитики, равного nil (значение не задано).

Пустая строка в качестве значения поля означает значение по умолчанию.

Константы, определенные в данном типе, используются в свойстве [ПодставитьЗаголовок / SubstCaption](#).

Класс *ПоказательОтчета / ReportIndicator* относится к группе классов, предназначенных для построения внутренних отчетов, в нем сконцентрированы все свойства, относящиеся к показателям отчета. Класс наследует свойства от родительского класса [Объект](#).

Внимание! Для создания объектов данного класса используются функции [ВставитьПоказатель](#) и [ДобавитьПоказатель](#) класса *Отчет/Report*.

Непосредственно в классе *ПоказательОтчета / ReportIndicator* определены следующие свойства и методы:

- [Поле Имя / Name](#)
- [Поле Содержимое / Contents](#)
- [Поле Заголовок / Caption](#)
- [Поле Владелец / Owner](#)
- [Поле РодИндекс / ParentIndex](#)
- [Поле Группа / Group](#)
- [Поле ЭтоГруппа / IsGroup](#)
- [Поле ТипПоказателя / IndicatorType](#)
- [Поле ТипЗначения / ValueType](#)
- [Поле ФильтрСчетов / AccountFilter](#)
- [Поле ФильтрПараметров / ParameterFilter](#)
- [Поле КоррЗначение / CorrValue](#)
- [Поле ВПересчетеНа / ConvertTo](#)
- [Поле ВПересчетеНаБазовые / ConvertToBase](#)
- [Поле ВПересчетеПоПараметру / ConvertByParam](#)
- [Поле КоррЗначениеРасш / CorrValueEx](#)
- [Поле ВПересчетеНаРасш / ConvertToEx](#)
- [Поле ВПересчетеНаБазовыеРасш / ConvertToBaseEx](#)
- [Поле ВПересчетеПоПараметруРасш / ConvertByParamEx](#)
- [Поле МетодАгрегирования / AggregateMethod](#)
- [Поле Ограничить / Limit](#)
- [Поле ЗначениеОграничения / LimitValue](#)
- [Поле ВидОграничения / LimitKind](#)
- [Поле ФорматОстОбор / SumFmt](#)
- [Поле ФорматПоказа / ShowFmt](#)
- [Поле ПоказЕдИзм / ShowUnit](#)
- [Поле Формат / Format](#)
- [Поле МинШирина / MinWidth](#)
- [Поле МаксШирина / MaxWidth](#)
- [Поле МинШиринаЕдИзм / UnitMinWidth](#)
- [Поле МаксШиринаЕдИзм / UnitMaxWidth](#)
- [Поле Опция / Option](#)
- [Поле ФорматРазделения / SplitFmt](#)
- [Поле ФорматВывода / OutputFmt](#)

Описание

ВидОграничения [ОстОбор : [Отчет.ВидыОстОбор](#); ДебКре : [Отчет.ВыводОстОбор](#)] :
[Отчет.ВидыОграниченияПоказателя](#);
LimitKind [SumKind : [Report.SumKinds](#); DebCre : [Report.SumDebCre](#)] :
[Report.IndicatorLimitKinds](#);

Аргументы

ОстОбор - индекс, который может принимать значение одной из предопределенных констант перечислимого типа [ВидыОстОбор](#). Он определяет, при выводе каких данных отчета устанавливается режим ограничения: начальных остатков (skBegSaldo), оборотов (skTurn) или конечных остатков (skEndSaldo).

ДебКре - индекс, который определяет, какая часть показателя будет выводиться в отчете: в свернутом виде (roll) или в разделенном виде с отдельным показом дебетовой (byDeb) или кредитовой (byCre) части. Индекс может принимать одно из значений перечислимого типа [ВыводОстОбор](#).

Назначение

Поле позволяет программным путем узнать и изменить вид ограничения заданных показателей при их отображении в отчете по оборотам. Показатели можно отграничить как по величине (больше или меньше заданной), так и по модулю. Поле может содержать одно из предопределенных значений перечислимого типа [ВидыОграниченияПоказателя](#). Ограничению, подвергается показатель, вид которого в отчете удовлетворяет заданной комбинацией индексов **ОстОбор** и **ДебКре**.

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
var RepInd : ReportIndicator;  
....  
Rep      = Report.CreateName( "ПоОборотам" );  
RepSplit = Rep.Split[Report.rdRow];  
RepInd   = Rep.InsertIndicator(1, 'Сумма', True);  
-- включаем режим исключения "мелких" свернутых остатков  
-- по первому показателю  
RepInd.Limit[Report.skBegSaldo, Report.Roll] = True;  
-- указываем минимальное значение, меньше которого все показатели  
-- будут исключены из отчета  
Rep.LimitValue[Report.skBegSaldo, Report.Roll] = 1000;  
-- ограничение будет действовать только для положительных величин  
Rep.LimitKind[Report.skBegSaldo, Report.Roll] = Rep.cutAbove;
```

Описание

Владелец : [Отчет](#);
Owner : [Report](#);

Назначение

Возвращает указатель на объект класса **Отчет/Report**, т.е. на отчет, которому принадлежит данный показатель отчета.

Поле доступно только на чтение.

Пример

```
var NameRep : String;  
...  
проц НазваниеОтчета(RI :ReportIndicator);  
  -- определить имя отчета  
  NameRep=RI.Owner.Name;  
  Message("ИмяОтчета = " + NameRep);  
end;
```

Описание

ВПересчетеНа : [Признак](#);
ConvertTo : [Sign](#);

Назначение

Позволяет узнать или изменить единицу измерения, в которую будет производиться пересчет показателей в текущем измерителе.

Внимание! Поле применимо только для тех измерителей, которые являются измерителями в буквальном смысле. Как известно, показатели отчета рассчитываются по любому из параметров выбранных счетов. В программном интерфейсе все такие параметры названы измерителями. Однако часть из этих параметров являются действительно [измерителями](#) (имеют модификатор измеритель в описании типа счета), в то время как остальные могут быть простыми аналитическими справочниками или параметрами простых типов, например, строка. Если свойству присвоить nil, пересчет не выполняется, то есть показатели выводятся в натуральных единицах измерения (как есть в проводках).

Помимо указания конкретного признака измерителя с помощью данного свойства, можно также использовать свойство [ВПересчетеНаБазовые](#).

Следует иметь в виду, что после установки свойства **ВПересчетеНа**, значение свойства **ВПересчетеНаБазовые** становится равным FALSE.

Пример

```
-- фрагмент кода бланка
with InsertIndicator(3, 'Количество') do
  Caption    = 'Количество';
  CorrValue  = False;
  Option[opShowSaldo] = False;
  if (aCurrency <> nil) then
    ConvertTo = aCurrency;
  else
    ConvertToBase = True;
  end;
  OutputFmt = ofBoth;
end;
```

Описание

ВПересчетеНаБазовые : Логическое;
ConvertToBase : Logical;

Назначение

Позволяет включить/отключить режим пересчета показателей с помощью одного измерителя в базовую единицу измерения. Аналогичный режим для второго измерителя устанавливается в свойстве [ВПересчетеНаБазовыеРасш / ConvertToBaseEx](#).

Внимание! Поле применимо только для тех измерителей, которые являются измерителями в буквальном смысле. Как известно, показатели отчета рассчитываются по любому из параметров выбранных счетов. В программном интерфейсе все такие параметры названы измерителями. Однако часть из этих параметров являются действительно [измерителями](#) (имеют модификатор измеритель в описании типа счета), в то время как остальные могут быть простыми аналитическими справочниками или параметрами простых типов, например, строка.

Если свойству присвоить nil, пересчет не выполняется, то есть показатели выводятся в натуральных единицах измерения (как есть в проводках).

Данное свойство предоставляет альтернативный вариант пересчета в дополнение к свойству [ВПересчетеНа](#). Когда **ВПересчетеНаБазовые** установлено в значение TRUE, то показатели пересчитываются в базовые единицы измерения, причем система допускает одновременное существование нескольких базовых единиц в одном справочнике. Например, в справочнике единиц измерения могут быть описаны килограммы и граммы (килограммы базовые), а также литры и декалитры (литры базовые). Каждый из видов единиц измерения - весовые и объемные - имеет собственную базовую единицу и не связан никакими соотношениями с другими видами единиц.

Следует иметь в виду, что после установки свойства **ВПересчетеНаБазовые** в TRUE, значение свойства **ВПересчетеНа** не определено и будет равно nil.

Пример

```
-- фрагмент кода бланка
with InsertIndicator(3, 'Количество') do
  Caption    = 'Количество';
  CorrValue  = False;
  Option[opShowSaldo] = False;
  if (aCurrency <> nil) then
    ConvertTo = aCurrency;
  else
    ConvertToBase = True;
  end;
  OutputFmt  = ofBoth;
end;
```

Описание

```
ВПересчетеНаБазовыеРасш[Номер : Целое] : Логическое;  
ConvertToBaseEx[Number : Integer] : Logical;
```

Аргументы

Номер - номер измерителя, значение 1 соответствует первому измерителю, значение 2 - второму.

Назначение

Позволяет включить/отключить режим пересчета в базовую единицу измерения показателей, рассчитываемых с помощью двух измерителей. Если значение индекса Номер = 1, то режим устанавливается для первого измерителя, если - Номер = 2, то - для второго измерителя.

Внимание! Поле используется для вычисляемых показателей, которые рассчитываются по заданному способу. Например, для показателей, определяемых как частное от деления двух измерителей или как разность измерителей.

Назначение поле аналогично свойству [ВПересчетеНаБазовые / ConvertToBase](#), которое используется для простых показателей.

Описание

ВПересчетеНаРасш[Номер : Целое] : [Признак](#);
ConvertToEx [Number : Integer] : [Sign](#);

Аргументы

Номер - номер измерителя, указывает, для какого из двух измерителей, задана единица измерения.

Назначение

Позволяет узнать или изменить единицу измерения, в которую будет производиться пересчет показателей.

Внимание! Поле применимо только для вычисляемых показателей, которые рассчитываются по заданному алгоритму. Например, для показателей, определяемых как частное от деления двух измерителей или как разность измерителей. Если Номер = 1, то единица измерения задана для первого измерителя, если Номер = 2, то - для второго измерителя.

Назначение поля аналогично свойству [ВПересчетеНа / ConvertTo](#), которое используется для получения единицы измерения простых показателей, имеющих один измеритель.

Описание

ВПересчетеПоПараметру : Строка;
ConvertByParam : String;

Назначение

Данное поле позволяет задавать пересчет в единицы измерения, связанные с параметром разбиения. Пересчет производится по аналогии с пересчетом в базовые единицы ([см. ВПересчетеНаБазовые](#)), но информация о том, какая единица является базовой, берется из свойства параметра, по которому идет разбиение. Для случая "без параметра" и если параметр равен nil, значения остаются в натуральных единицах измерения.

Если пересчет по параметру не установлен, то **ВПересчетеПоПараметру** вернет пустую строку. Если **ВПересчетеПоПараметру** присвоить пустую строку, то пересчет устанавливается в натуральные единицы измерения.

Пример

```
ВПересчетеПоПараметру = 'Товар.ЕдИзм' ;
```

Описание

```
ВПересчетеПоПараметруРасш[Номер :Целое] : Строка;  
ConvertByParamEx[Number :Integer] : String;
```

Аргументы

Номер - номер измерителя, указывает, для какого из двух измерителей задается пересчет. Если пересчет выполняется для первого измерителя, то номер равен 1, для второго - 2.

Назначение

Данное поле позволяет узнать или задать пересчет показателя для заданного по номеру измерителя в единицы измерения, связанные с параметром разбиения.

Внимание! Поле используется для вычисляемых показателей, которые рассчитываются по заданному способу. Например, показатель определяется как частное от деления двух измерителей или как разность измерителей.

Назначение поля аналогично свойству [Поле ВПересчетеПоПараметру / ConvertByParam](#), которое позволяет задавать пересчет в единицы измерения для простых показателей, имеющих один измеритель.

Описание

Группа : [ПоказательОтчета](#);

Group : [ReportIndicator](#);

Назначение

Свойство позволяет программно установить или узнать группу, в которой находится текущий показатель. Поле возвращает ссылку на показатель, идентифицирующий группу, в которую он входит. Для корневых показателей возвращается значение **nil**.

Свойство используется в отчетах по оборотам с иерархическими показателями и доступно на чтение и запись.

Предупреждение. Уникальность имен показателей должна соблюдаться внутри всего отчета, а не только в группе.

Пример

```
-- найти "родительскую" группу самого верхнего уровня
func TestParent(RepInd: ReportIndicator) : ReportIndicator;
    if RepInd.Group <> nil then
        return TestParent(RepInd.Group);
    else
        return RepInd;
    end;
end;
```

Описание

Заголовок : Строка;
Caption : String;

Назначение

Задаёт или определяет заголовок столбца в построенном отчете для существующего показателя. Поле доступно на чтение и запись.

Пример

```
var Rep :Report;  
var RepInd :ReportIndicator;  
....  
Rep = Report.CreateName("ПоОборотам");  
RepInd = Rep.InsertIndicator(1, 'Сумма', True) ;  
RepInd.Caption = "Сумма (руб)";
```

Описание

ЗначениеОграничения [ОстОбор : [Отчет.ВидыОстОбор](#); ДебКре : [Отчет.ВыводОстОбор](#)] :Число;
LimitValue [SumKind : [Report.SumKinds](#); DebCre : [Report.SumDebCre](#)] :Numeric;

Аргументы

ОстОбор - индекс, который может принимать значение одной из предопределенных констант перечислимого типа [ВидыОстОбор](#). Он определяет, при выводе каких данных отчета накладывается ограничение: начальных остатков (skBegSaldo), оборотов (skTurn) или конечных остатков (skEndSaldo).

ДебКре - индекс, который определяет, в каком виде будут выводиться начальные и конечные остатки и обороты: в свернутом виде (roll) или в разделенном виде с показом дебетовой (byDeb) или кредитовой (byCre) части показателя. Индекс может принимать одно из значений перечислимого типа [ВыводОстОбор](#).

Назначение

Позволяет узнать и изменить минимальное пороговое значение показателя, которое должно отображаться в отчете. Более мелкие значения в отчете отображаться не будут. Один из 9 возможных способов вывода показателя в отчете, на который действует заданное ограничение, определяется комбинацией индексов данного поля.

При этом режим ограничения должен быть включен с помощью поля [Ограничить](#).

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
var RepInd : ReportIndicator;  
....  
Rep      = Report.CreateName( "ПоОборотам" );  
RepSplit = Rep.Split[Report.rdRow];  
RepInd   = Rep.InsertIndicator(1, 'Сумма', True);  
-- включаем режим исключения "мелких" свернутых оборотов  
-- по первому показателю  
RepInd.Limit[Report.skTurn, Report.Roll] = True;  
-- указываем минимальное значение, меньше которого все  
-- показатели будут исключены из отчета  
Rep.LimitValue[Report.skTurn, Report.Roll] = 1000;
```

Описание

Имя : Строка;
Name : String;

Назначение

Позволяет получить или изменить имя показателя отчета. По умолчанию при установке имени показателя таким же значением заполняется [поле Содержимое](#). Данное свойство является программным интерфейсом к полю **Имя** [страницы "Показатели"](#) диалога "Внутренние отчеты".

Замечание. Имя показателя отчета задается с помощью функций [ВставитьПоказатель](#) и [ДобавитьПоказатель](#) класса **Отчет/Report**.

Поле доступно на чтение и запись.

Пример

```
var RepIndicat :ReportIndicator;  
...  
RepIndicat.Имя = 'Сумма';
```

Описание

КоррЗначение : Логическое;
CorrValue : Logical;

Назначение

Если значение поле установлено равным TRUE, то в отчетах по оборотам будет показываться корреспондирующее значение для показателя, вычисляемого по одному измерителю, иначе - корреспондирующее значение не выдается. Действие поля аналогично флагу **Корр. значение диалога** ["Добавление показателя"](#).

Если требуется выдавать корреспондирующее значение для второго измерителя (показатель рассчитывается с использованием двух измерителей), то необходимо использовать свойство [КоррЗначениеРасш / CorrValueEx](#).

Поле доступно на чтение и запись.

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
var RepInd : ReportIndicator;  
....  
Rep      = Report.CreateName( "ПоОборотам" );  
RepSplit = Rep.Split[Report.rdRow];  
RepInd   = Rep.InsertIndicator(1, 'Сумма', True);  
RepInd.CorrValue = true;
```

Описание

```
КоррЗначениеРасш[Номер : Целое] : Логическое;  
CorrValueEx[Number : Integer] : Logical;
```

Аргументы

Номер - номер измерителя, указывает, для какого из двух измерителей, требуется показать корреспондирующее значение.

Назначение

Если значение поле установлено равным TRUE, то в отчетах по оборотам будет показываться корреспондирующее значение для измерителя, заданного по номеру. Если номер равен 1, то корреспондирующее значение выдается для первого измерителя, если номер равен 2, то для второго измерителя.

Внимание! Поле используется для вычисляемых показателей, которые рассчитываются по заданному способу с использованием двух измерителей. Например, для показателей, определяемых как частное от деления двух измерителей или как разность измерителей.

Назначение поле полностью аналогично полю [КоррЗначение / CorrValue](#), которое используется для простых показателей, вычисляемых по одному измерителю.

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
var RepInd : ReportIndicator;  
....  
Rep      = Report.CreateName("ПоОборотам");  
RepSplit = Rep.Split[Report.rdRow];  
RepInd   = Rep.InsertIndicator(1, 'Сумма', True);  
RepInd.CorrValue = true;  
RepInd.CorrValueEx[2] = true;
```


Описание

МаксШирина : Число;
MaxWidth : Numeric;

Назначение

Поле используется для настройки показателей отчета. Оно позволяет задать или определить максимальную ширину колонки показателя. Нулевое значение поля означает, что максимальная ширина колонки показателя берется из шаблона, в остальных случаях максимальная ширина колонки устанавливается в соответствии со значением данного поля.

При "растяжке" колонки пользователем рамки этой растяжки берутся из настроек шаблона. Рамки при необходимости корректируются в соответствии с полученной шириной, определенной на основе заданных пользователем параметров.

Пример

```
-- фрагмент кода бланка
with InsertIndicator(5, 'ПривязаноВВалюте', True) do
  Caption          = "привязано (вал)";
  ValueType        = varUnit;
  Format            = "##0.00;-";
  MinWidth         = 10;
  MaxWidth         = 1000;
  AggregateMethod  = agrSum;
  ShowUnit         = True;
end;
```

Описание

МаксШиринаЕдИзм : Число;
UnitMaxWidth : Numeric;

Назначение

Поле используется для настройки показателей отчета и позволяет задать или определить максимальную ширину колонки для единицы измерения.

Максимальную ширину колонки для единицы измерения можно также задать в диалоге ["Настройка вывода показателя"](#)

Пример

```
-- фрагмент кода бланка
with InsertIndicator(5, 'ПривязаноВВалюте', True) do
  Caption      = "привязано (вал)";
  ValueType    = varUnit;
  Format        = "##0.00; ;-";
  ShowUnit     = True;
  UnitMinWidth = 10;
  UnitMaxWidth = 1000;
  AggregateMethod = agrSum;
  ShowUnit     = True;
end;
```

Описание

МетодАгрегирования : [Отчет.МетодыАгрегированияПоказателя;](#)

AggregateMethod : [Report.IndicatorAggregateMethods;](#)

Назначение

Позволяет задать одну из возможных операций агрегирования параметров (agrSum, agrMin, agrMax) или отказаться от нее (agrNone), а также узнать, какой метод агрегирования используется|неиспользуется.

Поле доступно на чтение и запись, и может содержать значение перечислимого типа [МетодыАгрегированияПоказателя](#).

Метод агрегирования можно устанавливать не только для пользовательских, но и для простых показателей. Вычисление минимального (agrMin) и максимального (agrMaxmax) значения доступно для показателей типа Измеритель, Частное, Число, Целое, Дата.

При вычислении минимальных, максимальных значений *принимаются во внимание все установленные значения*, т.е. если не было движений в строке/колонке отчета или, если это пользовательский показатель и значение не было реально установлено, то это значение не учитывается.

Пример

```
-- фрагмент кода бланка
with InsertIndicator(5, 'ПривязаноВВалюте', True) do
  Caption          = "привязано (вал)";
  ValueType        = varUnit;
  Format            = "##0.00;;-";
  MinWidth         = 10;
  AggregateMethod  = agrSum;
  ShowUnit         = True;
end;
```

Описание

МинШирина : Число;
MinWidth : Numeric;

Назначение

Поле используется для настройки показателей отчета и позволяет задать или определить минимальную ширину колонки показателя. Нулевое значение поля означает, что минимальная ширина колонки показателя берется из шаблона, в остальных случаях минимальная ширина колонки устанавливается в соответствии со значением данного поля.

Пример

```
-- фрагмент кода бланка
with InsertIndicator(5, 'ПривязаноВВалюте', True) do
    Caption          = "привязано (вал)";
    ValueType        = varUnit;
    Format            = "##0.00; ;-";
    MinWidth         = 10;
    MaxWidth          = 1000;
    AggregateMethod   = agrSum;
    ShowUnit          = True;
end;
```

Описание

МинШиринаЕдИзм : Число;
UnitMinWidth : Numeric;

Назначение

Поле используется для настройки показателей отчета и позволяет задать или определить минимальную ширину колонки для единицы измерения.

Минимальную ширину колонки для единицы измерения можно также задать в диалоге ["Настройка вывода показателя"](#)

Пример

```
-- фрагмент кода бланка
with InsertIndicator(5, 'ПривязаноВВалюте', True) do
  Caption      = "привязано (вал)";
  ValueType    = varUnit;
  Format        = "##0.00; ;-";
  ShowUnit     = True;
  UnitMinWidth = 10;
  UnitMaxWidth = 1000;
  AggregateMethod = agrSum;
  ShowUnit     = True;
end;
```

Описание

Ограничить [ОстОбор : [Отчет.ВидыОстОбор](#); ДебКре : [Отчет.ВыводОстОбор](#)] :Логическое;
Limit [SumKind : [Report.SumKinds](#); DebCre : [Report.SumDebCre](#)] :Logical;

Аргументы

ОстОбор - индекс, который может принимать значение одной из предопределенных констант перечислимого типа [ВидыОстОбор](#). Он определяет, при выводе каких данных отчета устанавливается режим ограничения: начальных остатков (skBegSaldo), оборотов (skTurn) или конечных остатков (skEndSaldo).

ДебКре - индекс, который определяет, какая часть показателя будет выводиться в отчете: в свернутом виде (roll) или в разделенном виде с отдельным показом дебетовой (byDeb) или кредитовой (byCre) части. Индекс может принимать одно из значений перечислимого типа [ВыводОстОбор](#).

Назначение

Позволяет узнать или включить/отключить режим исключения из отчета показателей, значения которых меньше величин, заданных с помощью поля [ЗначениеОграничения / LimitValue](#). Режим устанавливается для конкретного способа вывода данных, который определяется заданной комбинацией индексов **ОстОбор** и **ДебКре**.

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
var RepInd : ReportIndicator;  
....  
Rep      = Report.CreateName("ПоОборотам");  
RepSplit = Rep.Split[Report.rdRow];  
RepInd    = Rep.InsertIndicator(1, 'Сумма', True);  
-- включаем режим исключения "мелких" свернутых оборотов  
-- по первому показателю  
RepInd.Limit[Report.skTurn, Report.Roll] = True;  
-- указываем минимальное значение, меньше которого все  
-- показатели будут исключены из отчета  
Rep.LimitValue[Report.skTurn, Report.Roll] = 1000;
```

Описание

Опция [Опция : [Отчет.ОпцииПоказателя](#)] : Логическое;
Option[Option : [Report.IndicatorOptions](#)]: Logical;

Аргументы

Опция - константа, описанная в перечислимом типе [Отчет.ОпцииПоказателя](#) и характеризующая одну из разрешенных возможностей (опций) вывода показателей в отчетах по проводкам.

Назначение

Поле позволяет узнать и изменить опции (особенности) вывода заданного показателя из отчета по проводкам. При значении поля, равном True, опция, указанная индексом **Опция**, будет реализована в отчете для заданного показателя, иначе - не будет.

Событие [ПриОбновленииОтчета](#) вызывается только на клетках шаблона с данными отчета, которые соответствуют показателям, для которых с помощью данного свойства включена соответствующая опция.

Предупреждение. Опцию для вывода показателя можно задать только программно, используя данное свойство **Опция**.

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
var RepInd : ReportIndicator;  
....  
Rep      = Report.CreateName( "ПоОборотам" );  
RepSplit = Rep.Split[Report.rdRow];  
RepInd   = Rep.InsertIndicator(1, 'Сумма', True);  
-- по первый показатель выводим нарастающим итогом  
RepInd.Option[Report.opShowRowTotal] = True;  
...  
Rep.Build;
```

Описание

ПоказЕдИзм : Логическое;
ShowUnit : Logical;

Назначение

Поле дает возможность управлять выводом (показывать/скрывать) единиц измерения для заданного измерителя отчета (независимо от того, установлен ли пересчет показателей в какую-либо единицу или нет).

Если значение поле установлено равным TRUE, то показатели для данного измерителя выводятся вместе с единицей измерения.

Пример

```
-- фрагмент кода бланка
with InsertIndicator(5, 'ПривязаноВВалюте', True) do
  Caption          = "привязано (вал)";
  ValueType        = varUnit;
  Format            = "##0.00;;-";
  MinWidth         = 10;
  UnitMaxWidth     = 1000;
  AggregateMethod  = agrSum;
  ShowUnit       = True;
end;
```


Описание

РодИндекс : Целое;
ParentIndex : Integer;

Назначение

Поле предназначено для чтения порядкового номера данного показателя во внутреннем списке показателей. Показатели нумеруются, начиная с 1. Поле доступно только на чтение и позволяет идентифицировать показатели по их номеру.

Пример

```
var Rep :Report;  
var RepInd :ReportIndicator;  
var aНомерПоказателя :Integer = 0;  
....  
Rep = Report.CreateName("ПоОборотам");  
RepInd = Rep.InsertIndicator(1, 'Номер', True) ;  
RepInd.Contents = '#Номер';  
aНомерПоказателя = RepInd.ParentIndex;
```

Описание

Содержимое : Строка;
Contents : String;

Назначение

Позволяет узнать или задать имя параметра счета, по которому будет рассчитываться показатель, аналитический параметр, а также пользовательский показатель. В отчетах по проводкам в качестве значения этого свойства можно использовать названия полей записей, послуживших источниками данных для аналитических справочников измерителей. При этом перед именем поля необходимо указывать символ #.

Пример

```
var Rep :Report;  
var RepInd :ReportIndicator;  
....  
Rep = Report.CreateName("ПоОборотам");  
RepInd = Rep.InsertIndicator(1, 'Номер', True) ;  
RepInd.Contents = '#Номер';
```

Описание

ТипЗначения : Целое;
ValueType : Integer;

Назначение

Позволяет определить или задать один из возможных типов значения показателя отчета, предопределенных в проекте СИС2, в противном случае выдается ошибка "Недопустимый тип значения показателя". Поле возвращает целочисленное значение типа показателя, например для показателей типа "измеритель" - 17. При задании типа показателя можно использовать как целочисленное значение, так и константу, например, varUnit - для измерителей.

Для использования констант необходимо подключить проект СИС2 в качестве подпроекта, а в код бланка включить директиву: Import СИС2 Classes Константы;

Список всех возможных констант типов переменных, предопределенных в проекте СИС2, приведен в теме: [Функция ТипПеременной / VarType](#).

Поле доступно на чтение и запись.

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
var RepInd : ReportIndicator;  
....  
Rep      = Report.CreateName("ПоОборотам");  
RepSplit = Rep.Split[Report.rdRow];  
RepInd   = Rep.InsertIndicator(1, 'Сумма', True);  
RepInd.ValueType = varUnit;
```

Описание

ТипПоказателя : [Отчет.ТипыПоказателей](#);
IndicatorType : [Report.IndicatorTypes](#);

Назначение

Позволяет определить тип показателя отчета (пользовательский, вычисляемый или простой), который задается одной из констант перечислимого типа [ТипыПоказателей/IndicatorTypes](#).

Поле доступно только на чтение.

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
var RepInd : ReportIndicator;  
....  
Rep      = Report.CreateName("ПоОборотам");  
RepSplit = Rep.Split[Report.rdRow];  
RepInd   = Rep.InsertIndicator(1, 'Сумма', True);  
RepInd.ValueType = varUnit;
```

Описание

```
ФильтрПараметров : Строка;  
ParameterFilter : String;
```

Назначение

Поле позволяет узнать или установить дополнительный фильтр по параметрам для заданного показателя. Аналогичный фильтр может быть установлен пользователем для конкретного показателя в поле **Ограничения на параметры** диалога ["Расширенная настройка показателя"](#).

Если данный фильтр не установлен, то в отчет войдут проводки в соответствии с фильтром по параметрам, заданным на весь отчет в целом в свойстве [ПараметрыФильтра / ParameterFilter](#).

Описание

`ФильтрСчетов` : Строка;
`AccountFilter` : String;

Назначение

Поле позволяет узнать или установить дополнительный фильтр по счетам на параметры проводки для заданного показателя. Аналогичный фильтр может быть установлен пользователем для конкретного показателя в поле **Ограничения на счета** диалога ["Расширенная настройка показателя"](#).

Если данный фильтр не установлен, то в отчет войдут проводки в соответствии с фильтром, заданным на весь отчет в целом в свойстве [ФильтрСчетов / AccountFilter](#).

Описание

Формат : Строка;
Format : String;

Назначение

Данное поле позволяет установить или определить формат вывода показателя (целого или действительного числа, с использованием триад или нет, с отображением незначащих нулей или нет).

Более подробное описание форматов показателей приведено [на странице "Показатели"](#) диалога "Внутренние отчеты" (поле **Формат**).

Пример

```
-- фрагмент кода бланка
with Report.AddIndicator('Сумма') do
  Caption = 'Сумма (руб.)';
  Format  = ',##0.00';
  MinWidth = 10;
  MaxWidth = 1000;
end;
```

Описание

ФорматВывода : [Отчет.ФорматыВыводаПоказателя](#);
OutputFmt : [Report.IndicatorOutputFormats](#);

Назначение

Данное свойство используется в отчетах по проводкам и позволяет узнать или установить способ вывода показателей по дебету и/или кредиту.

Значение поля для заданного показателя может принимать одну из предопределенных констант типа [Отчет.ФорматыВыводаПоказателя](#);

Пример

```
-- фрагмент кода бланка
with InsertIndicator(3, 'Количество') do
  Caption    = 'Количество';
  CorrValue  = False;
  Option[opShowSaldo] = False;
  if (aCurrency <> nil) then
    ConvertTo = aCurrency;
  else
    ConvertToBase = True;
  end;
  OutputFmt  = ofBoth;
end;
```


Описание

```
ФорматОстОбор[ОстОбор : Отчет.ВидыОстОбор] : Отчет.ФорматыОстОборПоказателя;  
SumFmt[SumKind : Report.SumKinds] : Report.IndicatorSumFormats;
```

Аргументы

ОстОбор - одна из predefined констант перечислимого типа [ВидыОстОбор](#), которая определяет способ вывода суммы в отчетах в виде оборотов, начальных и конечных остатков.

Назначение

С помощью данного поля в отчетах по оборотам определяется или устанавливается один из возможных способов отображения показателя отчета при выводе сумм (обороты, начальные или конечные остатки), характеризующих индексом **ОстОбор**.

Каждую из 3 возможных сумм вывода можно связать с одной из четырех констант перечислимого типа [ФорматыОстОборПоказателя](#). Таким образом, каждую сумму вывода в клетке шаблона можно отображать следующими различными способами:

- **в разделенном виде** - дебет и кредит, только дебет или только кредит;
- **свернутом виде** - разницу дебет-кредит, дебетовую часть, кредитовую часть;
- **не выводить** - не выводится ни один показатель.

Пример

```
var Rep :Report;  
var RepInd :ReportIndicator;  
....  
Rep = Report.CreateName("ПоОборотам");  
RepInd = Rep.InsertIndicator(1, 'Сумма', True) ;  
RepInd. Caption = "сумма (руб)";  
RepInd.SumFmt[Report.skBegSaldo] = Report.msfCre;  
RepInd.SumFmt[Report.skTurn] = Report.msfCre;  
RepInd.SumFmt[Report.skEndSaldo] = Report.msfNone;
```

Описание

ФорматПоказа : [Отчет.ФорматыПоказаПоказателя](#);
ShowFmt : [Report.IndicatorShowFormats](#);

Назначение

Поле позволяет программным путем узнать и задать, в каких колонках должен выводиться показатель (в колонках разбиения, в итоговой колонке, одновременно в обеих колонках или нигде не выводиться). Поле может содержать одно из predetermined значений перечислимого типа [Отчет.ФорматыПоказаПоказателя](#).

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
var RepInd : ReportIndicator;  
....  
Rep      = Report.CreateName("ПоОборотам");  
RepSplit = Rep.Split[Report.rdRow];  
RepInd   = Rep.InsertIndicator(1, 'Сумма', True);  
-- выводить в итоговой колонке  
Rep.ShowFmt = Report.mshTotal;
```

Описание

ФорматРазделения : [Отчет.ФорматыРазделенияПоказателя](#);
SplitFmt : [Report.IndicatorSplitFormats](#);

Назначение

Поле позволяет узнать и изменить способ разделения расщепляемых параметров проводок (одноименных параметров счетов дебета и кредита, которые могут иметь различные значения) по каждому показателю отчета.

Значение поля по указанному измерителю может принимать одну из предопределенных констант типа [Отчет.ФорматыРазделенияПоказателя](#);

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
var RepInd : ReportIndicator;  
....  
Rep      = Report.CreateName( "ПоОборотам" );  
RepSplit = Rep.Split[Report.rdRow];  
-- параметры по первому показателю всегда расщеплять  
RepInd   = Rep.InsertIndicator(1, 'Сумма', True);  
RepInd.SplitFmt = Report.spAlways;  
...  
Rep.Build;
```

Описание

ЭтоГруппа : Логическое;
IsGroup : Logical;

Назначение

Возвращает TRUE, если показатель является ссылкой на группу, т.е. он используется в отчетах, где имеются группы показателей.

Свойство используется в отчетах по оборотам с иерархическими показателями и доступно на чтение и запись.

Предупреждение. Уникальность имен показателей должна соблюдаться внутри всего отчета, а не только в группе.

Пример

Класс *РазбиениеОтчета / ReportSplit* производный от класса [Объект / Object](#), относится к группе классов, предназначенных для построения внутренних отчетов, в нем сконцентрированы все свойства, касающиеся разбиения отчета.

В частности, для построения отчетов по оборотам в данном классе предусмотрены свойства Диапазон разбиения, ПереченьДиапазона, НачЗначениеДиапазона, ШагДиапазона, ПериодДиапазона для задания интервалов разбиения по параметрам типа дата, целое и число.

Непосредственно в классе *РазбиениеОтчета / ReportSplit* определены следующие свойства:

- [Поле Владелец / Owner](#)
- [Поле Измерение / Dimension](#)
- [Поле ТипРазбиения / SplitType](#)
- [Поле Период / Period](#)
- [Поле ШагПериода / PeriodStep](#)
- [Поле Прописью / Lettering](#)
- [Поле ВыравниватьПериод / AlignPeriod](#)
- [Поле Параметры / Parameters](#)
- [Поле ПоСправочнику / ByReference](#)
- [Поле ФильтрСправочника / ReferenceFilter](#)
- [Поле ДиапазонРазбиения / SplitRange](#)
- [Поле ПереченьДиапазона / RangeEnum](#)
- [Поле НачЗначениеДиапазона / RangeBegValue](#)
- [Поле ШагДиапазона / RangeStep](#)
- [Поле ПериодДиапазона / RangePeriod](#)
- [Поле Содержание / Contents](#)
- [Поле ЗагружаемыеПоля / LoadingFields](#)
- [Поле УпорядочиватьПо / SortOrder](#)
- [Поле Упорядочивать / Sort](#)
- [Поле ЗапросПартий / SeriesQuery](#)
- [Поле ПоказыватьИтоги / ShowTotals](#)
- [Поле ПоказыватьИерархию / ShowHierarchy](#)
- [Поле ГлубинаИерархии / HierarchyDepth](#)
- [Поле ИерархияОтКорня / HierarchyFromRoot](#)
- [Поле ГруппыВначале / GroupsFirst](#)
- [Поле ИнтерактивнаяИерархия / InteractiveHierarchy](#)
- [Поле ИтогиПоГруппам / GroupTotals](#)
- [Поле ИтогиПоГруппамВЗаголовке / GroupTotalsInHeader](#)

Описание

Владелец : [Отчет](#);
Owner : [Report](#);

Назначение

Возвращает указатель на объект класса **Отчет/Report**, т.е. на отчет, которому принадлежит данное разбиение.

Поле доступно только на чтение.

Пример

```
var NameRep : String;  
...  
proc НазваниеОтчета(RS :ReportSplit);  
  -- определить имя отчета  
  NameRep=RS.Owner.Name;  
  Message("ИмяОтчета = " + NameRep);  
end;
```

Описание

ВыравниватьПериод : Логическое;

AlignPeriod : Logical;

Назначение

Переключение данного свойства между значениями TRUE и FALSE позволяет соответственно включать и отключать выравнивание дат на границу периода (суток, недели, месяца и т.д.) независимо от шага разбиения по времени.

Когда выравнивание отключено, то периоды разбиения точно соответствуют шагу разбиения. Когда выравнивание включено и при этом начальная дата не кратна целому числу шагов разбиения, то первый шаг разбиения будет урезан таким образом, чтобы второй и все последующие шаги оказались выровненными на размер шага. Например, если отчет строится, начиная с пятницы, а период разбиения на строки - неделя и выравнивание включено, то первая строка будет содержать данные за пятницу, субботу и воскресенье, вторая за всю следующую неделю с понедельника по воскресенье и так далее. Если выравнивание отключить, то каждая строка отчета будет содержать данные за 7 дней, начиная с очередной пятницы и заканчивая четвергом следующей недели.

Пример

```
var Rep :Report;  
....  
Rep = Report.CreateName("ПоОборотам");  
Rep.Split[Report.rdTab].Period = Report.byWeek;  
Rep.Split[Report.rdTab].AlignPeriod = true;  
-- выравнивание понедельно
```

Описание

ГлубинаИерархии :Целое;
HierarchyDepth :Integer;

Назначение

При настройке иерархического отчета позволяет задать количество уровней иерархии (глубину вложенных групп объектов), отображаемых в отчете.

Система не позволяет использовать иерархию по столбцам.

Пример

```
var Rep :Report;  
....  
Rep = Report.CreateName("ПоОстаткам");  
Rep.Split[Report.rdRow].ShowHierarchy = true;  
Rep.Split[Report.rdRow].HierarchyDepth = 2;
```


Описание

ГруппыВначале : Логическое;
GroupsFirst : Logical;

Назначение

При настройке иерархического отчета позволяет задать метод сортировки элементов.

Если значение поля равно TRUE, группы будут находится перед простыми элементами, в случае же FALSE группы сортируются наравне с элементами.

Пример

```
var Rep :Report;  
....  
Rep = Report.CreateName("ПоОстаткам");  
Rep.Split[Report.rdRow].ShowHierarchy = true;  
Rep.Split[Report.rdRow].HierarchyFromRoot = true;  
Rep.Split[Report.rdRow].GroupsFirst = true;
```

Описание

ДиапазонРазбиения[Индекс :Целое] : [Отчет.ДиапазоныРазбиения](#);
SplitRange[Index:Integer] [Report.SplitRangeTypes](#);

Аргументы

Индекс - номер диапазона (интервала) разбиения.

Назначение

По заданному номеру разбиения для отчета по оборотам позволяет узнать или задать способ разбиения (по маске, по перечню, по шкале) в зависимости от заданной константы перечислимого типа [ДиапазоныРазбиения](#).

Примечание. При выборе константы **byEnum** для построения отчета необходимо задать свойство [ПереченьДиапазона / RangeEnum](#). Если выбрана константа **byScale**, то, следует задавать [шаг диапазона](#) и [начальное значение диапазона](#), а при разбиении по датам еще и [период диапазона](#).

Пример

```
-- фрагмент кода бланка
with локОтчет do
  ....
  with Split[rdTab] do
    SplitType = ByParam;
    Parameters = 'Дата';
    SplitRange[1] = byEnum;
    RangeEnum[1] = Стр(01.01.2006)+ ',' +
      Стр(01.04.2006)+ ',' + Стр(01.09.2006);
  end;
  Build;
end;
end;
```

Описание

ЗагружаемыеПоля : Строка;
LoadingFields : String;

Назначение

Данное свойство позволяет задать и получить список загружаемых полей, перечисленных через запятую. Поле доступно на чтение и запись.

Данное свойство позволяет включить заданные поля в *групповое разыменование* в отчете, что позволяет сократить количество запросов к серверу данных, и тем самым ускорить работу программы. Что особенно важно при большом числе обращений к серверу данных, например, в случае, когда значением разбиения по таблице/строке/столбцу является элемент справочника или документ, и есть необходимость программно обращаться к многочисленным полям этого справочника, соответствующей записи или документа.

Описание

ЗапросПартий :Логическое;
SeriesQuery :Logical;

Назначение

Данное свойство доступно в иерархических отчетах, а также в серверной реализации отчетов, например, при использовании отчетов в типовых операциях.

Если значение поля равно True, то строится отчет по партиям. В этом случае при отключении сортировки (свойство [УпорядочиватьПо / *SortOrder*](#) равно False) элементы соответствующего разбиения отчета, т.е. строки/колонки/таблицы, будут следовать в том порядке, в котором обрабатывались проводки. При этом тип разбиения может быть любым, за исключением разбиения по времени.

Пример

```
var Rep :Report;  
....  
Rep = Report.CreateName("ПоПартиям");  
Rep.Split[Report.rdTab].SortOrder = False;  
Rep.Split[Report.rdTab].SeriesQuery = True;
```

Описание

ИерархияОтКорня : Логическое;
HierarchyFromRoot : Logical;

Назначение

При настройке иерархического отчета позволяет задать метод отсчета глубины иерархии.

Если данное свойство равно true, то иерархия строится от самого корня объектов учета (отображаемых в данном измерении). Если же свойство равно false, то глубина иерархии отсчитывается от самой высокорасположенной группы, включающей все объекты учета, попавшие в соответствии с условиями отбора в результаты отчета.

Система не позволяет использовать иерархию по столбцам.

Более подробные сведения об использовании данного свойства см. описание флага **Выводить от корня** на странице ["Иерархия"](#) диалога "Внутренние отчеты".

Пример

```
var Rep :Report;  
....  
Rep = Report.CreateName("ПоОстаткам");  
Rep.Split[Report.rdRow].ShowHierarchy = true;  
Rep.Split[Report.rdRow].HierarchyFromRoot = true;
```

Описание

Измерение : [ОтчетReport.ИзмеренияОтчета](#);
Dimension : [Report.RepDimension](#);

Назначение

Позволяет определить вид измерения (разбиение по таблицам, по строкам или по столбцам), заданный одной из констант перечислимого типа [ИзмеренияОтчета/RepDimensions](#).

Поле доступно на только чтение.

Пример

```
var Rep :Report;  
....  
  
proc P1(Sender :Button); -  
  if Rep.Split[Report.rdTab].Dimension = Report.rdTab then  
    Message("Отчет строится по таблицам");  
  end;  
end;
```

Описание

ИнтерактивнаяИерархия : Логическое;
InteractiveHierarchy : Logical;

Назначение

Данное поле управляет возможностью интерактивного раскрытия групп в результатах отчета. Поле доступно на чтение и запись. Если значение поля равно TRUE, то разрешается интерактивное раскрытия групп, иначе (False) - нет.

Более подробные сведения об использовании данного свойства см. описание флага **Раскрывать интерактивно** на странице ["Иерархия"](#) диалога "Внутренние отчеты".

Пример

```
var Rep :Report;  
....  
Rep = Report.CreateName("ПоОборотам");  
Rep.Split[Report.rdRow].ShowHierarchy = true;  
Rep.Split[Report.rdRow].HierarchyFromRoot = true;
```

Поле ИтогиПоГруппам / GroupTotals

Описание

ИтогиПоГруппам : Логическое;
GroupTotals : Logical;

Назначение

Поле позволяет изменить режим вывода итогов по группам. Если значение поля равно TRUE, для каждой группы выводится строка с общими показателями, в противном случае - нет.

Пример

```
var Rep :Report;  
....  
Rep = Report.CreateName("ПоОстаткам");  
Rep.Split[Report.rdRow].GroupTotals = true;  
Rep.Build;
```


Описание

ИтогиПоГруппамВЗаголовке :Логическое;
GroupTotalsInHeader :Logical;

Назначение

Поле позволяет выбрать один из двух режимов вывода итогов по группам: либо непосредственно в строке с названием группы, либо дополнительной строкой. Если значение поля равно TRUE, итоги выводятся в той же строке, что и заголовок группы.

Пример

```
var Rep :Report;  
....  
Rep = Report.CreateName("ПоОстаткам");  
Rep.Split[Report.rdRow].GroupTotals = true;  
Rep.Split[Report.rdRow].GroupTotalsInHeader = true;  
Rep.Build;
```

Описание

НачЗначениеДиапазона[Индекс :Целое] :Вариант;
RangeBegValue[Index:Integer] :Variant;

Аргументы

Индекс - целочисленный номер разбиения.

Назначение

Позволяет установить или узнать начальное значение диапазона (периода), за который строится отчет, для переменной типа дата, целое и число. Это свойство необходимо определять, если выбран вид разбиения [byScale](#). Кроме **начального значения диапазона** необходимо также задавать [шаг диапазона](#), а при разбиении по времени еще и [период диапазона](#).

Свойство доступно на чтение и запись, и используется в отчетах по оборотам.

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
...  
Rep = Report.CreateName("ПоОборотам");  
RepSplit = Rep.Split[Report.rdCol];  
RepSplit.Period = Report.byYear;  
RepSplit.RangeBegValue[1]=01.01.2006;  
RepSplit.RangePeriod[1] = Report.byWeek;  
RepSplit.RangeStep[1]=2;
```

Описание

Параметры : Строка;
Parameters : String;

Назначение

Позволяет узнать и установить строку отбора проводок по параметрам (аналитическим признакам, валютам, единицам измерения) для отчета. Данное поле содержит один или несколько идентификаторов параметров разбиения, имеющихся у типов счетов заданного плана счетов ([свойство ПланСчетов / AccountPlan](#)), описанного в структуре учета. Если идентификаторов несколько, они разделяются запятыми, например, "Контрагент,Товар".

Поле доступно на чтение и запись. Данное свойство используется для типа разбиения **ByParam**, которое задается в свойстве [ТипРазбиения/SplitType](#)).

Параметры разбиения задают критерий суммирования показателей в элементах указанного измерения. Так, при разбиении на строки по совокупности параметров "Контрагент,Товар" каждая строка будет содержать уникальное сочетание контрагента и товара.

Данное свойство по своему назначению аналогично полю **Параметр** на странице ["Разрезы"](#) диалога "Внутренние отчеты".

Пример

```
with Report.Split[Report.rdRow] do
  SplitType = Report.byParam;
  Parameters = 'Товар';
  ShowTotals = true;
end;
```

Описание

```
ПереченьДиапазона[Индекс :Целое] :Строка;  
RangeEnum[Index:Integer] :String;
```

Аргументы

Индекс - целочисленный номер разбиения.

Назначение

Позволяет получить или задать список диапазонов для заданного номера Это свойство необходимо определять, если в отчете по оборотам выбран вид разбиения [byEnum](#).

Свойство доступно на чтение и запись.

Пример

```
-- фрагмент кода бланка  
with локОтчет do  
    ....  
    with Split[rdTab] do  
        SplitType = ByParam;  
        Parameters = 'Дата';  
        SplitRange[1] = byEnum;  
        RangeEnum[1] = Стр(01.01.2006)+ ',' +  
            Стр(01.04.2006)+ ',' + Стр(01.09.2006);  
    end;  
    Build;  
end;  
end;
```

Описание

Период : [Отчет.ТипыРазбиенияПоВремени](#);

Period : [Report.DateSplitTypes](#);

Назначение

Позволяет определить и задать период построения отчета, который задается одной из констант перечислимого типа [ТипыРазбиенияПоВремени/DateSplitTypes](#).

Поле доступно на чтение и запись.

Пример

```
var Rep :Report;  
....  
Rep = Report.CreateName("ПоОборотам");  
Rep.Split[Report.rdRow].Period = Report.byYear;
```

Описание

ПериодДиапазона[Индекс :Целое] : [Отчет.ТипыРазбиенияПоВремени](#);
RangePeriod[Index:Integer] : [Report.DateSplitTypes](#);

Аргументы

Индекс - целочисленный номер разбиения.

Назначение

Данное свойство используется только при разбиении по времени отчета по оборотам. Свойство позволяет узнать и изменить период агрегирования данных, за который строится отчет, по времени для того измерения (таблица, строка, столбец), по которому используется разбиение по времени.

В зависимости от выбранной константы отчет может быть построен по дням, неделям, месяцам, кварталам годам, а также по часам, минутам и секундам.

В паре с данным свойством используется свойство [ШагДиапазона](#), определяющее количество периодов (например, при разбиении по недельным периодам можно задать шаг, равный двум, для того чтобы получить отчет с агрегированием данных по двухнедельным отрезкам времени).

Свойство доступно на чтение и запись, и используется в отчетах по оборотам.

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
...  
Rep = Report.CreateName("ПоОборотам");  
RepSplit = Rep.Split[Report.rdCol];  
RepSplit.Period = Report.byYear;  
RepSplit.RangeBegValue[1]=01.01.2006;  
RepSplit.RangePeriod[1] = Report.byWeek;  
RepSplit.RangeStep[1]=2;
```

Описание

ПоказыватьИерархию :Логическое;
ShowHierarchy :Logical;

Назначение

Определяет, следует ли выводить результаты в иерархическом виде для заданного измерения. Иерархическое представление по таблицам можно включить/отключить только программным способом, в то время как иерархия по строкам также включается/отключается и из диалога настройки свойств отчета.

Система не позволяет включать иерархию по столбцам.

Пример

```
var Rep :Report;  
....  
Rep = Report.CreateName("ПоОборотам");  
Rep.Split[Report.rdRow].ShowHierarchy = false;
```

Описание

ПоказыватьИтоги :Логическое;
ShowTotals :Logical;

Назначение

Поле позволяет изменить режим вывода итогов. Если значение поля равно TRUE, итоги выводятся, в противном случае - нет.

Пример

```
Report.Split[Report.rdTab].SplitType = Report.byAcc;  
  
with Report.Split[Report.rdRow] do  
    SplitType = Report.byParam;  
    Parameters = 'Товар';  
    ShowTotals = true;  
end;
```


Описание

ПоСправочнику : Логическое;
ByReference : Logical;

Назначение

Данное поле позволяет включать и отключать для указанного измерения режим построения отчета по справочнику аналитических параметров. Поле имеет смысл только для тех измерений, по которым разбиение осуществляется по аналитическому параметру. Для включения режима необходимо присвоить полю **ПоСправочнику** значение TRUE. Когда этот режим включен, в отчет попадают все значения справочника. В противном случае в отчет попадают только те элементы справочника, которые были упомянуты в соответствующих параметрах проводок в учетных данных за обрабатываемый период.

Следует иметь в виду, что если параметр в разбиении задан вместе с разыменованием (например, "Товар.Регион", где и Товар, и Регион содержат ссылки на соответствующие справочники), то режим по справочнику применяется по отношению к начальному справочнику, т.е. первому в цепочке разыменований. Так, продолжая пример с разбиением "Товар.Регион", отметим, что система будет просматривать полностью справочник товаров (а не регионов). Это означает, что в отчет не обязательно попадут все регионы. На самом деле в отчет будут включены лишь те регионы, которые встретились в свойстве Регион товара при просмотре всех элементов справочника товаров.

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
var RepSplit1 :ReportSplit;  
....  
Rep = Report.CreateName("ПоПродажам");  
RepSplit = Rep.Split[Report.rdRow];  
-- строки по датам  
Rep.Split[Report.rdRow].SplitType = Report.ByDate;  
-- каждая строка - за один день  
RepSplit.Period = Report.ByDay;  
RepSplit.PeriodStep = 1;  
RepSplit1 = Rep.Split[Report.rdCol];  
RepSplit1.Period = Report.ByParam;  
RepSplit1.Parameters = "Валюта";  
-- выводим все валюты  
RepSplit1.ByReference = true;  
Rep.Построить;  
-- ...
```

Описание

Прописью : Логическое;
Lettering : Logical;

Назначение

Данное свойство позволяет прочитать и изменить способ отображения дат в отчете. Если свойство имеет значение TRUE, даты выводятся прописью. По умолчанию свойство имеет значение FALSE.

Поле доступно на чтение и запись.

Пример

```
var Rep :Report;  
....  
Rep = Report.CreateName("ПоОборотам");  
Rep.Split[Report.rdTab].Lettering = true;
```

Описание

Содержание : Строка;
Contents : String;

Назначение

Определяет содержимое ячеек результирующей таблицы, отображающих элементы разбиения для заданного измерения. С помощью данного поля можно выбирать, какая информация об элементах разбиения (имена, описания, другие доступные характеристики) будет выводиться.

Значение поля - это строка, которая должна содержать указанные через запятую имеющиеся характеристики элементов разбиения. Для счетов и аналитических признаков минимальный набор таких характеристик содержит "Имя" ("Name") и "Описание" ("Description"), что задает соответственно вывод имени (идентификатора) и/или описания (комментария). Эти параметры являются обязательными для элементов аналитических справочников (присутствуют всегда). У счета также обязательно есть имя, а описание устанавливается (в структуре учета) с помощью ключевого слова **Title** (если его нет - описание счета эквивалентно пустой строке).

Значение данного поля не может быть пустым - хотя бы одна характеристика должна быть указана всегда. Если в указанной строке обнаружится имя несуществующей характеристики, на стадии построения отчета генерируется ошибка.

Более подробно об использовании содержания разбиения см. описание поле **Выводить** на странице ["Разрезы"](#) диалога "Внутренние отчеты".

Если в качестве параметра разбиения ([Параметры/Parameters](#)) была задана группа параметров, а не один параметр, то поле **Содержание** может содержать несколько разделенных запятыми названий характеристик различных параметров, причем перед именем характеристики необходимо писать имя самого параметра (имя параметра отделяется от имени характеристики символом '.' точка). Например: "Контрагент.Имя,Контрагент.Описание,Товар.Имя,Товар.Описание" - для каждого элемента разбиения задается вывод идентификатора контрагента, за которым следует расширенное описание контрагента, далее идет идентификатор товара и его описание. Другой пример: "Товар.Описание,Контрагент.Описание" - выводятся описания товара и контрагента. Если название параметра опущено, то подразумевается характеристика первого параметра. Например, если в поле **Параметры** параметры были записаны в виде "Контрагент,Товар" (т.е. Контрагент - первый), то первый из вышеприведенных примеров можно кратко записать как "Имя,Описание,Товар.Имя,Товар.Описание", а второй - как "Товар.Описание,Описание" (название параметра "Контрагент" везде опущено).

Пример

```
var Rep :Report;  
....  
Rep = Report.CreateName("ПоОборотам");  
if Rep.Split[Report.rdRow] = Report.ByAcc then  
    -- ячейки таблицы будут содержать имя и описание счетов  
    Rep.Split[Report.rdRow].Contents = "Имя,Описание";  
end;
```

Описание

ТипРазбиения : [Отчет.ТипыРазбиения](#);
SplitType : [Report.SplitTypes](#);

Назначение

Свойство позволяет узнать и изменить [тип разбиения](#) данных в отчете (по счетам, по параметрам, по дате и др.). Свойство доступно на чтение и запись.

Внимание! Смена типа разбиения в данном свойстве **SplitType** не приводит к изменению [типа отчета](#). Тип отчета назначается при [создании отчета](#).

Пример

```
-- Примеры задания разбиений

Report.Split[Report.rdTab].SplitType = Report.byAcc;

with Report.Split[Report.rdRow] do
  SplitType = Report.byParam;
  Parameters = 'Товар';
  ShowTotals = true;
end;

with Report.Split[Report.rdCol] do
  SplitType = Report.byDate;
  Period = Report.byYear;
  PeriodStep = 1;
  AlignPeriod = true;
end;
```

Описание

Упорядочивать : Логическое;
Sort : Logical;

Назначение

Определяет необходимость сортировки значений в указанном измерении по параметру аналитического справочника или счета. Сам параметр, по которому выполняется сортировка, задается с помощью свойства ([Содержание/Contents](#)).

Если в поле **Упорядочивать** записано значение FALSE, сортировка в заданном измерении не выполняется. При этом следует иметь в виду, что при разбиении в каком-либо измерении по времени или по проводкам, сортировка всегда производится в хронологическом порядке.

См. также свойство [УпорядочиватьПо / SortOrder](#).

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
....  
Rep = Report.CreateName("ПоОборотам");  
RepSplit = Rep.Split[Report.rdRow];  
if RepSplit = Report.ByAcc then  
    RepSplit.Contents = "Имя";  
    RepSplit.Упорядочивать = True;  
end;
```

Описание

```
УпорядочиватьПо : Строка;  
SortOrder : String;
```

Назначение

Данное свойство позволяет управлять порядком сортировки строк, столбцов, таблиц в отчетах по оборотам. Измерение, для которого устанавливается или проверяется порядок сортировки, задается с помощью индекса Изм. Строка, содержащаяся в этом свойстве, может состоять из перечисления имен произвольных атрибутов аналитики, а также полей записи. Идентификаторы разделяются друг от друга запятыми. Упорядочивать можно и по атрибутам, которые не выводятся в отчете.

Следует иметь в виду, что сортировка с указанным порядком выполняется только в том случае, если по соответствующему измерению включено свойство [Упорядочить/Sort](#).

Если свойство **SortOrder** не установлено, оно считается равным строке вывода ([_Содержание/Contents](#)).

Разрешено упорядочивание в обе стороны, для чего перед именем атрибута следует вставить символ '-' (минус) для сортировки по убыванию или символ '+' (плюс) для сортировки по возрастанию.

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
....  
Rep = Report.CreateName("ПоОборотам");  
RepSplit = Rep.Split[Report.rdRow];  
RepSplit.Contents = "Имя";  
RepSplit.Упорядочивать = True;  
RepSplit.УпорядочиватьПо = "-Дата,+Описание";
```

Описание

ФильтрСправочника :Логическое;
ReferenceFilter :Logical;

Назначение

Данное поле используется при построении отчета по оборотам с разбиением по справочнику (см. поле [ПоСправочнику/ByReference](#)). Свойство позволяет задавать и отключать фильтр на справочник. Фильтр накладывается на те элементы, которые попадают в отчет из справочника, при желании фильтр и условие на параметры отчета можно подобрать так, чтобы они были согласованными.

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
....  
Rep = Report.CreateName("ПоПродажам");  
RepSplit = Rep.Split[Report.rdCol];  
RepSplit.Period = Report.ByParam;  
RepSplit.Parameters = "Валюта";  
-- выводим все валюты  
RepSplit.ByReference = true;  
RepSplit.ReferenceFilter = false;  
  
Rep.Построить;
```

Описание

```
ШагДиапазона[Индекс :Целое] :Вариант;  
RangeStep[Index:Integer] :Variant;
```

Аргументы

Индекс - целочисленный номер разбиения.

Назначение

Позволяет узнать и изменить шаг диапазона для заданного номера разбиения. Это свойство необходимо определять, если выбран вид разбиения [byScale](#). Кроме **шага диапазона** необходимо также задавать [начальное значение диапазона](#), а при разбиении по времени еще и [период диапазона](#).

Свойство доступно на чтение и запись, и используется в отчетах по оборотам.

Пример

```
var Rep :Report;  
var RepSplit :ReportSplit;  
...  
Rep = Report.CreateName("ПоОборотам");  
RepSplit = Rep.Split[Report.rdCol];  
RepSplit.Period = Report.byYear;  
RepSplit.RangeBegValue[1]=01.01.2006;  
RepSplit.RangePeriod[1] = Report.byWeek;  
RepSplit.RangeStep[1]=2;
```


Описание

ШагПериода :Целое;
PeriodStep :Integer;

Назначение

Позволяет узнать и изменить шаг периода отчета. Свойство доступно на чтение и запись.

Пример

```
var Rep :Report;  
....  
Rep = Report.CreateName("ПоОборотам");  
  
with Report.Split[Report.rdCol] do  
  SplitType = Report.byDate;  
  Period = Report.byYear;  
  PeriodStep = 1;  
  AlignPeriod = true;  
end;
```

Все классы, производные от класса [Объект](#), в Справочной системе сгруппированы по своему назначению на группы.

В группу классов для работы с сервером данных входят следующие:

- [Ограничение / Constraint](#)
- [Запись / Record / Документ / Document](#)
- [Структура / Structure](#)
- [Подтаблица / Subtable](#)
- [Запрос / Query](#)
- [ЗапросКонфликтовРепликации / ReplConflQuery](#)
- [Транзакция / Transaction](#)
- [Изоляция / Isolation](#)
- [Экспортер / Exporter](#)
- [Импортер / Importer](#)

Класс *Запись* (*Report, Документ, Document*), производный от класса *Объект*, используется для работы с записями об элементарных операциях хозяйственной деятельности. Иными словами, это программный интерфейс к записям, хранящимся в информационных базах.




Непосредственно в данном классе определены следующие свойства:

-  [Функция Создать / Create](#)
-  [Функция Открыть / Open](#)
-  [РежимЗагрузкиПолей / LoadingFieldsMode](#)
-  [Поле ЗагружаемыеПоля / LoadingFields](#)
-  [Поле Иерархическая / Hierarchical](#)
-  [Поле Абстрактный / Abstract](#)
-  [Поле Привязано / Mapped](#)
-  [Поле ФильтрЗаписи / RecordFilter](#)
-  [Поле КоличествоПользовательскихОграничений / UserConstraintsCount](#)
-  [Поле ПользовательскоеОграничение / UserConstraint](#)
-  [Функция ДобавитьПользовательскоеОграничение / AddUserConstraint](#)
-  [Процедура УдалитьПользовательскоеОграничение / DeleteUserConstraint](#)

-  [Событие ПриОткрытииБланка / OnOpenBlank](#)
-  [Событие ПриОткрытииКартотеки / OnOpenCardfile](#)
-  [Событие ПриОписании / OnDescribe](#)

-  [Поле Состояние / State](#)
-  [Поле ВерсияЗаписи / RecordVersions](#)
-  [Поле Изменена / Modified](#)
-  [Поле Удалена / Deleted](#)
-  [Поле КлючЗаписи / DocID / КлючДокумента](#)
-  [Поле КлючЗаписиСтрока / DocIDStr / КлючДокументаСтрока](#)
-  [Поле ВнешнийКлюч / ExtID](#)
-  [Поле ЭтоГруппа / IsGroup](#)
-  [Поле ЗаписьГруппы / GroupDoc / ДокументГруппы](#)
-  [Поле ОписаниеЗаписи / RecordDescription](#)
-  [Поле НеЛогироватьДляРепликации / NoLoggingForReplication](#)
-  [Поле ПараметрЗаписи / PostHint](#)
-  [Функция Заблокировать / Lock](#)
-  [Процедура Разблокировать / UnLock](#)
-  [Процедура Редактировать / Edit](#)
-  [Процедура Записать / Post](#)
-  [Процедура Отменить / Cancel](#)
-  [Процедура Удалить / Delete](#)
-  [Процедура Восстановить / UnDelete](#)
-  [Процедура ЗагрузитьПоля / LoadFields](#)

и перечислимые типы:

-  [Тип СостоянияЗаписи / RecordStates](#)
-  [Тип ВерсииЗаписи / RecordVersions](#)
-  [Тип ПараметрыЗаписи / PostHints.](#)

Важно отметить, что необходимо различать поля-свойства класса *Запись* и поля, физически хранящие данные в записях информационной базы. Обращаем ваше внимание на тот факт, что слово "*запись*" используется и как название класса, и как объект из контекста базы данных. Некоторые поля записей (документов) имеют соответствующие одноименные поля записей (например, **DocID** - это и поле записи, и свойство объекта класса *Запись*). Некоторые поля документов являются вычисляемыми и формируются "на лету" из других полей документа.

Класс *Запись* является базовым классом языка ТБ.Скрипт для всех документов, определенных прикладным

программистом. В связи с этим все его свойства наследуются производными пользовательскими классами записей (документов). Иногда это может приводить к конфликту имен, если в описании документа в MTL-файле вводится поле записи с тем же именем, что и имя поля-свойства класса *Запись*. Например, *Запись* имеет поле-свойство **Состояние**, и если в MTL-описание записи вводится одноименное поле, то получить к нему доступ можно будет только через функцию [ВзятьПоле / GetField](#) с указанием имени поля в кавычках (например, `D.GetField("Состояние");`). Краткая запись **D.Состояние** будет адресовываться к свойству объекта **D** класса *Запись*. На стадии компиляции проекта о таких конфликтах выдается предупреждение.

Обобщим вышесказанное: объекты класса *Запись* представляют собой программную оболочку для записей информационной базы.

В документе существует несколько служебных полей, соответствующих служебным полям записи: [DocID](#), [DocIDStr](#), [ExtID](#), [ModifyDate](#), [CreateDate](#) и [некоторые другие](#). **DocID** - уникальное для текущей таблицы поле (по умолчанию, целого типа, если иное не указано в MTL-файле), которое заполняется на основе внутреннего системного счетчика документов в таблице. **DocIDStr** - строковое представление DocID, дополненное полным именем класса записи. **ExtID** - уникальный в глобальном масштабе строковый идентификатор.

Важно отметить, что для внутренних записей, хранящихся в базах данных, управляемых посредством СУБД, поля **DocID**, **ExtID** и **ModifyDate** являются обязательными (присутствуют во всех записях даже без явного описания в MTL-файле). В случае внешних записей (проецируемых в информационные базы Студии из внешних баз данных, все поля кроме **DocID** являются опциональными (могут отсутствовать)). Наличие тех или иных полей определяется описанием структуры записи, которая дается в соответствующем MTL-файле. Синтаксис и логика работы MTL-файлов приводится в главе [Язык описания модели данных](#).

Существует еще два служебных поля, которые вводятся в записи только по явному указанию в MTL-описании - это строковые поля **CreateUser** и **ModifyUser**, содержащие соответственно имя пользователя, который ее создал и последним ее редактировал.

При работе с объектами класса *Запись* и производных от него следует иметь в виду, что изменение свойств, хранящихся в базе, всегда осуществляется в рамках [транзакций](#) - неделимых операций с базой данных. Все действия из одной транзакции либо полностью выполняются (в случае успеха), либо полностью игнорируются (в случае возникновения какой-либо ошибки). Такой механизм обеспечивает целостность базы и надежность хранения данных.

Если не применять никаких дополнительных вызовов, исправление каждого свойства записи приводит к открытию и закрытию отдельной транзакции. Подобный подход целесообразно использовать только в том случае, если нужно исправить единственное поле единственной записи. Во всех остальных случаях настоятельно рекомендуется применять различные средства оптимизации быстродействия и эффективности проекта. Так, перед изменением нескольких полей одной записи следует вызвать метод [Edit](#) (это приводит к открытию транзакции), а по завершении редактирования - [Post](#) (завершение транзакции). Перед массовыми правками записей рекомендуется явным образом открывать транзакцию с помощью вызова [BeginTransaction](#), а по их завершению - закрывать ее посредством [EndTransaction](#).

См. также другие [приемы оптимизации работы с записями](#).

Перечислимый тип *ВерсииЗаписи / RecordVersions*, определенный в классе *Запись*. предназначен для задания возможных версий записи.

С помощью свойства [ВерсияЗаписи](#) запись может приводиться к одной из следующих версий, определенных в данном типе:

- **ТекущаяВерсия / CurrentVersion** - запись в ее текущем виде;
- **ПредыдущаяВерсия / PreviousVersion** - запись в том виде, какой она была до начала редактирования (перед последним вызовом **Edit** или после последнего вызова **Post**). После установки версии в данное состояние, запись возвращает значение полей на момент перед началом редактирования;
- **ПерваяВерсия / FirstVersion** - после установки версии в данное состояние запись возвращает значение полей на момент начала транзакции;
- **ЧужаяВерсия / ForeignVersion** - версия записи, хранящаяся на сервере. Поскольку клиентская система работает с локальной копией записи (в рамках изоляции), то для получения доступа к значениям полей, которые реально хранятся на сервере, необходимо установить версию в данное состояние.

Перечислимый тип *ПараметрыЗаписи* / *PostHints*, определенный в классе *Запись* предназначен для включения|выключения [пользовательских ограничений](#), ведения лога репликации и истории записи при записи документа.

В данном типе определены следующие константы, используемые при записи документа:

- **ИгнорироватьПользовательскиеОграничения / IgnoreUserConstraints** - игнорирование пользовательских ограничений;
- **ИгнорироватьСистемныеОграничения / IgnoreSystemConstraints** - игнорирование системных ограничений;
- **ИгнорироватьРепликацию / IgnoreReplication** - не ведется лог репликации;
- **ИгнорироватьИсторию / IgnoreHistory** - не ведется лог истории записи (пока не реализовано).

Константы, определенные в данном типе, используются в свойстве [ПараметрЗаписи/PostHint](#) класса *Запись*.

В классе **Запись** определен специальный тип **СостоянияЗаписи / RecordStates / СостоянияДокумента / DocumentStates**с перечислением возможных состояний записи. Объект-запись может находиться в одном из следующих состояний:

- **Нормальная / Normal / Нормальный** - в нормальном состоянии;
- **Редактируемая / Edited / Редактируемый** - в состоянии редактирования;
- **Новая / Created / Новый** - запись только что создана;
- **Дублированная / Duplicated / Дублированный** - запись только что создана путем дублирования;
- **Ошибочная / Invalid / Ошибочный** - в некорректном состоянии.

Эти константы используются в свойстве [Состояние / State](#) данного класса и в методах других классов ТБ.Скрипт.

Описание

Абстрактный :Логическое;
Abstract :Logical;

Назначение

Позволяет узнать, является ли класс записи абстрактным (т.е. описан ли он в MTL-файле с модификатором Abstract). Нельзя создавать экземпляры объектов абстрактных классов - такие классы лишь декларируют свойства, необходимые для классов-наследников.

Поле доступно только на чтение.

Пример

```
func CreateRecord(var RecordClass :Class Record) : Record;
-- если класс записи является абстрактным
if RecordClass.Abstract then
-- ищем первого наследника
if LengthOfArray(RecordClass.ChildClasses) > 0 then
    RecordClass = RecordClass.ChildClasses[1];
else
    return nil;
end;
end;
Result = RecordClass.Create;
end;
```


Описание

ВерсияЗаписи : Запись.ВерсииЗаписи;
RecordVersion : Record.RecordVersions;

Назначение

Позволяет узнать и изменить версию записи, т.е. фактически привести ее содержимое в соответствие с тем, каким оно было в определенные моменты работы с записью.

Поле может содержать одно из значений типа [ВерсииЗаписи / RecordVersions](#). В частности запись можно вернуть в состояние на момент начала ее редактирования, на момент записи транзакции или узнать изменения, внесенные в запись параллельно другими пользователями.

Пример

```
func Дельта (РабочаяЗапись :Запись) : Число;  
var БылоЗначение : Число;  
  -- возвращаем запись в предыдущее состояние  
  РабочаяЗапись.ВерсияЗаписи = Запись.ПредыдущаяВерсия;  
  -- проверяем интересующее нас поле  
  БылоЗначение = ТекущаяЗапись.Сумма;  
  -- возвращаем запись в актуальное состояние  
  РабочаяЗапись.Версия = Запись.ТекущаяВерсия;  
  Result = БылоЗначение - ТекущаяЗапись.Сумма;  
end;
```

Описание

ВнешнийКлюч : Строка;
ExtID : String;

Назначение

Содержит значение уникального в глобальном контексте идентификатора документа. Система гарантирует, что во всех где-либо установленных информационных базах не может встретиться двух документов с одним ExtID. Это свойство полезно при слиянии картотек документов из нескольких информационных баз.

Данное поле доступно только на чтение и устанавливается самой системой.

Пример

```
func CompareDocuments (D1: Record; D2:Record): Record;  
  if D1.ExtID = D2.ExtID then  
    return D1;  
  else  
    return nil;  
  end;  
end;
```

Описание

ЗагружаемыеПоля : Строка;
LoadingFields : String;

Назначение

Позволяет определить и задать поля записей для предварительной загрузки в виде строки с перечнем имен используемых полей через ";".

Указание в данной строке лишь тех полей, с которыми необходимо в текущий момент работать, дает возможность оптимизировать быстродействие программы на ТБ.Скрипт, обращающейся к записям в информационной базе. Поля записи с указанными именами будут загружаться упреждающе, сокращая тем самым время доступа при фактическом обращении к ним.

Поле-свойство **ЗагружаемыеПоля** может использоваться в комбинации с полем-свойством [РежимЗагрузкиПолей](#). Установки этих двух полей дополняют друг друга (складываются логической операцией объединения, пояснения в примере).

Свойства **РежимЗагрузкиПолей** и **ЗагружаемыеПоля** следует использовать при работе с документами, полученными:

- при открытии записи через его DocID;
- при разыменовании ссылочных полей.

Если документ получен через [Query.Current](#), то следует пользоваться одноименными свойствами класса [Запрос / Query](#).

Пример

```
Бланк "БланкРедактор1", Редактор Пример.Накладная;  
-- ... другие процедуры и функции ...  
func OpenDoc(Номер:Целое):Record;  
    var D : Record;  
    -- !!! в контексте бланка-редактора тип Record эквивалентен  
    -- типу редактируемого документа, то есть Пример.Накладная  
    D = Record.Open("{Проект.Пример.Накладная:"+Str(Номер)+ " }");  
    -- в общем случае для получения полного имени класса  
    -- документа следует использовать код вида:  
    -- Record.ClassProject+"."+Record.ClassName  
    D>LoadingFieldsMode= mdSimple + mdPeriodical;  
    -- только простые и периодические поля  
    D>LoadingFields="Позиции"; -- плюс еще один массив структур  
    Result = D;  
end;  
end -- конец описания кода бланка
```

Описание

ЗаписьГруппы : Запись;

GroupDoc : Record;

Назначение

Возвращает ссылку на документ (запись), идентифицирующий группу, в которую входит текущий документ. Если документ не входит ни в одну группу, возвращает nil.

Поле доступно и на запись, и на чтение.

Пример

```
-- найти "родительскую" группу самого верхнего уровня
func TestParent (D: Record): Record;
  if D.GroupDoc <> nil then
    return TestParent(D.GroupDoc);
  else
    return D;
  end;
end;
```

Описание

Иерархическая :Логическое;
Hierarchical :Logical;

Назначение

Позволяет узнать, является ли класс записи иерархическим (т.е. описан ли он в MTL-файле с модификатором **Hierarchical**). В объектах иерархических классов записей доступны поля **IsGroup** и **GroupDoc**. Поле доступно только на чтение.

Пример

```
proc ProcessRecord(RecordX :Record);  
  -- если класс записи является иерархическим  
  if RecordX.ClassType.Hierarchical then  
    -- делаем указанную запись групповой  
    RecordX.IsGroup = true;  
  end;  
end;
```

Описание

Изменена : Логическое;
Modified : Logical;

Назначение

Позволяет определить и установить признак модифицированности документа (записи). Обычно это свойство изменяется самой системой в процессе выполнения над документом различных операций и влияет на логику его последующей обработки. Например, в момент закрытия документа, отредактированного в бланке-редакторе, данное свойство равно TRUE. Тогда бланк выдает запрос, нужно ли сохранить документ, и если пользователь отвечает положительно, то вызывает метод [Записать / Post](#) (сохраняет документ). Если же в момент закрытия бланка в окне бланка-редактора данное свойство равно FALSE, документ просто закрывается без сохранения.

Изменять свойство можно только, когда документ находится в состоянии Edited или Created. В противном случае генерируется исключительная ситуация.

С помощью поля **Modified** программист может реализовать различные схемы обработки документов. Например, при создании нового документа, как правило, требуется инициализировать программно некоторые поля значениями по умолчанию. При этом документ переходит в состояние Edited, а свойство **Modified** становится равным TRUE. В результате, при попытке закрыть новый документ, даже если пользователь ничего в нем не менял, будет выдаваться запрос на сохранение, что не очень логично. Избежать такого поведения программы можно, установив **Modified** в FALSE сразу после инициализации требуемых полей документа.

Следует отметить, что если при изменении значения какого-либо поля записи новое значение совпадает со старым, то система не устанавливает свойство **Modified** в TRUE. Если для документа с неустановленным флагом **Modified** вызвать [Записать / Post](#), то вместо этого фактически выполнится [Отменить / Cancel](#). Последнее справедливо только для документов находящихся в режиме редактирования (**State** = Edited), а для документов со [Состояние / State](#), равным *Created* или *Duplicated*, **Post** работает как обычно.

Таким образом, если документ реально не изменился и для него вызван **Post**, запись в базу данных не производится, и поэтому у документа не будет изменено значение поля **ModifyDate**. Если же необходимо обновить значение поля **ModifyDate**, не меняя сам документ, то перед вызовом **Post** следует присвоить **Modified** значение TRUE.

Пример

```
-- обработчик события ПриОткрытии бланка-редактора
proc ПриОткрытии(Режим:Logical);

-- если документ новый, то его состояние Created
-- и Edit вызывать не надо

if (State <> Record.Created) then
    -- если открывается старый документ, переводим его
    -- в состояние редактирования
    Edit;
else
    -- инициализируем поле документа
    Валюта = "USD"; -- в результате присваивания Modified = TRUE
    -- сбрасываем флаг Modified
    Modified = FALSE;
end;

end;
```

Описание

ИнформацияОКлассе : [ИнфКлассаЗаписи](#);
ClassInfo : [RecordClassInfo](#);

Назначение

В классе **Запись|Record**, а также в наследниках этого класса поле **ClassInfo** возвращает объект RTTI-класса [ИнфКлассаЗаписи](#).

Поле доступно только на чтение.

Описание

КлючЗаписи : Вариант;
КлючДокумента : Вариант;
DocID : Variant;

Назначение

Содержит значение уникального ключа записи (идентификатора) для конкретной картотеки. С помощью идентификатора строится полностью квалифицированное имя документа (записи), необходимое при вызове некоторых функций и процедур работы с документами. Например, документ с DocID 17 будет иметь полное имя "{Тест.Пример.Накладная:17}".

Реальный тип поля зависит от его описания в MTL-файле; по умолчанию это целое.

Данное поле доступно только на чтение и устанавливается самой системой.

Пример

```
-- инициализируем новый документ  
proc УстановкаНачальныхЗначений;  
  -- изначально номер документа равен его DocID  
  Номер = Str(DocID);  
end;
```


Описание

```
КлючЗаписиСтрока : Строка;  
КлючДокументаСтрока : Строка;  
DocIDStr : String;
```

Назначение

Содержит значение уникального в контексте картотеки идентификатора записи (документа), включая также и имя класса, например, "{Тест.Пример.Накладная:17}".

Данное поле доступно только на чтение и устанавливается самой системой.

Пример

```
func Clone(Init: Document): Document;  
    var D : Document;  
    D = Init.Open(Init.DocIDStr);  
    -- ... работа с документом ...  
    return D;  
end;
```

Описание

КоличествоПользовательскихОграничений :Целое;
UserConstraintsCount :Integer;

Назначение

Возвращает количество пользовательских ограничений у класса записей.

Пример

```
proc ОбновитьСписокОграничений;  
  var I, J, K :Integer;  
  var vRecClasses :Class[] Record;  
  var vRecClass :class Record;  
  FDocClasses = [];  
  FConstraints = [];  
  vRecClasses = Record.ChildClasses(True);  
  K = 0;  
  for I = 1..LengthOfArray(vRecClasses) do  
    vRecClass = vRecClasses[I];  
    for J = 1..vRecClass.UserConstraintsCount do  
      K = K + 1;  
      FDocClasses[K] = vRecClass;  
      FConstraints[K] = vRecClass.UserConstraint[J];  
    end;  
  end;  
  СекОграничения.Count = K;  
end;
```

Описание

НеЛогироватьДляРепликации :Логическое;
NoLoggingForReplication :Logical;

Назначение

Свойство возвращает значение логического типа. Если значение поля равно True, то не ведется лог репликации при записи документа, иначе - ведется (False).

Поле доступно на чтение и запись.

Предупреждение. В ближайшем будущем свойство NoLoggingForReplication будет удалено, поэтому разработчикам настоятельно рекомендуется исключить его из кода прикладных проектов, заменив свойством [ПараметрЗаписи](#), в котором в первом параметре задана константа IgnoreReplication.

Описание

ОписаниеЗаписи :Строка;
RecordDescription :String;

Назначение

Свойство позволяет получить значение [описательного поля](#), в котором последовательно перечислены значения всех полей, указанных в Mtl-описании после ключевого слова **RecordDescription|ОписаниеЗаписи**. После каждого значения указывается разделитель, заданный в Mtl-описании. Причем, если какое-либо поле содержит пустое значение, разделитель все равно выводится.

Пример

```
Func Pl(Rec :Record): String;  
  var S: String;  
  ...  
  if (Rec <> nil) then  
    S=Rec.RecordDescription;  
    Result = S;  
  end;  
end;
```

Описание

ПараметрЗаписи [Параметр : [Запись.ПараметрыЗаписи](#)] :Логическое;
PostHint [Hint : [Record.PostHints](#)] :Logical;

Аргументы

Параметр - параметр перечислимого типа, который может принимать одно из значений, описанных в перечислимом типе [ПараметрыЗаписи / PostHints](#).

Назначение

Возвращает значение True, если при записи документа должны выполняться следующие действия для перечисленных значений заданного параметра:

- **IgnoreUserConstraints** - игнорируются [пользовательские ограничения](#);
- **IgnoreSystemConstraints** - отключаются системные ограничения;
- **IgnoreReplication** - не ведется лог репликации;
- **IgnoreHistory** - не ведется лог истории записи (пока не реализовано).

Пример

```
proc Button1OnClick(Sender :Button);
var vDoc : Пресс.ТабЖур.ТабЖур2;

if Пресс.ТабЖур.ТабЖур2.UserConstraintsCount = 0 then
  -- Добавляем пользовательское ограничение
  Пресс.ТабЖур.ТабЖур2.AddUserConstraint('Check date', 'Дата>1.1.2006',
    'Дата должна быть после 2005 года!', True, True, True, True);
end;

BeginTransaction([Пресс.ТабЖур.ТабЖур2]);
try
  vDoc = Пресс.ТабЖур.ТабЖур2.Create;
  vDoc.PostHint[vDoc.IgnoreUserConstraints] = True;
  -- Отключаем пользовательское ограничение
  vDoc.Post;
  EndTransaction;
except
  AbortTransaction;
  raise;
end;
end;
```

Описание

ПользовательскоеОграничение[Индекс :Целое] : [Ограничение](#);
UserConstraint[Index :Integer] : [Constraint](#);

Аргументы

Индекс - порядковый номер ограничения для конкретного класса.

Назначение

Возвращает ссылку на ограничение по заданному номеру.

Пример

```
proc Button2OnClick(Sender :Button);
  ConstraintsName = edName.Text;
  ConstraintsFilter = edConstraint.Text;
  ConstraintsMessage = edMessage.Text;
  aOnPost = cbPost.State;
  aOnEdit = cbEdit.State;
  aOnDelete = cbDelete.State;
  aOnUndelete = cbUndelete.State;
  if Trim(ConstraintsName) = '' then
    Message("Необходимо указать имя ограничения!");
    Return;
  end;
  if Trim(ConstraintsFilter) = '' then
    Message("Необходимо указать ограничение!");
    Return;
  end;
  if not aOnPost and not aOnEdit and not aOnDelete
    and not aOnUndelete then
    Message("Необходимо указать хотя бы одно событие, "+
      " при котором сработает ограничение!");
    Return;
  end;
  if IsEditing then
    with SelectedClass.UserConstraint[CurrentConstraintIndex] do
      Name = ConstraintsName;
      Constraint = ConstraintsFilter;
      Message = ConstraintsMessage;
      OnPost = aOnPost;
      OnEdit = aOnEdit;
      OnDelete = aOnDelete;
      OnUndelete = aOnUndelete;
    end;
  else
    SelectedClass.AddUserConstraint(
      ConstraintsName,
      ConstraintsFilter,
      ConstraintsMessage,
      aOnPost,
      aOnEdit,
      aOnDelete,
      aOnUndelete);
  end;
  Close(cmOk);
end;
```

Описание

Привязано :Логическое;
Mapped :Logical;

Назначение

Возвращает значение True, если записи сопоставлена таблица в базе данных, иначе - False.

Поле доступно только на чтение.

Пример

```
Func F1(Rec:Record) :Logical;  
  if Rec.Mapped then  
    Message("Запись привязана к таблице БД.");  
    Result = True;  
  else  
    Result = False;  
  fi;  
end;
```

Описание

РежимЗагрузкиПолей : Целое;
LoadingFieldsMode : Integer;

Назначение

Позволяет определить и изменить режим предварительной загрузки полей записи. Код режима может содержать произвольную комбинацию следующих значений:

- **mdAll** - загружать все поля
- **mdNone** - не загружать поля
- **mdSimple** - загружать простые поля
- **mdPeriodical** - загружать периодические поля
- **mdStruct** - загружать периодические структуры
- **mdArrays** - загружать массивы
- **mdStructArrays** - загружать массивы структур
- **mdBlob** - загружать большие поля ("большие двоичные объекты")

Эти константы определены в служебном проекте СИС, для их подключения необходимо использовать СИС как подпроект текущего проекта, а в начале модуля - написать директиву

```
Import СИС Classes Константы
```

По умолчанию (если в системном Реестре не указано иное) режим изначально равен mdSimple.

Применение различных режимов дает возможность оптимизировать быстродействие программы на ТБ.Скрипт, обращающейся к записям в информационной базе. Поля записи, отвечающие заданному режиму, будут загружаться упреждающе, сокращая тем самым время доступа при фактическом обращении к ним.

Поле-свойство **РежимЗагрузкиПолей** может использоваться в комбинации с полем-свойством [ЗагружаемыеПоля / LoadingFields](#). Установки этих двух полей дополняют друг друга (складываются логической операцией объединения, пояснения в примере).

Свойства **РежимЗагрузкиПолей** и **ЗагружаемыеПоля** следует использовать при работе с документами, полученными:

- при открытии записи (документа) с помощью DocID;
- при разыменовании ссылочных полей.

Последний случай означает обращение к записи (документу (записи) по ссылке на него. Например, в программе может быть описана переменная

```
var D : Пример.Накладная;
```

которая способна хранить ссылку на любой документ указанного типа и, после присваивания ей корректного значения, обеспечивает доступ к полям документа:

```
var T : Date;  
T = D.CreateDate;
```

Если документ получен через [Query.Current](#), то следует пользоваться одноименными свойствами класса [Запрос / Query](#).

Пример

```
Бланк "БланкРедактор1", Редактор Пример.Накладная;  
-- ... другие процедуры и функции ...  
func OpenDoc(Номер:Целое):Record;  
  var D : Record;  
  var sProject : String;  
  var sDocType : String;  
  -- !!! в контексте бланка-редактора тип Record эквивалентен  
  -- типу редактируемого документа, то есть Пример.Накладная  
  sProject = Record.ClassProject;  
  sDocType = Record.ClassName;  
  -- открываем "{Проект.Пример.Накладная:№}"  
  D = Record.Open("{ "+sProject + " "+"."+sDocType+ " ":" "+Str(Номер)+ " }");
```



```
D.LoadingFieldsMode= mdSimple + mdPeriodical;  
-- только простые и периодические поля  
D.LoadingFields="Позиции"; -- плюс еще один массив структур  
Result = D;  
end;  
end -- конец описания кода бланка
```

Описание

Состояние : `Запись.СостоянияЗаписи`;
State : `Record.RecordStates`;

Назначение

Позволяет определить текущее состояние объекта **Запись**. Поле доступно только на чтение и может содержать одно из значений типа [СостоянияЗаписи / RecordStates](#). В частности документ (запись) может находиться в состояниях создания, просмотра, редактирования, некорректном.

Если состояние записи (документа) равно Edited, это означает, что либо пользователь, либо сама программа пытались изменить и, возможно, изменили какое-либо поле документа. Для того чтобы узнать, было ли поле (или поля) действительно изменены, необходимо проверить свойство [Изменена / Modified](#). Если документ (запись) имеет состояние Edited, он будет записан в информационную базу при последующем вызове метода [Записать / Post](#) (если он будет).

Состояние Created является частным случаем Edited. Проверка состояния документа на значение Created позволяет на программном уровне определить, является ли документ новым, и присвоить некоторые поля по умолчанию.

Перевести документ (запись) из одного состояния в другое может как пользователь (например, в результате вызова документа на редактирование в бланке-редакторе), так и прикладная система. В частности, вызов метода [Редактировать / Edit](#) переводит документ в состояние Edited, а метода [Записать / Post](#) - в состояние Normal.

В сочетании с данным свойством бывает полезно использовать свойство [Изменена / Modified](#).

Пример

```
proc InitRecord (aRecord :Record);
-- Устанавливаем начальные значения в новом документе
  if (aRecord.State = Record.Created) then
    aRecord.Валюта = "РУБ";
    aRecord.Контрагент = "Население";
  end;
end;
```

Описание

Удалена : Логическое;
Deleted : Logical;

Назначение

Содержит значение TRUE, если документ (запись) помечен как удаленный. Такой документ может быть при необходимости восстановлен, другие операции с ним запрещены (например, вызов [Редактировать / Edit](#) для удаленного документа генерирует исключение). Помеченные к удалению документы физически удаляются [мастером сборки мусора](#), который вызывается командой [Сборка мусора](#) из окна администрирования.

Данное поле доступно только на чтение.

Пример

```
-- переменная D ссылается на некий документ
if D.Deleted then
    D.Undelete;
end;
```

Описание

`ФильтрЗаписи` : Строка;
`RecordFilter` : String;

Назначение

Свойство позволяет устанавливать постоянный фильтр на класс записей. Этот фильтр будет добавляться ко всем запросам на данный тип документов.

Для того чтобы постоянный фильтр, установленный на класс записей, был проигнорирован в запросе, нужно в коде бланка записать

[Query](#).OpenHint[Query.IgnoreRecordFilter]=True;

или

Запрос.ПараметрОткрытия[Запрос.ИгнорироватьФильтрЗаписи] = True;

По умолчанию значение поля **ПараметрОткрытия** равно False.

Описание

ЭтоГруппа : Логическое;

IsGroup : Logical;

Назначение

Возвращает TRUE, если запись является ссылкой на группу. Такая ситуация возможна, если картотека обладает свойством иерархичности ([см. описание языка MTL](#)), то есть допускает создание в ней групп записей.

Поле доступно и на запись, и на чтение.

Пример

```
proc InitNew (D:Record );
  if D.IsGroup <> TRUE then
    D.Валюта = "USD";
    D.Контрагент = "Население";
  else
    -- ничего не инициализируем в записи-группе
  end;
end;
```

Процедура Восстановить / UnDelete

Описание

Восстановить;
UnDelete;

Назначение

Восстанавливает запись, помечает ее как удаленную.

Пример

```
proc DocRevert (aRecord :Record);  
  if aRecord.Deleted then  
    aRecord.UnDelete;  
  end;  
end;
```

Описание

```
ЗагрузитьПоля (Режим :Целое; ИменаПолей :Строка);  
LoadFields (Mode :Integer; FieldNames :String);
```

Аргументы

Режим - режим загрузки указанных полей; допустимыми значениями являются константы или их комбинации (см. [РежимЗагрузкиПолей / LoadingFieldsMode](#));

ИменаПолей - имена полей или маски, разделенные символом "точка с запятой".

Назначение

Вызывает загрузку незагруженных ранее полей записи, указанных поименно с помощью параметра **ИменаПолей**. Операция выполняется одним SQL-запросом.

Формат заполнения параметра **ИменаПолей** см. в [Query.LoadingFields](#).

Пример

```
-- процедура в бланке редакторе  
proc ПриНажатии_КнопкаПодробности(Подробности :Кнопка);  
    -- загружаем поля структуры  
    Document.LoadFields(mdAll, "ПодтаблицаРеквизитов.*")  
    -- выводим реквизиты  
    -- ...  
end;
```

Описание

Записать ;
Post ;

Назначение

Записывает документ в базу и переводит его в [состояние](#) *Normal*. Данный метод нельзя повторно применять к документу, уже находящемуся в состоянии просмотра.

При выполнении **Post** документ автоматически разблокируется (выполняется внутренний вызов [UnLock](#)).

Пример

```
proc RecordPost (aRecord :Record);  
  -- Записывает изменения в картотеку, если документ находится в  
  -- измененном состоянии  
  if (aRecord.State = Record.Edited) or  
      (aRecord.State = Record.Created) then  
    aRecord.Post;  
  end;  
end;
```


Описание

Отменить ;
Cancel ;

Назначение

Отменяет сделанные в модифицированном документе (записи) изменения, возвращая его поля в первоначальное состояние.

Данный метод можно применять только к документу, находящемуся в [состояниях](#) *Edited* или *Created*. Если состояние документа было *Edited*, оно меняется на *Normal*. Состояние *Created* остается без изменений.

При выполнении **Cancel** документ автоматически разблокируется (выполняется внутренний вызов [Разблокировать / UnLock](#)).

Пример

```
proc RecordCancel (aRecord :Record);  
  -- Отменяет изменения в документе, если документ находится в  
  -- измененном состоянии  
  if (aRecord.State = Record.Edited) or (aRecord.State = Record.Created) then  
    aRecord.Cancel ;  
  end;  
end;
```

Описание

Разблокировать ;
UnLock ;

Назначение

Разблокирует документ (запись) в базе и уменьшает счетчик его блокировок на единицу. Если счетчик блокировок стал равным нулю, с документом могут работать другие пользователи.

Следует иметь в виду, что документ также автоматически разблокируется при вызове [Отменить/Cancel](#) или [Записать/Post](#).

Пример

```
proc Pl(D:Record);
try
  D.Lock;
except
  Message("Попробуйте еще раз через 5 минут");
  return;
end;

-- ... работа с документом ...

D.Unlock;
end;
```

Описание

Редактировать ;
Edit ;

Назначение

Переводит запись в [состояние](#) редактирования (*Edited*). Данный метод нельзя повторно применять к записи, уже находящейся в состоянии редактирования.

При выполнении **Edit** запись автоматически блокируется, выполняется внутренний вызов [Lock](#)(TRUE).

Рекомендуется переводить запись в состояние редактирования, когда требуется изменить сразу несколько ее полей, поскольку это помогает оптимизировать быстродействие системы.

Например, в процедуре

```
proc P1(D:Record);  
  D.Name = "ABC";  
  D.Amount = 150.0;  
  D.Address = "Moscow, Russia";  
end;
```

при модификации каждого поля система будет делать внутренний вызов пары процедур [Редактировать/Edit](#) и [Записать/Post](#).

Фактически это означает полноценную, возможно сетевую, транзакцию с перезаписью всего документа, повторенную в данном случае три раза.

Вышеприведенный код можно оптимизировать следующим образом:

```
proc P1(D:Record);  
  D.Edit;  
  D.Name = "ABC";  
  D.Amount = 150.0;  
  D.Address = "Moscow, Russia";  
  D.Post;  
end;
```

В этом случае будет выполнена лишь одна транзакция при вызове завершающего **Post**.

Пример

```
proc RecordEdit (aRecord :Record);  
  -- Переводит запись в состояние редактирования  
  if (aRecord.State <> Record.Edited) and  
    (aRecord.State <> Record.Created) then  
    aRecord.Edit;  
  end;  
end;
```

Описание

```
Удалить ([СПроверкой:Логическое]);  
Delete ([DoCheck:Logical]);
```

Аргументы

СПроверкой - выражение логического типа, определяющее нужно ли выполнять проверку ссылочной целостности (TRUE) или нет (FALSE) перед удалением; если параметр опущен, он предполагается равным FALSE.

Назначение

Помечает запись как удаленную. Физически она будет удалена из картотеки только при следующем вызове [мастера сборки мусора](#) командой [Сборка мусора](#) из окна администрирования. Удаленные записи по умолчанию не видны в окне картотеки, но при [соответствующих настройках](#) их можно сделать видимыми наравне с остальными записями.

Если в процедуру был передан параметр TRUE, то перед выполнением операции система проверяет, нет ли в базе других записей, ссылающихся на данную, и если такие есть, то удаление не происходит. В этом случае генерируется исключение и, по умолчанию, выводится соответствующее сообщение об ошибке.

Нельзя удалить запись, находящуюся в состоянии редактирования.

Пример

```
Пример  
proc DocDelete (aRecord :Record);  
  if (aRecord.Deleted = FALSE) then  
    aRecord.Delete;  
  end;  
end;
```

Описание

УдалитьПользовательскоеОграничение [Индекс :Целое];
DeleteUserConstraint [Index :Integer];

Аргументы

Индекс - порядковый номер ограничения, которое требуется удалить.

Назначение

Удаляет ограничение по заданному индексу.

Пример

```
func FindConstraint(ARecClass :Class Record;  
    AConstraint :Constraint) :Integer;  
var I :Integer;  
for I = 1..ARecClass.UserConstraintsCount do  
    if ARecClass.UserConstraint[I] = AConstraint then  
        return I;  
    end;  
end;  
end;  
  
proc ButDelete_OnClick(Sender :Button);  
var vRecClass :class Record;  
var vConstraint :Constraint;  
var vIndex :Integer;  
if Template.CurrentSection = СекОграничения then  
    vRecClass = FDocClasses[Template.Frame];  
    vIndex = FindConstraint(vRecClass, FConstraints[Template.Frame]);  
    if vIndex > 0 then  
        vRecClass.DeleteUserConstraint(vIndex);  
        ОбновитьСписокОграничений;  
        Template.Frame = Template.Frame;  
    end;  
end;  
end;
```

Описание

ПриОписании :Строка;
OnDescribe :String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события **OnDescribe**.

Обработчик

```
func OnDescribe (Record :Record; FullDescribe :Logical) :String;
```

Параметры:

Record - ссылка на запись;

FullDescribe - логический параметр, определяющий место наступления события.

Событие вызывается:

- для диалогов запросов подтверждений (FullDescribe = True);
- перед началом операции Drag&Drop(FullDescribe = False);
- для показа описания в диалоге свойств документа.

Событие *не вызывается при обращении к скриптовому свойству* [RecordDescription](#).

Событие **OnDescribe** установить можно только программно, например, в Profile.Init (файл Profile.cod). Напоминаем, что метод [Init](#) вызывается системой автоматически в самом начале загрузки проекта.

Событие предназначено для того чтобы формировать строку-описание документа для разных сообщений в тех сложных случаях, когда [описательных полей](#) недостаточно.

Запрос подтверждения при закрытии бланка-редактора теперь включает не заголовок бланка, а описание записи (при необходимости вызывается событие OnDescribe).

Пример

```
func ПриОписанииТовара(ADoc :Справочники.Товар;  
  AFull :Boolean) :String;  
  if AFull then  
    if ADoc.IsGroup then  
      Result = "Группа: " + ADoc.Название;  
    else  
      Result = "Товар: " + ADoc.Название;  
    end;  
  else  
    Result = ADoc.Название;  
  end;  
end;
```

Описание

```
ПриОткрытииБланка :Строка;  
OnOpenBlank :String;
```

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции- обработчика события, которое происходит в момент открытия документа в бланке редакторе.

Обработчик

```
func OnOpenBlank (var Record :Record; WindowStyle :WindowStyles) :Integer;
```

Параметры:

Record - ссылка на запись;

WindowStyle - стиль окна, predeterminedная [константа](#), которая определяет способ открытия бланка.

Для любого документа данное событие позволяет перекрыть момент открытия его в бланке редакторе. Событие вызывается из любого места в программе (из списка найденных документов, из отчета как уточнение и т.п.)

Событие может устанавливаться только программно. Наиболее естественное место для его установки - Profile.Init (файл Profile.cod). Напоминаем, что метод [Init](#) вызывается системой автоматически в самом начале загрузки проекта.

Пример

```
proc Init;  
  Документы.Накладная.OnOpenBlank =  
    "Profile.Init.ПриОткрытииНакладной";  
end;
```

Описание

```
ПриОткрытииКартотеки :Строка;  
OnOpenCardfile :String;
```

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события **OnOpenCardfile**, которое происходит в момент открытия картотеки документа.

Обработчик

```
func OnOpenCardfile (var Record :Record; Filter :String; Window.WindowStyle :  
Window.WindowStyles) :Integer;
```

Параметры:

Record - ссылка на запись картотеки, которая выделена в бланке-редакторе картотеки;

Filter - строковое выражение, определяющее условия отбора записей (документов), попадающих в картотеку.

WindowStyle - предопределенная [константа](#), которая определяет способ открытия картотеки.

Для любого документа событие позволяет перекрыть момент открытия его картотеки. Событие вызывается из любого места в программе (из списка найденных документов, из отчета как уточнение и т.п.).

Событие может устанавливаться только программно. Наиболее естественное место для его установки - Profile.Init (файл Profile.cod). Напоминаем, что метод [Init](#) вызывается системой автоматически в самом начале загрузки проекта.

Описание

ДобавитьПользовательскоеОграничение(Имя :Строка; {Ограничение :Строка}; {Сообщение :Строка}; {ПриЗаписи :Логическое}; {ПриИзменении :Логическое}; {ПриУдалении :Логическое}; {ПриВосстановлении :Логическое}) :[Ограничение](#);

AddUserConstraint(Name :String, {Constraint :String}, {Message :String}, {OnPost :Logical}, {OnEdit :Logical}, {OnDelete :Logical}, {OnUndelete :Logical}) :[Constraint](#);

Аргументы

Имя - имя, по которому можно идентифицировать ограничение;

Ограничение - ограничение, которое накладывается на запись при наступлении заданного события;

Сообщение - текст сообщения, выдаваемый пользователю при нарушении условия ограничения;

ПриЗаписи, ПриИзменении, ПриУдалении, ПриВосстановлении - необязательные параметры, характеризующие событие, при котором выполняется ограничение. По умолчанию значение этих параметров равно False, т.е. при наступлении заданного события ограничение не действует. Для выполнения указанного ограничения нужному параметру следует присвоить значение True.

Назначение

Добавляет новое ограничение с указанными параметрами и возвращает объект класса [Constraint](#).

Пример

```

proc ОбновитьСписокОграничений;
  var I, J, K :Integer;
  var vRecClasses :Class[] Record;
  var vRecClass :class Record;
  FDocClasses = [];
  FConstraints = [];
  vRecClasses = Record.ChildClasses(True);
  K = 0;
  for I = 1..LengthOfArray(vRecClasses) do
    vRecClass = vRecClasses[I];
    for J = 1..vRecClass.UserConstraintsCount do
      K = K + 1;
      FDocClasses[K] = vRecClass;
      FConstraints[K] = vRecClass.UserConstraint[J];
    end;
  end;
  СекОграничения.Count = K;
end;

proc ButAdd_OnClick(Sender :Button);
  var vRecClass :class Record;
  if ChooseClass(Record, vRecClass, 'Выберите класс') = 1 then
    vRecClass.AddUserConstraint(NewName(vRecClass), "",
      "Message", True, True, True, True);
    ОбновитьСписокОграничений;
  end;
end;
```

Функция Заблокировать / Lock

Описание

```
Заблокировать ([Ждать:Логическое]) : Логическое;  
Lock ([Wait:Logical]): Logical;
```

Аргументы

Ждать - логический параметр, влияет на логику работы функции лишь в том случае, если блокируемая запись уже заблокирована другим пользователем. В этом случае аргумент **Ждать** определяет, следует ли ждать, когда запись будет разблокирована. Если параметр равен TRUE, система выведет на экран диалоговое окно с предложением подождать, когда другой пользователь закончит работу с требуемым документом. При этом функция не возвращает управление в вызвавший ее фрагмент кода до тех пор, пока не будет снята блокировка или локальный пользователь не нажмет кнопку **Отмена**.

Если параметр равен FALSE, функция проверяет, можно ли блокировать запись и, если - да, то блокирует ее, но в любом случае (даже если запись не удалось заблокировать) управление тут же возвращается в вызвавшую процедуру/функцию. Когда параметр опущен, он подразумевается равным TRUE.

Назначение

Блокирует запись в базе и увеличивает счетчик ее блокировок на единицу. После этого с заблокированной записью можно работать только с данного рабочего места.

Следует иметь в виду, что запись также автоматически блокируется при вызове метода [Открыть / Open](#) и разблокируется при вызове [Отменить / Cancel](#) или [Записать / Post](#).

Если запись удалось заблокировать, функция возвращает TRUE.

В противном случае либо возвращается FALSE (если **Ждать** было равно FALSE), либо генерируется исключительная ситуация (если **Ждать** было равно TRUE).

Пример

```
proc P1(D:Record);  
try  
  D.Lock;  
except  
  Message("Попробуйте еще раз через 5 минут");  
  return;  
end;  
  
-- ... работа с записью ...  
  
D.Unlock;  
end;
```

Описание

Открыть (Идентификатор:Строка): Запись;
Open (ID: String): Record;

Аргументы

Идентификатор - строка с полностью квалифицированным названием класса, производного от класса **Запись**, включающая также и DocID, например, "{Проект.Пример.Накладная:7}".

Назначение

Создает объект **Документ** на основе данных, хранящихся в указанной записи. Если записи с запрашиваемым идентификатором не существует, создается новый объект класса **Запись**, но в некорректном состоянии. Последующие обращения к такому объекту будут генерировать исключение.

Пример

```
Бланк "Созидатель";

-- ... другие процедуры и функции ...

func OpenDoc(DocType: Class Record; Номер:Целое) :Record;
var sProject : String;
var sDocType : String;
sProject = DocType.ClassProject;
sDocType = DocType.ClassName;
-- открываем, например, "{Проект.Пример.Накладная:№}"
Result = DocType.Open("{" + sProject + "." + sDocType + ":" + Str(Номер) + "}");
end;

end -- конец описания кода бланка
```

Описание

Создать: Запись;

Create: Record;

Назначение

Создает новый документ (запись) некоторого типа, который определяется контекстом вызова данной функции (при этом новая запись в информационной базе пока не создается). Например, если функция вызывается из кода бланка, который является редактором документа "Накладная", то создается документ "Накладная". Если функция вызывается из кода, не связанного с документом (то есть, если это класс, описанный с помощью ключевого слова Класс (см. Главу [Общие принципы программирования на ТБ.Скрипт](#)), или бланк, не являющийся редактором какого-либо документа, функция генерирует исключительную ситуацию.

В новом документе инициализируются лишь [служебные поля](#), в том числе и поля CreateUser, ModifyUser, если они введены в структуру документа.

Пример



```
Бланк "БланкРедактор1", Редактор Пример.Накладная;
-- процедура обработки нажатия кнопки Button1 на шаблоне бланка
proc Button1Pressed(Об:String);
  var D : Record;
  -- !!! в контексте бланка-редактора тип Record эквивалентен
  -- типу редактируемого документа, то есть Пример.Накладная
  D = Record.Create;
  message(D.docidstr);
  -- получим строку вида "{Проект.Пример.Накладная:7}"
  D=nil; -- это лишь демонстрация, поэтому уничтожаем документ
end;
end -- конец описания кода бланка
```

Класс *Запрос (Query)*, производный от родительского класса [Объект](#), предназначен для построения запросов к информационной базе, которые представляют собой динамически генерируемые подборки документов, отвечающих заданным условиям.

В классе *Запрос* имеются следующие свойства:

-  [Функция Создать / Create](#)
-  [Поле Записи / Records](#)
-  [Поле Фильтр / Filter](#)
-  [Поле Упорядочивание / Order](#)
-  [Поле ВключатьУдаленные / IncludeDeleted](#)
-  [Поле РежимИерархии / HierarchyMode](#)
-  [Поле ТекущаяГруппа / CurrentGroup](#)
-  [Поле РежимЗагрузкиПолей / LoadingFieldsMode](#)
-  [Поле ЗагружаемыеПоля / LoadingFields](#)
-  [Поле РазмерПакета / PacketSize](#)
-  [Поле ПараметрОткрытия / OpenHint](#)
-  [Поле ВКонец / EOF](#)
-  [Поле ВНачале / BOF](#)
-  [Поле Активен / Active](#)
-  [Поле Текущий / Current](#)
-  [Поле Количество / Count](#)
-  [Процедура Выбор / Select](#)
-  [Процедура Закреть / Close](#)
-  [Процедура Первый / First](#)
-  [Процедура Последний / Last](#)
-  [Процедура Следующий / Next](#)
-  [Процедура Предыдущий / Previous](#)
-  [Процедура ПереместитьНа / MoveBy](#)
-  [Функция Найти / Find](#)
-  [Функция НайтиБлижайший / FindNearest](#)
-  [Функция НайтиСледующий / FindNext](#)
-  [Функция ВычислитьСоставные / CalcAggregates](#)
-  [Функция ПопадаетПодФильтр / MatchFilter](#)

и перечислимые типы:

-  [Тип ПараметрыОткрытия / OpenHints.](#)
-  [Тип РежимыИерархии / HierarchyModes.](#)

В классе *Запрос* определен специальный перечислимый тип *ПараметрыОткрытия / OpenHints*, содержащий константы для задания оптимизирующих параметров запроса. Значения этого типа используются при установке свойства [ПараметрОткрытия / OpenHint](#).

В типе *ПараметрыОткрытия* определены следующие значения:

- **БольшойРезультат / LargeResult** - указывает, что в результате запроса может получиться большой объём данных. Влияет на работу с ограниченным набором СУБД, в частности - с MS SQL, Oracle. В случае, если `LargeResult = True`, а `Packeting, AlwaysPacketing = False`, то для MS SQL будет открыт запрос с серверным курсором, и на клиент будут переданы первые N записей, количество которых регулируется свойством [PacketSize](#) (по умолчанию 60);
- **Пакетировать / Packeting** - указывает, что необходимо брать записи из СУБД несколькими запросами с ограничением количества записей в запросе, определяется в свойстве **PacketSize** (по умолчанию 60). Используется только при включенном режиме **LargeResult** и влияет на работу с ограниченным набором СУБД, в частности - с MS SQL, Oracle.
- **ВсегдаПакетировать / AlwaysPacketing** - свойство имеет смысл при сортировке по ссылочным полям. *Рекомендуется использовать это свойство только в карточках.*

Внимание. Если свойство `ВсегдаПакетировать=True` и сортировка выполняется по ссылочным полям, запрос становится некорректным, т.е. возможны случаи, когда при выборке следующего пакета, в нем окажется запись, которая уже есть на клиенте. Это происходит, если запись (документ), на которую ссылались, была изменена другим пользователем, но нотификация об изменении с сервера не пришла, так как запрос открыт не на этот тип записи. В этом случае запись будет проигнорирована и на клиенте останется старая версия записи. Также возможен и другой случай, когда при выборке какая-то запись не попадет к клиенту по той же самой причине (запись, на которую ссылаемся, была изменена).

- **ОптимальноеЧтениеПодтаблиц / OptimReadSubtables** - используется только для не пакетизируемых запросов. В Paradox игнорируется. В том случае, когда в запросе присутствуют поля подтаблицы, то на сервере данных одновременно с запросом на шапку документа будут открыты запросы на подтаблицы, которые указаны в полях подзагрузки `LoadingFields`. Использование свойства совместно со свойством **LargeResult** в некоторых случаях позволяет более эффективно обрабатывать документы прикладными алгоритмами. Т.к. запросы открываются один раз на шапку документа и на подтаблицы, а затем, по мере выбора запроса, на клиенте отправляются пакетами записей. Размер пакета регулируется свойством **PacketSize** (по умолчанию 60).
- **ШирокийРезультат / WideResult** - используется для оптимизации SQL-запросов в режиме пакетирования. Влияет на работу с ограниченным набором СУБД, в частности - с MS SQL. Когда `WideResult = True`, то сервер данных сначала формирует запрос на список DocID, а затем формирует запрос для получения документов.

Пример:

```
SELECT TOP 101 [DocID] FROM [Процесс] WHERE ... ORDER BY ...  
SELECT [Field1], [Field2]... FROM [Процесс] WHERE [DocID]  
IN [<Список DocID из первого запроса>] ORDER BY ...
```

Если в конфигурационном файле в секции описания параметров драйвера указывается опция `OptimWideResult`, равная 1, то для данного типа драйвера допускается оптимизация SQL-запросов, в случае, когда `Query.OpenHints[WideResult] = True`.

Пример:

```
[MSSQL2000_ADO]  
...  
OptimWideResult=1
```

- **ИзолированныйЗапрос / IsolatedQuery** - указывает серверу данных на то, что данный запрос по возможности нужно выполнять, не блокируя других пользователей. Рекомендуется использовать для запросов, которые медленно исполняются на большом объёме данных. Например, из-за неудачного или очень сложного фильтра или неудачной сортировки по полям, когда нет подходящего индекса. Данное свойство игнорируется для метода [Select](#), если в запросе присутствует подтаблица и распространяется на свойство [Count](#) и метод [CalcAggregates](#). В отладочной версии изолированные запросы помечаются сообщениями "Begin isolated query", "End isolated query", "Begin isolated Aggr", "End isolated Aggr".
- **ИгнорироватьФильтрЗаписи / IgnoreRecordFilter** - свойство позволяет в запросе проигнорировать постоянный фильтр, установленный на класс записей. Для этого значение свойства следует задать равным `True`. По умолчанию значение свойства равно `False`.

- **Прерываемый / Interrupted** - используется для того, чтобы иметь возможность прерывать выполнения запроса. По умолчанию свойство выключено `Query.OpenHints[Interrupted] = False`. Если оно включено `Query.OpenHints[Interrupted] = True`, то во время выполнения методов запроса **Select, CalcAggregates, Count** можно прервать выполнения прикладного кода, нажав клавиши **Ctrl+Break**. В отладочной версии прерываемые запросы помечаются префиксом "AS". Данное свойство поддерживается только для СУБД MS SQL, а для остальных игнорируется. Его рекомендуется устанавливать у запросов, для которых заранее известно, что они могут долго выполняться, чтобы у пользователя была возможность в любой момент прервать выполнение прикладного алгоритма.

Не рекомендуется злоупотреблять этим свойством, т.к. для выполнения прерываемого запроса MS SQL требуются дополнительные ресурсы.

В классе **Запрос** определен специальный перечислимый тип **РежимыИерархии / HierarchyModes**, содержащий константы для задания способов сортировки записей в результатах запроса. Значения этого типа используются при установке свойства [РежимИерархии / HierarchyMode](#).

В типе **РежимыИерархии** определены следующие значения:

- **БезИерархии / NoHierarchy** - результаты запроса представлены в виде "плоского" списка (групповые и негрупповые записи чередуются в нем в порядке следования в базе);
- **СмешаннаяИерархия / MixedHierarchy** - результаты запроса представлены в виде иерархического списка, состоящего из логических поднаборов записей, каждый из которых содержит записи одной группы; внутри каждой группы групповые и негрупповые записи чередуются в порядке следования в базе;
- **ГруппыВНачале / GroupsFirst** - результаты запроса представлены в виде иерархического списка, состоящего из логических поднаборов записей, каждый из которых содержит записи одной группы; внутри каждой группы групповые записи предшествуют негрупповым;
- **ТолькоГруппы / OnlyGroups** - результаты запроса представлены в виде иерархического списка, состоящего из логических поднаборов записей, каждый из которых содержит записи одной группы, причем сами эти записи также являются группами; негрупповые записи в запрос не попадают;
- **ТолькоЗаписи / OnlyRecords** - результаты запроса представлены в виде иерархического списка, состоящего из логических поднаборов записей, каждый из которых содержит записи одной группы, причем сами эти записи не являются группами; групповые записи в запрос не попадают.

Описание

Активен : Логическое;

Active : Logical;

Назначение

Возвращает TRUE, если запрос был выполнен (вызвана процедура [Выбор / Select](#)), и FALSE - в противном случае.

Пример

```
proc Запросить(Q : Query);
  try
    if NOT Q.Active then
      Q.Select;
    end;
  except
    -- неверные параметры запроса, или он не был создан
  end;
end;
```

Описание

ВключатьУдаленные : Логическое;
IncludeDeleted : Logical;

Назначение

Управляет видимостью удаленных документов в запросе. Если значение поля равно TRUE, то в результат запроса, помимо действующих документов, включаются документы, помеченные к удалению. В противном случае, список документов не содержит удаленных. По умолчанию значение поля равно FALSE.

Пример

```
Q = Query.Create;  
Q.IncludeDeleted = TRUE;
```

Описание

ВКонец : Логическое;
EOF : Logical;

Назначение

Возвращает значение TRUE после попытки переместить внутренний указатель запроса (доступный [Текущий / Current](#)) за последний документ из списка документов, удовлетворяющих условиям запроса, и FALSE в противном случае.

Если в запросе 0 элементов, то оба свойства и **BOF**, и **EOF** равны TRUE. Данный способ позволяет проверить, не пуст ли результат запроса, причем этот способ является более предпочтительным, чем проверка свойства **Count**, поскольку выполняется гораздо быстрее.

Поле доступно только на чтение.

Пример

```
var Q : Query;  
var T : Numeric;  
Q = Query.Create([Тест.Пример.Счет]);  
Q.Order = "Дата";  
Q.Filter = "ТипСчета=1"; -- 'Действующий счет'  
Q.Select; -- делаем выборку  
T = 0;  
-- прямой цикл по списку найденных документов  
while Q.EOF <> TRUE do  
    T = T + Q.Current.Total; -- обрабатываем документы  
    Q.Next;  
end;
```

Описание

ВНачале : Логическое;
BOF : Logical;

Назначение

Возвращает значение TRUE после попытки переместить внутренний указатель запроса (доступный по [Текущий / Current](#)) за первый документ из списка документов, удовлетворяющих условиям запроса, и FALSE в противном случае.

Если в запросе 0 элементов, то оба свойства и **BOF**, и **EOF** равны TRUE. Данный способ позволяет проверить, не пуст ли результат запроса, причем этот способ является более предпочтительным, чем проверка свойства **Count**, поскольку выполняется гораздо быстрее.

Поле доступно только на чтение.

Пример

```
var Q : Query;  
var T : Numeric;  
Q = Query.Create([Тест.Пример.Счет]);  
Q.Order = "Дата";  
Q.Filter = "ТипСчета=1"; -- 'Действующий счет'  
Q.Select; -- делаем выборку  
T = 0;  
-- обратный цикл по списку найденных документов  
Q.Last;  
while Q.>BOF <> TRUE do  
    T = T + Q.Current.Total; -- обрабатываем документы  
    Q.Previous;  
end;
```

Описание

ЗагружаемыеПоля : Строка;
LoadingFields : String;

Назначение

Позволяет определить и задать поля записей для предварительной загрузки. Имена используемых полей перечисляются в строке через точку с запятой (;).

Указание в данной строке лишь тех полей, с которыми необходимо в текущий момент работать, дает возможность оптимизировать быстродействие программы на ТБ.Скрипт, обращающейся к записям в информационной базе. Поля записи с указанными именами будут загружаться упреждающе, сокращая тем самым время доступа при фактическом обращении к ним.

Поле-свойство **ЗагружаемыеПоля** может использоваться в комбинации с полем-свойством [РежимЗагрузкиПолей](#). Установки этих двух полей дополняют друг друга (складываются логической операцией объединения, пояснения в примере).

Если документ получен не через запрос, а путем его открытия по DocID или при разыменовании ссылочного поля, то следует пользоваться одноименными свойствами класса [Document](#).

В свойстве **ЗагружаемыеПоля** можно указывать имена полей в структурах и полей, получаемых по разыменованию ссылочных полей (например, ДругойТипДокумента.Номер - для загрузки номеров из записей, ссылки на которые хранятся в поле ДругойТипДокумента). Для таких полей может применяться символ * (звезда), указывающий на необходимость загружать все подлежащие свойства (например, ДругойТипДокумента.* - для загрузки всех полей из записей, получаемых по ссылке ДругойТипДокумента).

Пусть, например, в MTL-файле описаны два следующих класса документов:

```
Document Full Title "Документ с полным набором полей";
  Field Номер : Integer Title "Номер документа";
  Field Докум : inherited Doc1;
  Struct Структ2 array Integer;
    Field Номер2 : Real;
    Field Докум2 : Doc1;
  end;
end;

Document Doc1;
  Field F1 :String;
end;
```

Если в записях класса Full необходимо проанализировать поля Докум.F1 и Структ2.Докум2.F1, то следует использовать примерно следующий код:

```
q = Kernel.Query.Create([DB1.Full]);
q.LoadingFieldsMode = 0;
q.LoadingFields = 'Докум.F1;Структ2.Докум2.F1';
q.Filter=ActualFilter;
q.Select;
```

При этом будет произведена оптимизация выборки по полям и ссылкам.

Если же в структурном поле Структ2 потребуется обращаться не только к Докум2, а ко всем полям, то обязательно следует указать Структ2.*:

```
q.LoadingFields = 'Докум.F1; Структ2.*; Структ2.Докум2.F1';
```

В противном случае при обращении к полю Структ2.Номер2 произойдет заметное снижение производительности.

Пример

См. также [Пример работы с объектом Запрос](#)

Описание

```
Записи Класс[] Запись;  
Records :Class[] Record;
```

Назначение

Позволяет задать или узнать классы записей (документов), по которым строится запрос.

Пример

```
-- подготовить выборку по счетам  
Q = Query.Create;  
Q.Order = "Дата";  
Q.Records = [Тест.Пример.Счет];
```

См. также [Пример работы с объектом Запрос](#)

Описание

Количество : Целое;
Count : Integer;

Назначение

Возвращает количество документов, попавших в результаты запроса. Поле доступно только на чтение.

Свойство работает как при открытом, так и при закрытом запросе, то есть фактически для получения количества документов используются только перечень классов документов запроса и его фильтр.

Пример

```
var Q : Query;  
Q = Query.Create([Тест.Пример.Счет]);  
Q.Order = "Дата";  
Q.Filter = "ТипСчета=1"; -- 'Действующий счет'  
Q.Select; -- делаем выборку  
Message("Всего "+Str(Q.Count)+ " действующих счетов");
```

Описание

ПараметрОткрытия [Параметр : [Запрос.ПараметрыОткрытия](#)] :Логическое;
OpenHint [Hint : [Query.OpenHints](#)] :Logical;

Аргументы

Параметр - необязательный оптимизирующий параметр запроса, задаваемый одной из предопределенных констант типа [ПараметрыОткрытия / OpenHints](#).

Назначение

Позволяет изменить или узнать оптимизирующие параметры запроса. Установленные параметры действуют только, начиная с момента построения запроса, то есть для включения или сброса той или иной оптимизации необходимо установить данное поле до вызова метода [Выбор](#). Если значение поля для какого-либо индекса (параметра) равно TRUE, то соответствующий режим оптимизации включён/включается.

Изменять параметры следует обдуманно, так как в результате их неоправданного использования произойдёт не ускорение, а замедление исполнения запроса. Данные параметры можно использовать, например, для получения нескольких первых записей из большого объёма данных (если более удачный алгоритм придумать невозможно).

Пример

```
-- нужно получить первые 10 записей за год:
var vQuery :Query;
vQuery = Query.Create([Документы.Накладные]);
vQuery.PacketSize = 10;
vQuery.Filter = '';
vQuery.OpenHint[Query.LargeResult] = True;
vQuery.OpenHint[Query.Packetting] = True;
vQuery.Select;
-- первые десять значений уже можно обрабатывать
--...
-- при обращении к 11-ой записи будет запрошен следующий пакет
```


Описание

РазмерПакета : Целое;
PacketSize : Integer;

Назначение

Позволяет изменить или узнать размер пакета, который используется при считывании документов с сервера. С помощью этого поля можно управлять динамикой буферизации операций чтения. Размер пакета определяет количество документов в буфере. По умолчанию размер пакета равен 60 и не может быть меньше 2. Как правило, оптимальный размер подбирается экспериментальным путем и зависит от решаемой задачи.

Пример

См. также [Пример работы с объектом Запрос](#)

Описание

РежимЗагрузкиПолей : Целое;
LoadingFieldsMode : Integer;

Назначение

Позволяет определить и изменить режим предварительной загрузки полей документов в запросе. Код режима может содержать произвольную комбинацию следующих значений:

- **mdAll** - загружать все поля
- **mdNone** - не загружать поля
- **mdSimple** - загружать простые поля
- **mdPeriodical** - загружать периодические поля
- **mdStruct** - загружать периодические структуры
- **mdArrays** - загружать массивы
- **mdStructArrays** - загружать массивы структур
- **mdBlob** - загружать большие поля ("большие двоичные объекты")

Эти константы определены в служебном проекте СИС, для их подключения необходимо использовать СИС как подпроект текущего проекта, а в начале модуля написать директиву:

Import СИС Classes Константы

По умолчанию (если в системном Реестре не указано иное) режим изначально равен mdSimple.

Применение различных режимов дает возможность оптимизировать быстродействие программы на ТБ.Скрипт, обращающейся к записям в информационной базе. Поля записи, отвечающие заданному режиму, будут загружаться упреждающе, сокращая тем самым время доступа при фактическом обращении к ним.

Поле-свойство **РежимЗагрузкиПолей** может использоваться в комбинации с полем-свойством [ЗагружаемыеПоля](#). Установки этих двух полей дополняют друг друга (складываются логической операцией объединения, пояснения в примере).

Если документ получен не через запрос, а путем его открытия по DocID или при разыменовании ссылочного поля, то следует пользоваться одноименными свойствами класса [Document](#).

Пример

См. также [Пример работы с объектом Запрос](#)

Описание

РежимИерархии : Запрос.[РежимыИерархии](#);
HierarchyMode : Query.HierarchyModes;

Назначение

Управляет способом сортировки записей в результатах запроса. Возможные варианты значения данного поля задаются перечислением **Запрос.РежимыИерархии**.

Пример

```
var Q : Query;  
Q = Query.Create([Первичка.Счет]);  
-- групповые записи должны быть перечислены в первую очередь  
Q.HierarchyMode = Query.GroupFirst;  
...
```

Описание

ТекущаяГруппа : Запись;
CurrentGroup : Record;

Назначение

Позволяет узнать и изменить запись, задающую текущую группу. Поле имеет смысл для классов записей, поддерживающих иерархию, и только в том случае, если [РежимИерархии](#) запроса отличен от варианта **БезИерархии**.

Пример

```
var Q : Query;  
Q = Query.Create([Первичка.Счет]);  
-- групповые записи должны быть обработаны в первую очередь  
Q.HierarchyMode = Query.GroupFirst;  
Q.Select;  
while not Q.EOF do  
  if Q.Current.IsGroup then  
    Q.CurrentGroup = Q.Current;  
    -- обработать группу  
    ProcessSubGroup(Q);  
  else  
    break; -- группы закончились  
  end;  
  Q.Next;  
end;
```

Описание

Текущий : Запись;
Current : Document;

Назначение

Возвращает указатель на текущий документ из списка документов, удовлетворяющих условиям запроса. Путем разыменования этой ссылки программист может получить доступ к любому полю документа.

Поле доступно не только на чтение, но и на запись. В последнем случае программа осуществляет переход на указанный документ (делает его текущим), если такой документ присутствует в результатах запроса. Если требуемого документа нет в результатах запроса, генерируется исключительная ситуация.

До тех пор пока запрос не построен с помощью процедуры **Выбор** или после того как для него вызвана процедура **Закрыть** поле **Текущий** равно nil.

Пример

```
proc ButtonThisPressed(B:Button);
var Q:Query;
  Q = Query.Create([Пример.ЮрЛицо]);
  Q.Select; -- строим запрос
  -- try...
  -- выводим текущий документ в бланк-редактор
  Document = Q.Current;
end;
```

См. также [Пример просмотра документов из результатов запроса](#).

Описание

Упорядочивание : Строка;
Order : String;

Назначение

Позволяет задать или узнать порядок сортировки документов в выборке. В строке указывается название одного или нескольких полей (разделенных точкой с запятой), по которым должны быть отсортированы документы в запросе. После имени каждого поля можно указать символ '+' (плюс) или '-' (минус). Плюс задает сортировку по возрастанию, минус - по убыванию.

Если поле **Упорядочивание** изменяется уже после того, как запрос был построен с помощью вызова метода [Выбор](#), запрос формируется заново - с учетом нового порядка сортировки, причем текущий внутренний указатель запроса (поле [Текущий](#)) сохраняет свою позицию.

Пример

```
Q = Query.Create;  
Q.Order = "пол;ФИО";  
-- сначала женщины, потом мужчины  
-- и те, и другие по ФИО в алфавитном порядке  
Q.Records = [Тест.Пример.Сотрудники];
```

См. также [Пример работы с объектом Запрос](#)

Описание

```
Выбор;  
Select;
```

Назначение

Осуществляет выборку документов, используя параметры объекта Запрос.

Пример

См. также [Пример работы с объектом Запрос](#)

Описание

Закреть
Close;

Назначение

Закрывает запрос, ранее сформированный с помощью вызова метода [Выбор / Select](#)), и FALSE - в противном случае. После закрытия запрос переводится в то же начальное состояние, в котором он находился до вызова метода **Выбор**. При этом результатов запроса более не существует, а поле **Текущий** получает значение **nil**.

Пользоваться данным методом рекомендуется в целях оптимизации быстродействия при необходимости изменить параметры запроса. Если запрос был построен, то изменение таких свойств как **Фильтр** или **Упорядочивание** приводит к повторному перестроению запроса уже с новыми параметрами, после чего документ, бывший текущим до перестроения, вновь делается текущим. Эта операция изменения указателя на текущий документ может занять столько же времени, сколько сама выборка документов. Поэтому рекомендуется перед изменением параметров запроса вызывать процедуру **Закреть**. В результате этого указатель на текущий документ получает значение **nil**, и повторного позиционирования после построения запроса не происходит.

При попытке закрыть запрос, который уже был закрыт или не был построен, генерируется исключительная ситуация.

Пример

```
var Q : Query;  
Q = Query.Create([Тест.Пример.Счет]);  
Q.Order = "Дата";  
Q.Filter = "ТипСчета=1";  
Q.Select; -- делаем выборку  
... -- анализ данных  
Q.Close; -- Q.Current становится равным nil  
Q.Filter = "ТипСчета=2";  
Q.Select; -- делаем выборку  
... -- анализ данных
```

Если строку **Q.Close** закомментировать, то пример по-прежнему будет работоспособным, но замедлится примерно в 2 раза.

Описание

Первый;
First;

Назначение

Осуществляет переход к первому документу из списка документов, попавших в результат запроса. Порядок следования документов в списке определяется параметрами запроса.

Пример

```
var Q : Query;  
Q = Query.Create([Тест.Пример.Счет]);  
Q.Order = "Дата";  
Q.Filter = "ТипСчета=1"; -- 'Действующий счет'  
Q.Select; -- делаем выборку  
  
-- цикл по списку найденных документов  
while Q.EOF <> TRUE do  
    -- обрабатываем документы 1-ый раз  
    -- ...  
    Q.Next;  
end;  
  
Q.First; -- возвращаемся в начало списка  
  
-- еще раз цикл по списку найденных документов  
while Q.EOF <> TRUE do  
    -- обрабатываем документы 2-ой раз, уже по-другому  
    -- ...  
    Q.Next;  
end;
```

См. также [Пример работы с объектом Запрос](#)

Описание

ПереместитьНа (Количество:Целое) ;
MoveBy (Number:Integer) ;

Аргументы

Количество - определяет, на сколько документов и в какую сторону следует переместить указатель вдоль списка документов, удовлетворяющих условиям запроса.

Назначение

Перемещает внутренний указатель запроса на указанное количество документов по списку документов, вошедших в результаты запроса. Аргумент может быть как положительным, так и отрицательным. В последнем случае переход осуществляется в сторону начала списка.

Если указанное количество превышает количество документов, находящихся в списке от текущей позиции до его начала или конца (в зависимости от знака аргумента), текущим становится соответственно первый или последний документ.

Пример

```
Класс "БланкЮЛ", Редактор Пример.Контрагент;  
-- кнопки "быстрой перемотки" списка документов  
ButtonFFNext : Button;  
ButtonRWPrev : Button;  
  
proc ButtonFFNextPressed(O:String);  
var Q:Query;  
Q = Query.Create([Пример.Контрагент]);  
Q.Select; -- строим запрос  
-- делаем текущим документ, открытый в бланке-редакторе  
Q.Current = Document;  
-- перескакиваем на 5 документов вперед  
Q.MoveBy(5);  
-- выводим текущий документ в бланк-редактор  
Document = Q.Current;  
end;  
  
proc ButtonRWPrevPressed(O:String);  
var Q:Query;  
Q = Query.Create([Пример.Контрагент]);  
Q.Select;  
Q.Current = Document;  
-- перескакиваем на 5 документов назад  
Q.MoveBy(-5);  
Document = Q.Current;  
end;  
  
end -- класса бланка-редактора
```

Описание

Последний;
Last;

Назначение

Осуществляет переход к последнему документу из списка документов, попавших в результат запроса. Порядок следования документов в списке определяется параметрами запроса.

Пример

```
var Q : Query;  
var T : Numeric;  
Q = Query.Create([Тест.Пример.Счет]);  
Q.Order = "Дата";  
Q.Filter = "ТипСчета=1"; -- 'Действующий счет'  
Q.Select; -- делаем выборку  
T = 0;  
Q.Last; -- переходим к последнему документу  
-- обратный цикл по списку найденных документов  
while Q.BOF <> TRUE do  
    T = T + Q.Current.Total; -- обрабатываем документы  
    Q.Previous;  
end;
```

Описание

Предыдущий ;
Previous ;

Назначение

Осуществляет переход к предыдущему документу из списка документов, попавших в результат запроса. Порядок следования документов в списке определяется параметрами запроса.

Пример

См. также [Пример просмотра документов из результатов запроса](#).

Описание

Следующий ;
Next ;

Назначение

Осуществляет переход к следующему документу из списка документов, попавших в результат запроса. Порядок следования документов в списке определяется параметрами запроса.

Пример

См. также [Пример просмотра документов из результатов запроса](#).

Описание

```
Фильтр : Строка;  
Filter: String;
```

Назначение

Позволяет установить или узнать фильтр, задающий условие отбора документов. В строке записывается выражение, содержащее названия полей документов, константы, функции, а также арифметические и логические операции над ними. Значение выражения должно иметь логический тип. В выборку документов попадают только те документы, у которых выражение, записанное в фильтре, равно значению ИСТИНА.

Общие правила составления фильтрующего выражения совпадают с правилами [установки пользовательских фильтров в картотеках](#). Кроме [стандартных функций](#), которые можно использовать в выражениях фильтров, допускается применять имена [служебных полей](#) и так называемые [кванторы](#).

Если поле Фильтр изменяется уже после того, как запрос был построен с помощью вызова метода [Выбор / Select](#), запрос формируется заново - с учетом нового фильтра, причем текущий внутренний указатель запроса (поле [Текущий / Current](#)) сохраняет свою позицию.

Следует иметь в виду, что выражения некоторых видов выполняются в фильтрах оптимизированным способом - посредством одного SQL-запроса, в то время как в остальных случаях выполняется непосредственный анализ содержимого записей. К оптимизированным выражениям для фильтров относятся:

- сравнение полей с константами;
- сравнение полей по разыменованным ссылкам с константами;
- функция Match, если ее аргумент не содержит функций, а только текст и подстановочные символы;
- функция Pos;
- выражения вида:
 <ИмяПодтаблицы>[<Выражение>].<ИмяПоляПодтаблицы> <ОперацияСравнения> <Выражение>
 и
 <ИмяМассива>[<Выражение>] <ОперацияСравнения> <Выражение>

Пример

```
Q = Query.Create;  
Q.Filter = "(ОКЛАД>2000) AND (ОТДЕЛ=" + Str(ОтделНомер) + ")";  
Q.Records = [Тест.Пример.Сотрудники];
```

См. также [Пример работы с объектом Запрос](#)

Описание

ВычислитьСоставные(Выражение :Строка) :Вариант;
CalcAggregates(Expression :String) :Variant;

Аргументы

Выражение - строка, содержащая одно или несколько перечисленных через запятую названий агрегатных макросов; если макрос принимает параметр, то параметр записывается после имени макроса в круглых скобках. Поддерживаемые агрегатные макросы:

Количество / Count - количество записей в результатах запроса;

Макс(<ИмяПоля>) / Max(<FieldName>) - максимальное значение в указанном поле среди записей из результатов запроса;

Мин(<ИмяПоля>) / Min(<FieldName>) - минимальное значение в указанном поле среди записей из результатов запроса;

Сум(<ИмяПоля>) / Sum(<FieldName>) - сумма значений в указанном поле по всем записям из результатов запроса.

Назначение

Функция предназначена для вычисления так называемых агрегатных значений, задаваемых с помощью макросов в параметре **Выражение**. Макросы выполняют обработку полей по набору записей, а не просто возвращают их значения.

Функция возвращает одно или несколько значений агрегатных макросов. Если в выражении был запрошен один макрос, то возвращаемое значение - есть результат работы макроса. Если в выражении было задано несколько макросов, то функция возвращает массив значений типа **Вариант**, причем размер массива соответствует количеству запрошенных макросов и элементы массива содержат результаты работы макросов в порядке их перечисления в выражении.

Функция работает как при открытом, так и при закрытом запросе, то есть фактически для нахождения значения любого макроса (минимального или максимального значения поля, суммы по полю или количества записей) используются только перечень классов документов запроса и его фильтр.

Если фильтру запроса не удовлетворяет ни одна запись, функция возвращает значение, у которого тип ([VarType](#)) равен **varNull** (для макросов Min, Max, Sum) или 0 (для Count).

Функция выполняется одним SQL-запросом, что позволяет оптимизировать скорость выполнения ТБ.Скрипт-кода.

Пример

```
var Q :Query;
var SMin, SMax :Numeric;
var SAve :Numeric;
var VarMas :Variant[];
-- создаем запрос по справочнику сотрудников
Q = Query.Create([Тест.Пример.Сотрудники]);
-- ограничиваемся первым отделом
Q.Filter = "Отдел=1";
-- строить запрос с помощью Select не обязательно
-- Q.Select;
if Q.Count > 0 then
    -- запрашиваем минимальное, максимальное значение окладов и
    -- их сумму по отделу
    VarMas = Q.CalcAggregates("Min(Оклад),Max(Оклад),Sum(Оклад)");
    -- минимальный оклад
    SMin = VarMas[1];
    -- максимальный оклад
    SMax = VarMas[2];
    -- вычисляем средний оклад
    SAve = VarMas[3];
    SAve = SAve / Q.Count;
```

```
message("Оклады: минимальный - " + Str(SMin)
        + "; максимальный - " + Str(SMax)
        + "; средний - " + Str(SAve));
end;
```


Описание

```
Найти(Искомое:Вариант):Логическое;  
Find(Wanted:Variant):Logical;
```

Аргументы

Искомое - значение произвольного типа (тип соответствует типу поля, по которому в первую очередь производилось упорядочивание).

Назначение

По указанному значению в некотором поле документа ищет его в результатах запроса. Если требуемый документ найден, он становится текущим (доступен посредством [Текущий / Current](#)). В случае успеха функция возвращает TRUE, иначе - FALSE.

Если в качестве искомого поля используется **DocID** документа, то вместо него можно передавать ссылку на сам документ.

Пример

```
-- найти экземпляр счета, который приписан к указанной сделке;  
-- идентификатор сделки хранится с поле "Сделка" документа "Счет"  
proc НайтиСчет(D:Тест.Пример.Сделка; var S:Тест.Пример.Счет);  
var Q : Query;  
  Q = Query.Create([Тест.Пример.Счет]);  
  Q.Order = "Сделка";  
  Q.Select;  
  -- ищем конкретный счет по сделке D  
  if Q.Find(D.DocID) then  
    S = Q.Current;  
    trace("Найден счет номер"+Str(Q.Current.OrderNumber));  
  else  
    S = пусто;  
  end;  
end;  
end;
```

Описание

НайтиБлижайший(Искомое:Вариант) :Логическое;
FindNearest(Wanted:Variant) :Logical;

Аргументы

Искомое - значение произвольного типа (тип соответствует типу поля, по которому в первую очередь производилось упорядочивание).

Назначение

Ищет указанное значение в поле, по которому упорядочивались документы в выборке. Если полного соответствия искомого и содержимого документов не обнаружено, текущим становится документ, имеющий наиболее близкое к искомому значение в обрабатываемом поле.

Если найдено полное соответствие, функция возвращает TRUE, иначе - FALSE.

Если в качестве искомого поля используется **DocID** документа, то вместо него можно передавать ссылку на сам документ.

Пример

```
var Q : Query;  
Q = Query.Create([Тест.Пример.Счет]);  
Q.Order = "Контрагент";  
Q.Select;  
if Q.FindNearest("Маркет") then  
    Message("Найдет счет "+Str(Q.Current.Контрагент));  
else  
    Message("Найдет похожий счет "+Str(Q.Current.Контрагент));  
end;
```

Описание

НайтиСледующий :Логическое;
FindNext :Logical;

Назначение

Повторяет поиск значения, заданного при предыдущем вызове функции [Find](#) (или [FindNearest](#)).

Если найдено еще одно искомое значение, функция возвращает TRUE (и делает документ с искомым значением текущим), иначе - FALSE.

Пример

```
var Q : Query;  
var N : Integer;  
Q = Query.Create([Тест.Пример.Счет]);  
Q.Filter = "ДатаСч>Сегодня-3"; -- действующий счет  
Q.Order = "Контрагент";  
Q.Select;  
N = 0;  
-- подсчитываем счета, выставленные фирме "Маркет"  
if Q.Find("Маркет") then  
    N = N + 1;  
    while Q.FindNext do  
        N = N + 1;  
    end;  
end;
```

Функция ПопадаетПодФильтр / MatchFilter

Описание

```
ПопадаетПодФильтр(ПроверяемаяЗапись :Запись) :Логическое;  
MatchFilter(RecordToCheck :Record) :Logical;
```

Аргументы

ПроверяемаяЗапись - ссылка на запись, которую необходимо проверить на соответствие фильтру запроса.

Назначение

Функция предназначена для проверки попадания указанной записи под фильтр текущего запроса.

Функция возвращает значение "Истина", если **ПроверяемаяЗапись** попадает под фильтр, и значение "Ложь" - в противном случае. Данный метод работает и при закрытом запросе.

Пример

```
- процедура проверяет, принадлежит ли запись о сотруднике  
-- первому отделу  
func ПервыйОтдел(Сотрудник : Тест.Пример.Сотрудники) :Logical;  
var Q :Query;  
    -- создаем запрос по справочнику сотрудников  
    Q = Query.Create([Тест.Пример.Сотрудники]);  
    -- ограничиваемся первым отделом  
    Q.Filter = "Отдел=1";  
    return Q.MatchFilter(Сотрудник);  
end;
```

Описание

```
Создать([Классы :Класс[] Запись]) : Запрос;  
Create([Classes :Class[] Record]) : Query;
```

Аргументы

Классы - необязательный массив классов записей (документов).

Назначение

Создает новый объект Запрос, позволяющий делать выборки записей заданных классов.

Если параметр опущен, классы документов должны быть указаны позднее с помощью поля [Записи](#). Непосредственно отбор документов осуществляется при вызове процедуры [Выбор](#).

Пример

```
func DoQuery(Filter: String): Numeric;  
var Q : Query;  
    -- подготовить выборку по счетам  
    Q = Query.Create([Тест.Пример.Счет]);  
    -- ... обработка  
end;
```

См. также [Пример работы с объектом Запрос](#)

Класс *ЗапросКонфликтовРепликации* (*ReplConflQuery*), производный от родительского класса *Объект*, предназначен для построения запросов по конфликтам репликации данных для текущей информационной базы.

В процессе репликации возможно выявление ситуаций, когда реплицируемые (принимаемые) данные каким-либо образом конфликтуют с содержимым информационной базы. Например, одна и та же запись (с равным ExtID) может быть отредактирована как в приемной информационной базе, так и в той базе, откуда прибыл пакет репликации.

Вначале система делает попытку разрешить конфликты на нижнем уровне. Оставшиеся после этого конфликтные записи (прибывшие в пакете репликации) сохраняются в специальном месте и затем могут быть запрошены с помощью класса *ЗапросКонфликтовРепликации*. После построения запроса разработчик проекта имеет возможность попытаться проанализировать конфликты на прикладном уровне. Например, можно выяснить состав одновременно изменившихся полей, с тем чтобы безопасным образом принять совместные правки разных филиалов, или же откорректировать записи, нарушающие условие уникальности по некоторому полю.

В классе *ЗапросКонфликтовРепликации* вводятся следующие новые свойства:

-  [Функция Создать / Create](#)
-  [Поле Записи / Records](#)
-  [Поле Фильтр / Filter](#)
-  [Поле ВключатьУдаленные / IncludeDeleted](#)
-  [Поле РазмерПакета / PacketSize](#)
-  [Поле ПараметрОткрытия / OpenHint](#)
-  [Поле ТекущийТипКонфликта / CurrentConflType](#)
-  [Поле РежимЗагрузкиПолей / LoadingFieldsMode](#)
-  [Поле ЗагружаемыеПоля / LoadingFields](#)
-  [Поле ТекущийИсточникКонфликта / CurrentConflSource](#)
-  [Процедура ПринятьТекущийКонфликт / AcceptCurrentConfl](#)
-  [Процедура ОтменитьТекущийКонфликт / AbortCurrentConfl](#)
-  [Поле ВНачале / BOF](#)
-  [Поле ВКонце / EOF](#)
-  [Поле Текущий / Current](#)
-  [Поле Количество / Count](#)
-  [Поле Активен / Active](#)
-  [Процедура Выбор / Select](#)
-  [Процедура Закрыть / Close](#)
-  [Процедура Первый / First](#)
-  [Процедура Последний / Last](#)
-  [Процедура Следующий / Next](#)
-  [Процедура Предыдущий / Previous](#)
-  [Процедура ПереместитьНа / MoveBy](#)
-  [Поле ЗагружатьТолькоЗаголовки / LoadHeadersOnly](#)

и перечислимый тип для описания возможных типов конфликтов:

-  [ТипыКонфликтовРепликации / ReplConflTypes](#).

В классе **ЗапросКонфликтовРепликации** определен специальный перечислимый тип **ТипыКонфликтовРепликации / ReplConfTypes**, описывающий возможные варианты конфликтов репликации. Значения данного типа используются свойством [ТекущийТипКонфликта](#).

В перечислимом типе определены следующие значения:

- **Изменен / Changed** - реплицируемый документ был изменен в базе, в которую производится репликация, со времени предыдущей репликации; потенциально существует две разных копии одного и того же документа;
- **ЗначенияУникальныхПолей / UniqueFieldsValues** - реплицируемый документ нарушает условие уникальности по какому-либо уникальному ключу, в базе уже существует документ с таким значением уникального поля.

Описание

Активен : Логическое;

Active : Logical;

Назначение

Возвращает TRUE, если запрос был выполнен (вызвана процедура [Выбор](#)), и FALSE - в противном случае. Если для запроса был также вызван метод [Заккрыть](#), то он снова становится неактивным.

Пример

```
proc Запросить(Q : ReplConflQuery);
  try
    if NOT Q.Active then
      Q.Select;
    end;
  except
    -- неверные параметры запроса, или он не был создан
  end;
end;
```


Описание

ВключатьУдаленные : Логическое;
IncludeDeleted : Logical;

Назначение

Управляет видимостью удаленных документов в запросе. Если значение поля равно TRUE, то в результат запроса, помимо действующих документов, включаются документы, помеченные удаленными. В противном случае, список документов не содержит удаленных. По умолчанию значение поля равно FALSE.

Пример

```
var Q : ReplConflQuery;  
Q = ReplConflQuery.Create;  
Q.IncludeDeleted = TRUE;
```

Описание

ВКонец : Логическое;
EOF : Logical;

Назначение

Возвращает значение TRUE после попытки переместить внутренний указатель запроса (доступный по [Текущий/Current](#)) за последний документ из списка документов, удовлетворяющих условиям запроса, и FALSE в противном случае.

Если в запросе 0 элементов, то оба свойства и **BOF**, и **EOF** равны TRUE. Данный способ позволяет проверить, не пуст ли результат запроса, причем этот способ является более предпочтительным, чем проверка свойства **Count**, поскольку выполняется гораздо быстрее.

Поле доступно только на чтение.

Пример

```
var Q : ReplConflQuery;  
var T : Numeric;  
Q = ReplConflQuery.Create([Тест.Пример.Счет]);  
Q.Select; -- делаем выборку  
T = 0;  
-- прямой цикл по списку найденных документов  
while Q.EOF <> TRUE do  
    -- обрабатываем документы по одному  
    Q.Next;  
end;
```

Описание

ВНачале : Логическое;
BOF : Logical;

Назначение

Возвращает значение TRUE после попытки переместить внутренний указатель запроса (доступный по [Текущий/Current](#)) за первый документ из списка документов, удовлетворяющих условиям запроса, и FALSE в противном случае.

Если в запросе 0 элементов, то оба свойства и **BOF**, и **EOF** равны TRUE. Данный способ позволяет проверить, не пуст ли результат запроса, причем этот способ является более предпочтительным, чем проверка свойства **Count**, поскольку выполняется гораздо быстрее.

Поле доступно только на чтение.

Пример

```
var Q : ReplConflQuery;  
var T : Numeric;  
Q = ReplConflQuery.Create([Тест.Пример.Счет]);  
Q.Select; -- делаем выборку  
T = 0;  
-- обратный цикл по списку найденных документов  
Q.Last;  
while Q.BOF <> TRUE do  
    -- обрабатываем документы по одному  
    Q.Previous;  
end;
```

Описание

`ЗагружаемыеПоля` : Строка;
`LoadingFields` : String;

Назначение

Позволяет определить и задать поля записей для предварительной загрузки. Имена используемых полей перечисляются в строке через точку с запятой (;).

Назначение этого поля аналогично полю [ЗагружаемыеПоля](#) в классе Запрос.

Пример

См. [Пример работы с объектом ЗапросКонфликтовРепликации](#).

Описание

ЗагружатьТолькоЗаголовки :Логическое;
LoadHeadersOnly :Logical;

Назначение

Поле позволяет определить, следует ли при построении запроса загружать все поля записей или только служебные (**ExtID**, **ModifyDate** и пр.). Если значение поля равно TRUE, загружаются только служебные поля, если FALSE - то все сразу. По умолчанию свойство равно FALSE, т.е. загружаются все поля. Переключение в режим загрузки только заголовков позволяет ускорить этап анализа содержимого запроса конфликтов репликаций (т.е. пока не задействованы поля прикладного уровня). Вне зависимости от того, какой режим включен, в любой момент можно прочитать из записи любое имеющееся в ней поле, в том числе и не служебное.

Значение поля можно менять в ходе работы с запросом.

Класс Объект : [ЗапросКонфликтовРепликации](#)

Поле Записи / Records

Описание

Записи :Класс[] Запись;
Records :Class[] Record;

Назначение

Позволяет задать или узнать список классов записей, по которым строится запрос - это те классы, по которым необходимо разрешить конфликты.

Пример

См. [Пример работы с объектом ЗапросКонфликтовРепликации.](#)

Описание

Количество : Целое;

Count : Integer;

Назначение

Возвращает количество документов, попавших в результаты запроса. Поле доступно только на чтение.

Пример

```
var Q : ReplConflQuery;  
Q = ReplConflQuery.Create;  
Q.Select; -- делаем выборку  
Message("Всего обнаружено "+Str(Q.Count)+ " конфликтов");
```

Описание

ПараметрОткрытия [Параметр : [Запрос.ПараметрыОткрытия](#)] :Логическое;
OpenHint [Hint :Query.OpenHints] :Logical;

Аргументы

Параметр - необязательный оптимизирующий параметр запроса, задаваемый одной из предопределенных констант типа [ПараметрыОткрытия / OpenHints](#).

Назначение

Позволяет изменить или узнать оптимизирующие параметры запроса. Установленные параметры действуют только начиная с момента построения запроса, то есть для включения или сброса той или иной оптимизации необходимо установить данное поле до вызова метода [Выбор](#). Если значение поля для какого-либо индекса (параметра) равно TRUE, то соответствующий режим оптимизации включён/включается.

Позволяет изменить или узнать оптимизирующие параметры запроса. Установленные параметры действуют только, начиная с момента построения запроса, то есть для включения или сброса той или иной оптимизации необходимо установить данное поле до вызова метода [Выбор/Select](#). Если значение поля для какого-либо индекса (параметра) равно TRUE, то соответствующий режим оптимизации включён/включается.

Изменять параметры следует обдуманно, так как в результате их неоправданного использования произойдёт не ускорение, а замедление исполнения запроса. Данные параметры можно использовать, например, для получения нескольких первых записей из большого объёма данных (если более удачный алгоритм придумать невозможно).

Пример

```
-- нужно проанализировать конфликты пакетами по 10 записей
var vQuery :ReplConflQuery;
vQuery = ReplConflQuery.Create([Документы.Накладные]);
vQuery.PacketSize = 10;
vQuery.OpenHint[Query.LargeResult] = True;
vQuery.OpenHint[Query.Packeting] = True;
vQuery.Select;
-- первые десять значений уже можно обрабатывать
--...
-- при обращении к 11-ой записи будет запрошен следующий пакет
```


Описание

РазмерПакета : Целое;
PacketSize : Integer;

Назначение

Позволяет изменить или узнать размер пакета, который используется при считывании документов из запроса с сервера. С помощью этого поля можно управлять динамикой буферизации операций чтения.

Размер пакета определяет количество документов в буфере. По умолчанию размер пакета равен 60 и не может быть меньше 2. Как правило, оптимальный размер подбирается экспериментальным путем и зависит от решаемой задачи.

Пример

```
-- нужно проанализировать конфликты пакетами по 10 записей
var vQuery :ReplConflQuery;
vQuery = ReplConflQuery.Create([Документы.Накладные]);
vQuery.PacketSize = 10;
vQuery.OpenHint[Query.LargeResult] = True;
vQuery.OpenHint[Query.Packetting] = True;
vQuery.Select;
-- первые десять значений уже можно обрабатывать
--...
-- при обращении к 11-ой записи будет запрошен следующий пакет
```

Описание

РежимЗагрузкиПолей : Целое;
LoadingFieldsMode : Integer;

Назначение

Позволяет определить и изменить режим предварительной загрузки полей документов в запросе. Код режима может содержать произвольную комбинацию следующих значений:

- **mdAll** - загружать все поля
- **mdNone** - не загружать поля
- **mdSimple** - загружать простые поля
- **mdPeriodical** - загружать периодические поля
- **mdStruct** - загружать периодические структуры
- **mdArrays** - загружать массивы
- **mdStructArrays** - загружать массивы структур
- **mdBlob** - загружать большие поля ("большие двоичные объекты")

Эти константы определены в служебном проекте СИС, для их подключения необходимо использовать СИС как подпроект текущего проекта, а в начале модуля написать директиву:

Import СИС Classes Константы

По умолчанию (если в системном Реестре не указано иное) режим изначально равен mdSimple.

Применение различных режимов дает возможность оптимизировать быстродействие программы на ТБ.Скрипт, обращающейся к записям в информационной базе. Поля записи, отвечающие заданному режиму, будут загружаться упреждающе, сокращая тем самым время доступа при фактическом обращении к ним.

Поле-свойство **РежимЗагрузкиПолей** может использоваться в комбинации с полем-свойством [ЗагружаемыеПоля](#). Установки этих двух полей дополняют друг друга (складываются логической операцией объединения, пояснения в примере).

Если документ получен не через запрос, а путем его открытия по DocID или при разыменовании ссылочного поля, то следует пользоваться одноименными свойствами класса [Record](#).

Пример

См. [Пример работы с объектом ЗапросКонфликтовРепликации](#).

Описание

Текущий : Запись;
Current : Record;

Назначение

Возвращает указатель на текущий документ из списка документов, удовлетворяющих условиям запроса.

Следует иметь в виду, что все записи, возвращаемые запросом **ЗапросКонфликтовРепликации** во время выполнения запроса, еще не сохранены в рабочей области информационной базы - они находятся в специальной области, доступной только посредством методов данного класса.

Путем разыменования этой ссылки программист может получить доступ к любому полю документа.

Поле доступно не только на чтение, но и на запись. В последнем случае программа осуществляет переход на указанный документ (делает его текущим), если такой документ присутствует в результатах запроса. Если требуемого документа нет в результатах запроса, генерируется исключительная ситуация.

До тех пор пока запрос не построен с помощью процедуры [Выбор/Select](#) или после того как для него вызвана процедура **Заккрыть** поле **Текущий** равно nil.

Пример

```
proc ButtonPressed(B:Button);
var Q :ReplConflQuery;
var Firm :Пример.ЮрЛицо;
  Q = ReplConflQuery.Create([Пример.ЮрЛицо]);
  Q.Select; -- строим запрос
  while not Q.EOF do -- для каждой записи
    Firm = Q.Current;
    -- анализируем поля текущего документа
  end;
end;
```

Описание

ТекущийИсточникКонфликта :Запись;
CurrentConflSource :Record;

Назначение

Если результаты запроса не пусты, поле содержит ссылку на запись из текущей информационной базы, которая конфликтует с импортируемой записью, доступной через свойство [Текущий/Current](#).

Поле доступно только на чтение.

Пример

См. [Пример работы с объектом ЗапросКонфликтовРепликации](#).

Класс Объект : [ЗапросКонфликтовРепликации](#)

Поле ТекущийТипКонфликта / CurrentConflType

Описание

ТекущийТипКонфликта : ЗапросКонфликтовРепликации.[ТипыКонфликтовРепликации](#);

CurrentConflType : ReplConflQuery.ReplConflTypes;

Назначение

Возвращает [тип](#) конфликта [текущей](#) записи. Поле доступно только на чтение.

Пример

См. [Пример работы с объектом ЗапросКонфликтовРепликации.](#)

Описание

Фильтр : Строка;
Filter : String;

Назначение

Позволяет установить или узнать фильтр, задающий условие отбора документов в запросе. В строке записывается выражение, содержащее названия полей документов, константы, функции, а также арифметические и логические операции над ними. Значение выражения должно иметь логический тип. В выборку документов попадают только те документы, у которых выражение, записанное в фильтре, дает значение ИСТИНА.

Общие правила составления фильтрующего выражения совпадают с правилами установки [пользовательских фильтров](#) в картотеках. Кроме [стандартных функций](#), которые можно использовать в выражениях фильтров, допускается применять имена [служебных полей](#) и так называемые [кванторы](#).

Если поле **Фильтр** изменяется уже после того, как запрос был построен с помощью вызова метода [Выбор/Select](#), запрос формируется заново - с учетом нового фильтра, причем текущий внутренний указатель запроса (поле [Текущий/Current](#)) сохраняет свою позицию.

Следует иметь в виду, что выражения некоторых видов выполняются в фильтрах оптимизированным способом - посредством одного SQL-запроса, в то время как в остальных случаях выполняется непосредственный анализ содержимого записей. К оптимизированным выражениям для фильтров относятся:

- сравнение полей с константами;
- сравнение полей по разыменованным ссылкам с константами;
- функция Match, если ее аргумент не содержит функций, а только текст и подстановочные символы;
- функция Pos;
- выражения вида:
 <ИмяПодтаблицы>[<Выражение>].<ИмяПоляПодтаблицы> <ОперацияСравнения> <Выражение>
 и
 <ИмяМассива>[<Выражение>] <ОперацияСравнения> <Выражение>

Пример

См. [Пример работы с объектом ЗапросКонфликтовРепликации](#).

Процедура Выбор / Select

Описание

Выбор;
Select;

Назначение

Осуществляет выборку документов, используя параметры объекта **ЗапросКонфликтовРепликации**.

В результаты запроса попадают все записи, для которых были предприняты неуспешные попытки репликации (импорта в данную информационную базу) из-за конфликтов с уже имеющимися в ИБ записями. Конфликтные записи остаются в особом разделе информационной базы вплоть до разрешения конфликта (исправление полей записи или её удаление) и никак не влияют на рабочее содержимое информационной базы, используемое прикладным проектом. Конфликтные записи могут быть в любой момент запрошены и проанализированы с помощью запроса конфликтов репликации.

Пример

См. [Пример работы с объектом ЗапросКонфликтовРепликации](#).

Описание

Закреть;
Close;

Назначение

Закрывает запрос, ранее сформированный с помощью вызова метода **Выбор**. После закрытия запрос переводится в то же начальное состояние, в котором он находился до вызова метода **Выбор**. При этом результатов запроса более не существует, а поле **Текущий** получает значение **nil**.

Пользоваться данным методом рекомендуется в целях оптимизации быстродействия при необходимости изменить параметры запроса. Если запрос был построен, то изменение, например, свойства **Фильтр** приводит к повторному перестроению запроса уже с новыми параметрами, после чего документ, бывший текущим до перестроения, вновь делается текущим. Эта операция изменения указателя на текущий документ может занять столько же времени, сколько сама выборка документов. Поэтому рекомендуется перед изменением параметров запроса вызывать процедуру **Закреть**. В результате этого указатель на текущий документ получает значение **nil**, и повторного позиционирования после построения запроса не происходит.

При попытке закрыть запрос, который уже был закрыт или не был построен, генерируется исключительная ситуация.

Пример

```
var Q : ReplConflQuery;  
Q = ReplConflQuery.Create([Тест.Пример.Счет]);  
Q.Select; -- делаем выборку  
... -- анализ данных  
Q.Close; -- Q.Current становится равным nil  
Q.Filter = "ТипСчета=2";  
Q.Select; -- делаем выборку  
... -- анализ данных
```

Если строку `Q.Close` закомментировать, то пример по-прежнему будет работоспособным, но замедлится примерно в 2 раза.

Описание

```
ОтменитьТекущийКонфликт;  
AbortCurrentConfl;
```

Назначение

Разрешает конфликт по текущей записи путем игнорирования изменений, полученных из пакета репликации.

Пример

См. [Пример работы с объектом ЗапросКонфликтовРепликации.](#)

Описание

Первый;
First;

Назначение

Осуществляет переход к первому документу из списка документов, попавших в результат запроса. Порядок следования документов в списке определяется параметрами запроса.

Пример

```
var Q : ReplConflQuery;  
Q = ReplConflQuery.Create([Тест.Пример.Счет]);  
Q.Filter = "ТипСчета=1"; -- 'Действующий счет'  
Q.Select; -- делаем выборку  
  
-- цикл по списку найденных документов  
while Q.EOF <> TRUE do  
    -- обрабатываем документы 1-ый раз  
    -- ...  
    Q.Next;  
end;  
  
Q.First; -- возвращаемся в начало списка  
  
-- еще раз цикл по списку найденных документов  
while Q.EOF <> TRUE do  
    -- обрабатываем документы 2-ой раз, уже по-другому  
    -- ...  
    Q.Next;  
end;
```

Описание

ПереместитьНа (Количество:Целое) ;
MoveBy (Number:Integer) ;

Аргументы

Количество - определяет на сколько документов и в какую сторону следует переместить указатель вдоль списка документов, удовлетворяющих условиям запроса.

Назначение

Перемещает внутренний указатель запроса на указанное количество документов по списку документов, вошедших в результаты запроса. Аргумент может быть как положительным, так и отрицательным. В последнем случае переход осуществляется в сторону начала списка.

Если указанное количество превышает количество документов, находящихся в списке от текущей позиции до его начала или конца (в зависимости от знака аргумента), текущим становится соответственно первый или последний документ.

Пример

```
var Q:ReplConflQuery;  
Q = ReplConflQuery.Create([Пример.ЮрЛицо]);  
Q.Select; -- строим запрос  
-- перескакиваем на 5 документов вперед  
Q.MoveBy(5);
```

Описание

Последний;
Last;

Назначение

Осуществляет переход к последнему документу из списка документов, попавших в результат запроса. Порядок следования документов в списке определяется параметрами запроса.

Пример

```
var Q : ReplConflQuery;  
Q = ReplConflQuery.Create([Тест.Пример.Счет]);  
Q.Select; -- делаем выборку  
Q.Last; -- переходим к последнему документу  
-- обратный цикл по списку найденных документов  
while Q.BOF <> TRUE do  
    -- обрабатываем документы  
    Q.Previous;  
end;
```

Описание

Предыдущий;
Previous;

Назначение

Осуществляет переход к предыдущему документу из списка документов, попавших в результат запроса. Порядок следования документов в списке определяется параметрами запроса.

Пример

```
var Q : ReplConflQuery;  
Q = ReplConflQuery.Create([Тест.Пример.Счет]);  
Q.Select; -- делаем выборку  
Q.Last; -- переходим к последнему документу  
-- обратный цикл по списку найденных документов  
while Q.BOF <> TRUE do  
    -- обрабатываем документы по одному  
    -- ...  
    -- переходим к предыдущему документу  
    Q.Previous;  
end;
```

Описание

ПринятьТекущийКонфликт;
AcceptCurrentConfl;

Назначение

Разрешает конфликт по текущей записи путем принятия изменений, полученных из пакета репликации.

Внимание! Процедура не производит каких-либо интеллектуальных действий по исправлению данных (например, исправление нарушения условия уникальности поля или коррекция ссылок на документ). Перед вызовом данной процедуры необходимо в явном виде произвести необходимые корректировки, такие как изменение значений уникальных полей, по которым есть конфликты, или корректировка ссылок. Если это не сделано, то попытка принять конфликт сгенерирует исключение.

Пример

См. [Пример работы с объектом ЗапросКонфликтовРепликации.](#)

Описание

Следующий;
Next;

Назначение

Осуществляет переход к следующему документу из списка документов, попавших в результат запроса. Порядок следования документов в списке определяется параметрами запроса.

Пример

```
var Q : ReplConflQuery;  
Q = ReplConflQuery.Create([Тест.Пример.Счет]);  
Q.Select; -- делаем выборку  
  
-- цикл по списку найденных документов  
while Q.EOF <> TRUE do  
    -- обрабатываем документы по одному  
    -- ...  
    -- переходим на следующий документ  
    Q.Next;  
end;
```

Функция Создать / Create

Описание

```
Создать([Классы :Класс[] Запись]) : ЗапросКонфликтовРепликации;  
Create([Classes :Class[] Record]) : ReplConflQuery;
```

Аргументы

Классы - необязательный массив классов записей (документов).

Назначение

Создает новый объект ЗапросКонфликтовРепликации, позволяющий делать выборки документов, которые вызвали конфликты при репликации данных и не были занесены в информационную базу.

Если параметр опущен, классы документов могут быть указаны позднее с помощью поля [Записи/Records](#).

Если набор классов не указан ни при создании запроса, ни позднее, то запрос будет строиться по всем классам проекта. Непосредственно отбор документов осуществляется при вызове процедуры [Выбор/Select](#).

Пример

См. [Пример работы с объектом ЗапросКонфликтовРепликации](#).

Класс *Изоляция* (*Isolation*), производный от родительского класса [Объект](#), используется для временного отключения клиента от уведомлений об изменении в таблицах заданных классов документов в информационной базе, что необходимо при анализе моментального непротиворечивого "снимка" данных в многопользовательской работе. Более подробно об изоляциях см. раздел [Транзакции и изоляции](#).

В классе *Изоляция* имеются следующие свойства:



[Создать / Create](#)



[Записи / Records](#)

Работать с изоляциями можно также с помощью методов [НачатьИзоляцию](#), [ЗавершитьИзоляцию](#) класса *Система*.

Описание

Записи :Класс [] [Запись](#);
Records :Class [] [Record](#);

Назначение

Возвращает массив ссылок на классы записей, для которых создана данная изоляция.

Поле доступно только на чтение.

Пример

```
proc ДобавитьЗапись (Iso :Isolation);
var x :Document;
  -- создаем документ того класса, для которого
  -- была открыта изоляция
  x = Iso.Records[1].Create;
  -- инициализируем поля
  x.Дата = Сегодня;
  -- записываем документ
  x.Post;
  -- фактически новый документ появится в картотеках
  -- на клиентах только после того как объект изоляции
  -- будет уничтожен
end;
```

Описание

```
Создать([КлассыЗаписей :Класс[] Запись]) :Изоляция;  
Create([RecordClasses :Class[] Record]) :Isolation;
```

Аргументы

КлассыЗаписей - массив ссылок на классы записей, для которых будет выполняться изоляция.

Назначение

Создает объект класса **Изоляция** и открывает изоляцию для указанных классов записей.

Если массив опущен, изоляция открывается на все классы документов всех проектов, для которых была создана текущая информационная база.

Возвращает ссылку на созданный объект.

Изоляция прекращает свое действие в момент уничтожения объекта, например, при выходе из его области видимости.










Пример

```
proc Analyze;  
Var Iso :Isolation;  
  -- создаем изоляцию по счетам  
  Iso = Isolation.Create([Счет]);  
  try  
    -- анализируем счета  
    -- ...  
  finally  
    -- обработка ошибок  
  end;  
  -- при выходе из процедуры объект Iso уничтожается,  
  -- а изоляция закрывается  
end;
```




Импортёр / Importer

Класс *Импортёр* (*Importer*) является производным от родительского класса [Объект](#) и наследует от него все свойства и методы. Класс позволяет импортировать из файла наборы записей (документов) определенных типов в информационную базу.

В классе *Импортёр* доступны следующие свойства:

-  [Функция Создать / Create](#)
-  [Поле АвтоСоотв / AutoCorr](#)
-  [Процедура ИмпортИзФайла / ImportFromFile](#)
-  [Процедура ДобавитьСоответствие / AddCorrespondence](#)
-  [Процедура СохранитьСоответствие / SaveCorrespondence](#)
-  [Процедура ЗагрузитьСоответствие / LoadCorrespondence](#)
-  [Поле Режим / ConflictMode](#)
-  [Поле Комментарии / Comments](#)
-  [Событие ПриКонфликте / OnConflict](#)

и перечислимые типы

-  [Тип ТипыРежимов / ConflictModeType](#)
-  [Тип СпособыРазрешенияКонфликта / ConflictResolveTypes](#)
-  [Тип Опции / Options](#)

Перечислимый тип *ТипыРежимов / ConflictModeType* определяет константы, обозначающие способы реакции системы на конфликты при импорте данных.

Константы, заданные в этом типе, используются в методе [ИмпортЗаписей](#) класса *Система* и свойстве [Режим / ConflictMode](#) класса *Импортёр*.

Определены следующие константы для задания режима разрешения конфликтов:

- **Прерывать / Abort** - операция импорта прерывается;
- **Пропускать / Skip** - импортируемая запись, которая вызвала конфликт, пропускается;
- **Заменять / Replace** - импортируемая запись, которая вызвала конфликт с имеющейся записью, заменяет эту имеющуюся запись;
- **Спрашивать / Ask** - системы выдает запрос пользователю, что делать.

По умолчанию предполагается спрашивать пользователя.

Описание

АвтоСоотв :Логическое;
AutoCorr :Logical;

Назначение

Если значение поля равно TRUE, то связь между полями записей в информационной базе и полями, описанными в файле, происходит автоматически (на основе их названий и типов). В противном случае соответствие должно быть задано с помощью процедуры [ДобавитьСоответствие](#) или [ЗагрузитьСоответствие](#).

После создания объекта значение поля по умолчанию равно TRUE.

Пример

```
var ImportObj :Importer;  
-- отключаем автоматическое соответствие  
  ImportObj.AutoCorr = False;
```

См. также [Пример использования объекта Импортёр](#).

Описание

Комментарии :Строка;
Comments :String;

Назначение

Поле предназначено для чтения/записи комментариев.

Пример

```
var ImportObj :Importer;  
-- ...  
    ImportObj.Comments;
```

Описание

Режим : [Импортёр.ТипыРежимов](#);

ConflictMode : [Importer.ConflictModeType](#);

Назначение

Задаёт способ разрешения возможных конфликтов при выполнении операции импорта. Доступные варианты описываются с помощью специального типа [ConflictModeType](#). По умолчанию значение поля равно *ТипыРежимов.Спрашивать*, что означает, что система будет запрашивать пользователя о дальнейших действиях при возникновении конфликтов.

Пример

```
proc ImportAllData (FileName:String);
var ImportObj :Importer;
  -- инициализируем объект
  ImportObj = Importer.Create;
  -- предписываем системе прерывать конфликты операции
  ImportObj.ConflictMode = Importer.Abort;
  -- выполняем операцию импортирования
  ImportObj.ImportFromFile(FileName, 'tbd');
end;
```


Описание

ДобавитьСоответствие(Откуда: Строка; Куда:Строка; Поля:Строка[]);
AddCorrespondence(Where:String; WhereFrom:String; Fields:String[]);

Аргументы

Откуда - имя записи (типа документов) в импортируемом файле;

Куда - имя записи (типа документов) в информационной базе;

Поля - массив строк, задающих соответствие полей в указанных записях; строки имеют вид: "<Имя поля Откуда> = <Имя поля Куда>".

Назначение

В явном виде устанавливает соответствие между полями записей, содержащихся в импортируемом файле, и полями записей информационной базы.

Пример

```
var ImportObj :Importer;  
-- ...  
ImportObj.AddCorrespondence('Test.DB1.Full', 'Full',  
    ['Номер1 = Номер', 'Дата = Дата']);
```

См. также [Пример использования объекта Импортер](#).

Описание

```
ЗагрузитьСоответствие(ИмяФайла :Строка);  
LoadCorrespondence(FileName :String;)
```

Аргументы

ИмяФайла - имя файла для загрузки списка соответствий полей.

Назначение

Загружает из указанного файла список соответствий полей.

Пример

```
ImportObj :Importer; -- глобальный объект-импортер  
  
-- ImportObj уже проинициализирован в момент вызова данной процедуры  
proc LoadImport1BtnClick (Sender :Object);  
var sName :String;  
-- запрашиваем имя файла для соответствий  
if ChooseFile(fName, "Выбор файла",  
  "Список соответствий полей(*.txt)|*.txt") <> cmOk then  
  return;  
end;  
  
ImportObj.LoadCorrespondence(sName);  
end;
```

Описание

```
ИмпортИзФайла(Файл:Строка; Формат:Строка);  
ImportFromFile(File:String; Format:String);
```

Аргументы

Файл - имя файла, из которого будет проводиться импорт;

Формат - формат считываемого файла в виде строки с расширением, например, "tbd", "tbc", "xml".

Назначение

Импортирует в базу все документы из указанного файла.

Пример

```
var ImportObj :Importer;  
-- ...  
-- выполняем операцию импортирования  
ImportObj.ImportFromFile("картотека1.tbc","tbc");
```

См. также [Пример использования объекта Импортёр](#).

Описание

```
СохранитьСоответствие(ИмяФайла :Строка);  
SaveCorrespondence(FileName :String);
```

Аргументы

ИмяФайла - имя файла для сохранения списка соответствий полей.

Назначение

Сохраняет в указанный файл список соответствий полей.

Пример

```
ImportObj :Importer; -- глобальный объект-импортер  
  
-- ImportObj уже проинициализирован в момент вызова данной процедуры  
proc SaveImport1BtnClick (Sender :Object);  
var sName :String;  
-- запрашиваем имя файла для соответствий  
if ChooseFile(fName, "Выбор файла",  
  "Список соответствий полей(*.txt)|*.txt") <> cmOk then  
  return;  
end;  
  
ImportObj.SaveCorrespondence(sName);  
end;
```

Описание

ПриКонflikте : Строка;
OnConflict : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при импорте при наличии конфликтующего документа (см. далее).

Обработчик

```
proc OnConflict (Importer :Importer; ExistingRecord :Record; ImportingRecord :Record; var  
ConflictResolveType : Importer.ConflictResolveTypes);
```

Параметры, передаваемые в обработчик:

Importer - объект класса **Importer**, вызвавший этот обработчик;

ExistingRecord - существующая запись;

ImportingRecord - импортируемая запись (все поля доступны только на чтение);

ConflictResolveType - способ разрешения конкретного конфликта, по умолчанию возвращаемый параметр равен rtSkip.

Обработчик этого события вызывается, когда в поле [Режим](#) установлено значение **Спрашивать** и при импорте встречается конфликтующий документ (событие срабатывает для каждого конфликта отдельно).

Перечислимый тип **Опции / Options**, определенный в классе **Импортер** предназначен для задания различных опций при выполнении импорта данных и содержит следующие константы:

- **ПоказатьМастер / ShowWizard** - открывается мастер импорта;
- **АвтоматическийИмпорт / AutoImport** - запускается импорт.

Константы, определенные в данном типе, используются в методе [ИмпортЗаписей](#) класса **Система**. Причем указанные опции задаются в методе как массив опций, что позволяет одновременно выполнить несколько различных действий. Например, если параметр **Опции** содержит два элемента [Importer.ShowWizard, Importer.AutoImport], то открывается мастер импорта и сразу запускается импорт. А после успешного завершения импорта, мастер автоматически закрывается.

Перечислимый тип *СпособыРазрешенияКонфликта*, определенный в классе *Импортер*, позволяет задать способы разрешения конфликтов, возникающих при импорте документов.

Перечислимый тип содержит следующий набор констант:

- **rtSkip / срПропустить** - проигнорировать (не импортировать) текущую импортируемую запись;
- **rtSkipAll / срПропуститьВсе** - проигнорировать (не импортировать) все записи текущего класса (т.е. того же класса, что и импортируемая запись);
- **rtOverwrite / срЗаменить** - импортировать текущую импортируемую запись поверх существующей записи;
- **rtOverwriteAll / срЗаменитьВсе** - импортировать все записи того же класса, что и импортируемая запись поверх существующих записей;
- **rtAbort / срПрервать** - прервать процесс импорта.

Константы, определенные в данном типе, используются в обработке события [ПриКонflikте](#) класса *Импортер*.

Описание

```
Создать ({Владелец :Объект}) :Импортер;  
Create ({Owner :Object}) :Importer;
```

Аргументы

Владелец - необязательный параметр, который *следует указывать, когда задействуется обработчик [ПриКонflikте](#)* (т.к. **Владелец** должен быть владельцем метода, который указывается в обработчике [ПриКонflikте](#)).

Следует учитывать, что на самом деле **Владелец** должен быть объектом класса **UserObject**, т.е. не встроенного класса, а реализованного на ТБ.Скрипт.

Назначение





Создает новый объект класса **Импортер**, с помощью которого затем осуществляется импорт выборки документов из файла в базу.

Пример








```
proc ImportBtnClick (Sender :Object);  
var fName :String;  
var ImportObj :Importer;  
-- запрашиваем имя файла для импорта  
if ChooseFile(fName, "Выбор файла", "Данные в формате tbc|*.tbc")  
<> cmOk then  
    return;  
end;  
-- инициализируем объект  
ImportObj = Importer.Create;  
-- выполняем операцию импортирования  
ImportObj.ImportFromFile(fName, 'tbc');  
end;
```


Класс *Ограничение / Constraint* является производным от родительского класса [Объект](#) и наследует от него все свойства и методы. Класс *Ограничение* позволяет добавлять пользовательские ограничения для конкретной сессии на время ее работы. После закрытия сессии эти ограничения нигде не сохраняются. Кроме этого, [пользовательские ограничения](#) можно задавать также при описании структуры данных в файлах *.mtl.

Для задания ограничений программным способом также добавлены свойства и методы к классу [Запись](#):

-  [Поле КоличествоПользовательскихОграничений / UserConstraintsCount](#)
-  [Поле ПользовательскоеОграничение / UserConstraint](#)
-  [Функция ДобавитьПользовательскоеОграничение / AddUserConstraint](#)
-  [Процедура УдалитьПользовательскоеОграничение / DeleteUserConstraint](#)

Класс *Ограничение* не имеет методов, у него определены только свойства:

-  [Поле Имя / Name](#)
-  [Поле Ограничение / Constraint](#)
-  [Поле Сообщение / Message](#)
-  [Поле ПриЗаписи / OnPost](#)
-  [Поле ПриИзменении / OnEdit](#)
-  [Поле ПриУдалении / OnDelete](#)
-  [Поле ПриВосстановлении / OnUndelete](#)

Описание

Имя :Строка;
Name :String;

Назначение

Свойство позволяет установить или узнать (считать) имя ограничения.

Поле доступно на чтение и запись.

Пример

```
inobject
var SelectedClassName :String;
var ConstraintsCount  :Integer;
var SelectedClass     :class Record;
...
proc RefillConstraints;
var i :Integer;
var vConstraint :Constraint;
...
for i = 1..ConstraintsCount do
  vConstraint = SelectedClass.UserConstraint[i];
  with vConstraint do
    ConstrBuffer[1, i] = i;
    ConstrBuffer[2, i] = vConstraint.Name;
    ConstrBuffer[3, i] = vConstraint.Constraint;
    ConstrBuffer[4, i] = vConstraint.Message;
    ConstrBuffer[5, i] = vConstraint.OnPost;
    ConstrBuffer[6, i] = vConstraint.OnEdit;
    ConstrBuffer[7, i] = vConstraint.OnDelete;
    ConstrBuffer[8, i] = vConstraint.OnUndelete;
    ...
  end;
end;
...
end;
```

Описание

Ограничение :Строка;
Constraint :String;

Назначение

Свойство позволяет установить или узнать (считать) собственно текст ограничения.

Поле доступно на чтение и запись.

Пример

```
inobject
var SelectedClassName :String;
var ConstraintsCount   :Integer;
var SelectedClass      :class Record;
...
proc RefillConstraints;
var i :Integer;
var vConstraint :Constraint;
...
for i = 1..ConstraintsCount do
  vConstraint = SelectedClass.UserConstraint[i];
  with vConstraint do
    ConstrBuffer[1, i] = i;
    ConstrBuffer[2, i] = vConstraint.Name;
    ConstrBuffer[3, i] = vConstraint.Constraint;
    ConstrBuffer[4, i] = vConstraint.Message;
    ConstrBuffer[5, i] = vConstraint.OnPost;
    ConstrBuffer[6, i] = vConstraint.OnEdit;
    ConstrBuffer[7, i] = vConstraint.OnDelete;
    ConstrBuffer[8, i] = vConstraint.OnUndelete;
    ...
  end;
end;
...
end;
```

Описание

ПриВосстановлении :Логическое;
OnUndelete :Logical;

Назначение

Свойство позволяет включить ограничение, заданное своим [именем](#) и [условием](#), при наступлении события ПриВосстановлении, а также узнать сработает, ли это ограничение при восстановлении записи (OnUndelete=True) или нет (OnUndelete=False).

Свойство доступно на чтение и запись.

Пример

```
inobject
var SelectedClassName :String;
var ConstraintsCount  :Integer;
var SelectedClass     :class Record;
...
proc RefillConstraints;
var i :Integer;
var vConstraint :Constraint;
...
for i = 1..ConstraintsCount do
    vConstraint = SelectedClass.UserConstraint[i];
    with vConstraint do
        ConstrBuffer[1, i] = i;
        ConstrBuffer[2, i] = vConstraint.Name;
        ConstrBuffer[3, i] = vConstraint.Constraint;
        ConstrBuffer[4, i] = vConstraint.Message;
        ConstrBuffer[5, i] = vConstraint.OnPost;
        ConstrBuffer[6, i] = vConstraint.OnEdit;
        ConstrBuffer[7, i] = vConstraint.OnDelete;
        ConstrBuffer[8, i] = vConstraint.OnUndelete;
        ...
    end;
end;
...
end;
```

Описание

ПриЗаписи :Логическое;
OnPost :Logical;

Назначение

Свойство позволяет включить ограничение, заданное своим [именем](#) и [условием](#), при наступлении события ПриЗаписи, а также узнать сработает, ли это ограничение при записи документа (OnPost=True) или нет (OnPost=False).

Свойство доступно на чтение и запись.

Пример

```
inobject
var SelectedClassName :String;
var ConstraintsCount  :Integer;
var SelectedClass     :class Record;
...
proc RefillConstraints;
var i :Integer;
var vConstraint :Constraint;
...
for i = 1..ConstraintsCount do
    vConstraint = SelectedClass.UserConstraint[i];
    with vConstraint do
        ConstrBuffer[1, i] = i;
        ConstrBuffer[2, i] = vConstraint.Name;
        ConstrBuffer[3, i] = vConstraint.Constraint;
        ConstrBuffer[4, i] = vConstraint.Message;
        ConstrBuffer[5, i] = vConstraint.OnPost;
        ConstrBuffer[6, i] = vConstraint.OnEdit;
        ConstrBuffer[7, i] = vConstraint.OnDelete;
        ConstrBuffer[8, i] = vConstraint.OnUndelete;
        ...
    end;
end;
...
end;
```

Описание

ПриИзменении :Логическое;
OnEdit :Logical;

Назначение

Свойство позволяет включить ограничение, заданное своим [именем](#) и [условием](#), при наступлении события ПриИзменении, а также узнать сработает, ли это ограничение при изменении записи (OnEdit=True) или нет (OnEdit=False).

Свойство доступно на чтение и запись.

Пример

```
inobject
var SelectedClassName :String;
var ConstraintsCount  :Integer;
var SelectedClass     :class Record;
...
proc RefillConstraints;
var i :Integer;
var vConstraint :Constraint;
...
for i = 1..ConstraintsCount do
    vConstraint = SelectedClass.UserConstraint[i];
    with vConstraint do
        ConstrBuffer[1, i] = i;
        ConstrBuffer[2, i] = vConstraint.Name;
        ConstrBuffer[3, i] = vConstraint.Constraint;
        ConstrBuffer[4, i] = vConstraint.Message;
        ConstrBuffer[5, i] = vConstraint.OnPost;
        ConstrBuffer[6, i] = vConstraint.OnEdit;
        ConstrBuffer[7, i] = vConstraint.OnDelete;
        ConstrBuffer[8, i] = vConstraint.OnUndelete;
        ...
    end;
end;
...
end;
```

Описание

ПриУдалении :Логическое;
OnDelete :Logical;

Назначение

Свойство позволяет включить ограничение, заданное своим [именем](#) и [условием](#), при наступлении события ПриУдалении, а также узнать сработает, ли это ограничение при удалении записи (OnDelete=True) или нет (OnDelete=False).

Свойство доступно на чтение и запись.

Пример

```
inobject
var SelectedClassName :String;
var ConstraintsCount  :Integer;
var SelectedClass     :class Record;
...
proc RefillConstraints;
var i :Integer;
var vConstraint :Constraint;
...
for i = 1..ConstraintsCount do
    vConstraint = SelectedClass.UserConstraint[i];
    with vConstraint do
        ConstrBuffer[1, i] = i;
        ConstrBuffer[2, i] = vConstraint.Name;
        ConstrBuffer[3, i] = vConstraint.Constraint;
        ConstrBuffer[4, i] = vConstraint.Message;
        ConstrBuffer[5, i] = vConstraint.OnPost;
        ConstrBuffer[6, i] = vConstraint.OnEdit;
        ConstrBuffer[7, i] = vConstraint.OnDelete;
        ConstrBuffer[8, i] = vConstraint.OnUndelete;
        ...
    end;
end;
...
end;
```

Описание

Сообщение :Строка;
Message :String;

Назначение

Свойство позволяет установить или узнать (считать) текст сообщения, выдаваемый пользователю при нарушении условия ограничения.

Поле доступно на чтение и запись.

Пример

```
inobject
var SelectedClassName :String;
var ConstraintsCount   :Integer;
var SelectedClass      :class Record;
...
proc RefillConstraints;
var i :Integer;
var vConstraint :Constraint;
...
for i = 1..ConstraintsCount do
  vConstraint = SelectedClass.UserConstraint[i];
  with vConstraint do
    ConstrBuffer[1, i] = i;
    ConstrBuffer[2, i] = vConstraint.Name;
    ConstrBuffer[3, i] = vConstraint.Constraint;
    ConstrBuffer[4, i] = vConstraint.Message;
    ConstrBuffer[5, i] = vConstraint.OnPost;
    ConstrBuffer[6, i] = vConstraint.OnEdit;
    ConstrBuffer[7, i] = vConstraint.OnDelete;
    ConstrBuffer[8, i] = vConstraint.OnUndelete;
    ...
  end;
end;
...
end;
```


Класс *Подтаблица* (*Subtable*), производный от родительского класса [Объект](#), используется для работы с многозначными полями и массивами структурных полей, входящими в состав документов.

В классе *Подтаблица* вводятся следующие свойства класса:

 [Поле ИнформацияОКлассе / ClassInfo](#)

и объектов:

 [Поле Количество / Count](#)

 [Функция Добавить / Add](#)

 [Функция ДобавитьДоп / AddEx](#)

 [Функция Вставить / Insert](#)

 [Процедура Удалить / Delete](#)

 [Процедура Очистить / Clear](#)


 [Функция СуммаПоля / SumOfField](#)


 [Функция СоздатьПредставлениеПодтаблицы / CreateSubtableView](#)

 [Поле Пункты / Items](#)

 [Поле ПунктыПоНомеру / ItemsByNumber](#)

 [Поле ИндексПоНомеру / IndexByNumber](#)

 [Поле НомерПоИндексу / NumberByIndex](#)

 [Поле Порядок / Order](#)

 [Поле Фильтр / Filter](#)

Внимание! Создать объект класса *Подтаблица* напрямую нельзя. Подтаблицы создаются только внутри системы и используются как свойства других объектов, причем такие свойства имеют атрибут "только чтение". Вместе с тем, программа, написанная на языке ТБ.Скрипт, может получить доступ к объекту *Подтаблица*, являющемуся свойством другого объекта, и изменять свойства самой подтаблицы.

Описание

ИндексПоНомеру(Номер:Целое): Вариант;
IndexByNumber(Number:Integer): Variant;

Аргументы

Номер - определяет, для какого элемент подтаблицы требуется узнать значение, использованное в качестве индекса.

Назначение

Позволяет прочитать индекс конкретного элемента подтаблицы по его порядковому номеру. Тип индекса зависит от типа подтаблицы. Если подтаблица описана в [MTL](#) как массив (Array) - индекс должен быть ординарного типа (Logical или Integer). Если подтаблица описана в MTL как периодическое поле (Period) - индекс может быть любого типа, на котором определены отношения больше/меньше (String, Integer, Numeric, Logical, Date). Индексы всех элементов подтаблицы имеют один и тот же тип.

Поле доступно только на чтение.

Пример

```
func Редактирование(Doc :Document; FieldName :String) :Integer;
  var КоличПоз, j:Integer;
  var Field:Subtable;
  var dat:Date[]; -- массив дат
  var val:Numeric[]; -- массив значение за даты

  Field = Subtable(Doc.GetField(FieldName));

  КоличПоз=Field.Count;
  for j=1..КоличПоз do
    dat[j]=Field.IndexByNumber[j];    -- дата
    val[j]=Field.ItemsByNumber[j];
    -- значение за эту дату
  end;

  -- вызываем прикладную функцию
  return Проверить(dat,val,КоличПоз);
end;
```

Класс Объект : [Подтаблица](#)

Поле ИнформацияОКлассе / ClassInfo

Описание

ИнформацияОКлассе : [ИнфПодтаблицы](#);
ClassInfo : [SubtableClassInfo](#);

Назначение

Свойство класса ClassInfo возвращает указатель на объект RTTI-класса [ИнфПодтаблицы](#), содержащий описание текущего класса.

Описание

Количество :Целое;
Count :Integer;

Назначение

Содержит количество значений (экземпляров) структур или полей, находящихся в подтаблице. Поле доступно только на чтение. Для добавления нового значения или удаления имеющегося необходимо использовать методы Добавить, Вставить, Удалить, Очистить.

Пример

```
proc p1(N: Пример.Накладная);  
var i: integer;  
  if N.Позиции.Количество = 0 then  
    i = N.Позиции.Добавить;  
    N.Позиции[i].Товар = "Услуги по установке";  
    N.Позиции[i].Цена = 15;  
    N.Позиции[i].Валюта = "USDпокурсу";  
  end;  
end;
```

Описание

```
НомерПоИндексу[Индекс :Вариант] :Целое;  
NumberByIndex [Index :Variant] :Integer;
```

Аргументы

Индекс - физический индекс строки подтаблицы.

Назначение

Свойство возвращает логический индекс строки подтаблицы по заданному физическому индексу в представлении с учетом фильтра и сортировки на подтаблицу.

Если данная строка не удовлетворяет фильтру, то свойство возвращает 0.

Поле доступно только на чтение.

Пример

```
var vStable :Subtable;  
var i :Integer;  
  
-- получаем в представлении курсы валют больше 22 в обратном порядке  
vStable = Subtable(Document.GetField("Курс")).CreateSubtableView('Курс>22', 'Курс-');  
ClearTrace;  
for i = 1..Курс.Count do  
    Trace(Str(i) + ' = ' + Str(vStable.NumberByIndex[Курс.IndexByNumber[i]]));  
end;  
-- В окне сообщений будет:  
-- 1 = 0 -- строки нет в представлении  
-- 2 = 0 -- ...  
-- 3 = 2 -- т.к. в обратном порядке  
-- 4 = 1 -- ...
```

Описание

Порядок :Строка;
Order :String;

Назначение

Свойство предназначено для установки сортировки подтаблицы.

Данное свойство доступно только для подтаблицы, полученной в результате вызова метода [Subtable.CreateSubtableView](#).

Предупреждение. При попытке изменить или получить свойство **Порядок/Order** у обычной подтаблицы выдается ошибка.

Описание

Пункты(Индекс:Вариант): Вариант;
Items(Index:Variant): Variant;

Аргументы

Индекс - определяет, какой элемент подтаблицы требуется записать/прочитать. Тип индекса определяется типом подтаблицы. Если подтаблица описана в [MTL](#) как массив (Array) - Индекс должен быть ординарного типа (Logical или Integer). Если подтаблица описана в MTL как периодическое поле (Period) - **Индекс** может быть любого типа, на котором определены отношения больше/меньше (String, Integer, Numeric, Logical, Date). Индексы всех элементов подтаблицы должны иметь один и тот же тип.

В данном свойстве **Индекс** - является физическим номером элемента, т.е. его значение не изменится, если на подтаблицу наложен фильтр, в отличие от логического индекса, используемого в свойстве [ПунктыПоНомеру/ItemsByNumber](#).

Назначение

Многозначное поле произвольного типа (определяемого в файле MTL) позволяет прочитать или записать конкретный элемент подтаблицы (это может быть структура или поле некоторого типа, в зависимости от использованного в MTL ключевого слова).

Данное поле является свойством, используемым по умолчанию. Это означает, что ТБ.Скрипт допускает обращение к элементам подтаблицы без обязательного указания поля Items. Например, две следующие записи равнозначны:

```
Накладная.Позиции.Items[i].Товар  
Накладная.Позиции[i].Товар
```

Несмотря на удобство краткой записи, она может при определенных условиях неправильно толковаться компилятором в случае динамической компоновки (позднего связывания), поддерживаемой ТБ.Скриптом.

Например, предположим, что в коде используется переменная **X** базового класса **Record** для работы с экземплярами документов различных типов, причем в документах имеется табличная часть **Позиции**. Поскольку переменная имеет абстрактный (неконкретный) тип **Record**, строка кода:

```
X.Позиции[i]
```

не может быть проверена на правильность на стадии компиляции (программе неизвестно, какие конкретные типы документов будут присвоены переменной X, и свойство **Позиции** отсутствует в классе **Record**), в связи с чем название свойства **Позиции** записывается в генерируемый двоичный код в виде строки. Проверка на наличие такого свойства у объекта, содержащегося в X, откладывается до момента исполнения данной строки кода. Отсюда становится ясно, почему этот механизм называется "поздним связыванием". В результате, на стадии выполнения приведенной строки программа попытается прочитать свойство **Позиции[i]** для некоторого экземпляра документа, скажем, класса **Накладная**. В классе **Накладная** свойство **Позиции** является структурной подтаблицей, то есть выражение "X.Позиции" возвращает экземпляр объекта описываемого класса **Подтаблица**. Следующие за этой строкой квадратные скобки с индексом оказываются с точки зрения системы лишними - подтаблица есть только одна, а в коде делается попытка обратиться к i-ой подтаблице. Если бы в коде была использована полная форма записи "X.Позиции.Items[i]", то ошибка не возникла бы.

Если в рассматриваемом примере переменную X описать конкретным типом **Накладная**, то строка "X.Позиции[i]" еще на стадии компиляции преобразуется в полную форму, подразумеваемую по умолчанию, то есть - "X.Позиции.Items[i]".

Пример

```
proc p1(N: Пример.Накладная);  
var i: integer;  
  -- цикл по позициям  
  for i = 1..N.Позиции.Количество do  
    -- проверяем количество каждого товара  
    if N.Позиции.Пункты[i].Количество = 0 then  
      N.Позиции.Пункты[i].Количество = 1;  
    end;  
  end;  
end;
```

end;

Описание

```
ПунктыПоНомеру(Номер:Целое): Вариант;  
ItemsByNumber(Number:Integer): Variant;
```

Аргументы

Номер - определяет, какой элемент подтаблицы требуется записать/прочитать. Фактически это порядковый номер элемента.

Назначение

Позволяет прочитать и записать конкретный элемент подтаблицы по его *логическому* порядковому номеру.

Если на подтаблицу наложен фильтр, то номер элемента изменяется. Например, в подтаблице физически имеется три элемента с номерами 1, 2 и 3. Допустим на таблицу наложен фильтр, которому не удовлетворяет второй элемент. В этом случае на логическом уровне в подтаблице останется два элемента с порядковыми номерами: 1 (соответствует физическому 1 элементу подтаблицы) и 2 (соответствует физическому 3 элементу подтаблицы).

В связи с вышеизложенным *при использовании циклов в отфильтрованных подтаблицах следует пользоваться* только данным свойством **ItemsByNumber**. Если фильтр отсутствует, то физические и логические индексы совпадают, поэтому выбор свойства непринципиален и можно воспользоваться свойством [Пункты/Items](#), где используются физические индексы.

Пример

```
func Редактирование(Doc :Document; FieldName :String) :Integer;  
    var КоличПоз, j:Integer;  
    var Field:Subtable;  
    var dat:Date[]; -- массив дат  
    var val:Numeric[]; -- массив значение за даты  
  
    Field = Subtable(Doc.GetField(FieldName));  
  
    КоличПоз=Field.Count;  
    for j=1..КоличПоз do  
        dat[j]=Field.IndexByNumber[j];    -- дата  
        val[j]=Field.ItemsByNumber[j];  
        -- значение за эту дату  
    end;  
  
    -- вызываем прикладную функцию  
    return Проверить(dat,val,КоличПоз);  
end;
```

Описание

Фильтр :Строка;
Filter :String;

Назначение

Свойство предназначено для установки фильтра на подтаблицу.

Данное свойство доступно только для подтаблицы, полученной в результате вызова метода [Subtable.CreateSubtableView](#).

Предупреждение. При попытке изменить или получить свойства **Фильтр/Filter** обычной подтаблицы выдается ошибка.

Описание

Очистить;
Clear;

Назначение

Очищает подтаблицу (удаляет все элементы).

Пример

```
-- "правильный" способ очистки подтаблицы  
proc ResetDoc(N: Пример.Накладная);  
    N.Позиции.Clear;  
end;
```

Описание

```
Удалить(Индекс: Целое; {Количество :Целое});  
Delete(Index :Integer; {Count :Integer});
```

Аргументы

Индекс - определяет позицию экземпляра (номер элемента в многозначной структуре или многозначном поле), который требуется удалить из подтаблицы.

Количество - необязательный параметр, указывает сколько элементов надо удалить, начиная с элемента, заданного в параметре **Индекс**. Если сумма элементов, заданная в параметрах **Индекс** и **Количество**, превышает количество элементов в подтаблице, то будет сгенерирована ошибка.

Назначение

Удаляет из подтаблицы одно значение, находящееся по указанной позиции. Если позиция выходит за допустимые пределы, генерируется исключительная ситуация.

Пример

```
-- способ очистки подтаблицы "в лоб": работает, но...  
-- лучше использовать метод N.Позиции.Clear  
proc ClearTable(N: Пример.Накладная);  
  -- пока есть позиции  
  while N.Позиции.Количество <> 0 do  
    -- удаляем их одну за другой  
    N.Позиции.Delete(1);  
  end;  
end;
```

Описание

```
СуммаПоля(Имя: Строка): Число;  
SumOfField(Name: String): Numeric;
```

Аргументы

Имя - название многозначного поля, значения которого требуется просуммировать. Поле должно быть числового (или приводимого к нему) типа.

Назначение

Суммирует значения указанного поля по всей подтаблице. Если имя поля указано неверно, генерируется исключительная ситуация.

Пример

```
-- подсчитать сумму Итого по накладной  
func GetSum(N: Пример.Накладная): Numeric;  
    return N.Позиции.SumOfField("Сумма");  
end;
```

Описание

Вставить(Позиция: Целое) :Структура;
Insert(Index: Integer) :Structure;

Аргументы

Позиция - определяет позицию (индекс элемента), в которую требуется вставить новый экземпляр структуры или поля.

Назначение

Вставляет в подтаблицу в указанную позицию новый (пустой) экземпляр структуры или поля. Если позиция выходит за допустимые пределы, генерируется исключительная ситуация. Индекс не должен быть меньше 1 или более чем на 1 превышать размер подтаблицы (количество имеющихся экземпляров многозначной структуры или многозначного поля).

Функция возвращает указатель на новую добавленную структуру.

Внимание. У наследников класса Subtable функция Insert возвращает объекты конкретных подклассов Structure, разумеется, если элементы подтаблицы - структуры.

Пример

```
proc p1(N: Пример.Накладная);
var S :Пример.Накладная.Позиции;
  -- если N.Позиции.Количество = 0
  -- то в следующей строке получим...
  S = N.Позиции.Вставить(2); -- сообщение об ошибке
  -- здесь можно было бы написать 1, чтобы "вставить"
  -- новый экземпляр в начало подтаблицы

  -- для заполнения полей структуры можно было бы
  -- применить код:
  -- N.Позиции[2].Товар = "Услуги по установке";
  -- но то же самое можно сделать и с помощью S:
  S.Товар = "Услуги по установке";
  -- этот вариант быстрее

  N.Позиции[2].Цена = 15;
  N.Позиции[2].Валюта = "USD";
end;
```

Описание

Добавить :Целое;
Add :Integer;

Назначение

Добавляет в подтаблицу новый (пустой) экземпляр структуры или поля. Возвращает индекс нового элемента.

Пример

```
proc p1(N: Пример.Накладная);  
var i: integer;  
  if N.Позиции.Количество = 0 then  
    i = N.Позиции.Добавить;  
    N.Позиции[i].Товар = "Услуги по установке";  
    N.Позиции[i].Цена = 15;  
    N.Позиции[i].Валюта = "USDпокурсу";  
  end;  
end;
```

Описание

ДобавитьДоп :Структура;
AddEx :Structure;

Назначение

Добавляет в подтаблицу новый (пустой) экземпляр структуры или поля. Возвращает указатель на добавленную структуру.

Внимание. У наследников класса Subtable функция AddEx возвращает объекты конкретных подклассов Structure, разумеется, если элементы подтаблицы - структуры.

Пример

```
proc p1(N: Пример.Накладная);  
var S :Пример.Накладная.Позиции;  
    S = N.Позиции.ДобавитьДоп;  
    S.Валюта = "РУБ";  
    S.Количество = 1;  
end;
```


Описание

```
СоздатьПредставлениеПодтаблицы([Фильтр :Строка]; [Порядок :Строка]) :Подтаблица;  
CreateSubtableView([Filter :String]; [Order :String]) :Subtable;
```

Аргументы

Фильтр - необязательный строковый параметр, устанавливает фильтр на подтаблицу;

Порядок - необязательный строковый параметр, задающий сортировку подтаблицы.

Назначение

Данный метод возвращает подтаблицу, отсортированную в соответствии с заданной сортировкой (поле **Порядок**) и отфильтрованную в соответствии с установленным фильтром (поле **Фильтр**).

У наследников класса **Subtable** функция возвращает объект своего класса.

Предупреждение. Если у подтаблицы установлены сортировка и/или фильтр, то пользоваться свойством [Items](#) нельзя, т.к. это может привести к ошибкам или к неправильному результату. Поэтому доступ к элементам подтаблицы с фильтром и/или сортировкой должен осуществляться только с помощью свойства [ItemsByNumber](#) или с помощью конструкции `vSubtable.Items[vSubtable.IndexByNumber [i]]`.

Класс *Структура* (*Structure*), производный от класса *Объект*, используется для работы со структурными полями, входящими в состав документов. Структура используется для объединения нескольких логически связанных полей в одну группу. Например, структура Контрагент с информацией о предприятии может входить в состав документов Поставщик, Покупатель, Заемщик, Кредитор и т.д.

Внимание! Создать объект класса *Структура* напрямую нельзя. Структуры создаются только внутри системы и используются как свойства других объектов (класса [ПодтаблицаКартотеки](#)), причем такие свойства имеют атрибут "только чтение". Вместе с тем, программа на ТБ.Скрипт может получить доступ к объекту *Структура*, являющемуся свойством другого объекта, и изменять свойства самой структуры.

Класс *Структура* наследует от родительского класса [Объект](#) следующие свойства и методы:

-  [Поле ИмяКласса / ClassName](#)
-  [Поле ПроектКласса / ClassProject](#)
-  [Поле РодительскийКласс / ParentClass](#)
-  [Поле ИнформацияОКлассе / ClassInfo](#)
-  [Функция УнаследованОт / InheritsFrom](#)
-  [Функция ПорожденныеКлассы / ChildClasses](#)
-  [Функция ВзятьПолеКласса / GetClassField](#)
-  [Процедура УстановитьПолеКласса / SetClass](#)
-  [Поле ТипКласса / ClassType](#)
-  [Процедура Присвоить / Assign](#)
-  [Функция Вычислить / Evaluate](#)
-  [Функция ВзятьПоле / GetField](#)
-  [Процедура УстановитьПоле / SetField](#)

Класс *Транзакция* (*Transaction*), производный от родительского класса [Объект](#), используется для пакетной работы с записями. Более подробные сведения о транзакциях приводятся в разделе [Транзакции и изоляции](#).

В классе *Транзакция* описываются следующие свойства:



[Функция Создать / Create](#)



[Поле Записи / Records](#)



[Процедура Применить / Apply](#)



[Процедура Отменить / Cancel](#)

Работать с транзакциями можно также с помощью методов [НачатьТранзакцию](#), [ЗавершитьТранзакцию](#), [ОтменитьТранзакцию](#) класса *Система*.

Описание

Записи :Класс [] Запись;
Records :Class [] Record;

Назначение

Возвращает массив ссылок на классы записей, для которых создана данная транзакция.

Поле доступно только на чтение.

Пример

```
proc ДобавитьЗапись (Т :Transaction);
var x :Document;
  -- создаем документ того класса, для которого
  -- была открыта транзакция
  x = Т.Records[1].Create;
  -- инициализируем поля
  x.Дата = Сегодня;
  -- записываем документ
  x.Post;
  -- фактически запись нового документа в базу
  -- будет осуществлена при последующем вызове Т.Apply
end;
```

Описание

```
Отменить [(ОткрытьЗаново :Логическое)];  
Cancel [(Reopen :Logical)];
```

Аргументы

ОткрытьЗаново - необязательный параметр, указывающий на необходимость заново открыть транзакцию после отмены изменений. Значение TRUE означает, что необходимо заново открыть транзакцию. Если при вызове процедуры параметр опущен, то он считается равным FALSE, то есть транзакция закрывается.

Назначение

Отменяет все изменения, произведенные ранее в рамках данного объекта транзакции после предыдущего вызова метода [Apply](#) или создания объекта.

Пример

```
проц Кнопка1ПриНажатии(Отправитель :Кнопка);  
var Tr :Транзакция;  
  Tr = Транзакция.Создать([Новый.Документы.Товар]);  
  try  
    -- добавляем 1-ую запись  
    ДобавитьЗапись(Tr); -- см. пример  
    -- добавляем 2-ую запись  
    ДобавитьЗапись(Tr); -- см. пример  
    -- записываем новые записи в базу  
    Tr.Применить;  
  except  
    -- произошли ошибки при добавлении одной из записей  
    -- отменяем все добавления  
    Tr.Отменить;  
  end;  
конец;
```

Описание

```
Применить [(ОткрытьЗаново :Логическое)];  
Apply [(Reopen :Logical)];
```

Аргументы

ОткрытьЗаново - необязательный параметр, указывающий на необходимость заново открыть транзакцию после записи изменений. Значение TRUE означает, что необходимо заново открыть транзакцию. Если при вызове процедуры параметр опущен, то он считается равным FALSE, то есть транзакция закрывается после записи.

Назначение

Записывает все изменения, произведенные в рамках данного объекта транзакции, в базу. Если метод **Apply** не был вызван для текущего объекта транзакции до его уничтожения (например, при выходе из области видимости), то система автоматически вызовет для него метод [Cancel](#) во время уничтожения, то есть отменит все изменения (вплоть до предыдущего вызова **Apply** или создания объекта).

Пример

```
проц Кнопка1ПриНажатии(Отправитель :Кнопка);  
  var Tr :Транзакция;  
  Tr = Транзакция.Создать([Новый.Документы.Товар]);  
  ДобавитьЗапись(Tr); -- см. пример  
  -- записываем новую запись в базу  
  Tr.Применить;  
конец;
```

Функция Создать / Create

Описание

```
Создать([КлассыЗаписей :Класс[] Запись]) :Транзакция;  
Create([RecordClasses :Class[] Record]) :Transaction;
```

Аргументы

КлассыЗаписей - массив ссылок на классы записей, для которых будет выполняться транзакция.

Назначение

Создает объект класса **Транзакция** и открывает транзакцию для указанных классов записей.

Если массив опущен, транзакция открывается на все классы документов всех проектов, для которых была создана текущая информационная база.

Возвращает ссылку на созданный объект.

Пример











```
-- обработчик нажатия кнопки в картотечной форме  
proc кнУдалить_ПриНажатии (Sender :Button);  
var T : Transaction;  
var i :Integer;  
  if (Cardfile.SelectedCount > 0) then  
    if ВопрДаОтказ('Удалить выделенные записи?') = cmOk then  
      -- начинаем транзакцию  
      T = Transaction.Create([Тест.Пример.Счет]);  
      try  
        for i = 1..Cardfile.SelectedCount do  
          Hint('Удаление счета номер '  
            + Str(Cardfile.Selected[i].Номер) + '...');  
          Cardfile.Selected[i].Delete;  
        end;  
        Hint('Обновление картотеки...');  
        -- записываем изменения в базу  
        T.Apply;  
      except  
        -- возникли ошибки, все изменения отклоняются  
        T.Cancel;  
        raise;  
      end;  
    end;  
  end;  
end;
```

Класс *Экспортер/Exporter* является производным от родительского класса [Объект](#) и наследует от него все свойства и методы. Класс позволяет экспортировать выборки записей (документов) определенных типов во внешний файл. Для файлов экспорта в программе поддерживаются следующие форматы: tbc, tbd, xml.

Объекты класса *Экспортер* создаются конструктором *Create*. Файл и формат экспорта указывается в методе *НачатьЭкспорт*. В конце экспорта должен быть выполнен метод *ЗакончитьЭкспорт*. Иначе экспорт будет не закончен и файл экспорта создан не будет.

Внимание. Если осуществляется экспорт нескольких записей (запросов) в цикле, то его следует разместить внутри блока *НачатьИзоляцию ... ЗакончитьИзоляцию* ([см. пример](#)).

В классе *Экспортер* используются следующие свойства:

-  [Функция Создать / Create](#)
-  [Процедура НачатьЭкспорт / StartExport](#)
-  [Процедура ЗакончитьЭкспорт / EndExport](#)
-  [Процедура УстановитьЭкспортируемыеПоля / SetExportedFields](#)
-  [Процедура ЭкспортЗаписи / ExportRecord](#)
-  [Процедура ЭкспортЗапроса / ExportQuery](#)
-  [Поле КодоваяСтраница / CodePage](#)
-  [Поле Комментарии / Comments](#)
-  [Поле ЭкспортироватьСсылки / FollowLinks](#)
-  [Поле ИгнорироватьВторичныйКлюч / IgnoreForeignKey](#)

и перечислимые типы

-  [Тип ТипКодовойСтраницы / CodePageType](#)

Описание

ИгнорироватьВторичныйКлюч :Логическое;
IgnoreForeignKey :Logical;

Назначение

Свойство логического типа позволяет установить или узнать, нужно ли игнорировать вторичный ключ. По умолчанию значение свойства равно False.

Если значение свойства равно True, то при экспорте записей в качестве внешнего ключа будет выступать значение служебного поля [ExtID](#) записи. При его отсутствии в качестве внешнего ключа берется значение поля **DocID**, даже если в записи описан внешний ключ [ForeignKey](#).

Поле доступно на чтение и запись.

Пример

```
var Exp :Exporter;  
...  
Exp.IgnoreForeignKey = False;
```

Описание

КодоваяСтраница : [Экспортер.ТипКодовойСтраницы](#);

CodePage : [Exporter.CodePageType](#);

Назначение

Свойство позволяет узнать и установить кодировку, в которой будет записан создаваемый файл. Свойство доступно только после вызова метода [НачатьЭкспорт / StartExport](#).

Поле доступно на чтение и запись и может содержать одно из значений перечислимого типа [ТипКодовойСтраницы / CodePageType](#). По умолчанию используется значение *cpUnicode*, т.е. создаваемый файл будет записан в кодировке Юникод.

Описание

Комментарии :Строка;
Comments :String;

Назначение

Поле предназначено для чтения/записи комментариев.

Пример

```
var Exp :Exporter;  
--...  
Exp.Comments;
```

Описание

ЭкспортироватьСсылки :Логическое;
FollowLinks :Logical;

Назначение

Свойство позволяет установить или определить, следует ли кроме записей, экспортируемых непосредственно, также экспортировать и те записи (True), на которые имеются ссылки из числа экспортируемых записей, или нет (False).

Пример

```
proc QueryBtnClick(Sender :Object);
var Q :Query;
var Exp :Exporter;
var fName :String;
-- инициализируем выборку документов конкретного типа
  Q = Query.Create([Test.Dbl.Full]);
-- задаем условия отбора документов
  Q.Filter = 'Номер > 300';
-- формируем выборку
  Q.Select;
-- предлагаем пользователю выбрать имя файла для экспорта
  if ChooseFile(fName, 'Выбор файла',
    'Данные в формате tbc|*.tbc') = cmOk then
    -- создаем объект-экспортер
    Exp = Exporter.Create;
    Exp.StartExport(fName, 'tbc');
    Exp.FollowLinks=False;
    -- экспортируем всю выборку, за исключением записей, на
    -- которые имеются ссылки из числа экспортируемых записей
    Exp.ExportQuery(Q);
    Exp.EndExport;
  end;
end;
```

Описание

```
ЗакончитьЭкспорт;  
EndExport;
```

Назначение

Завершает процедуру экспорта, проведенную с помощью текущего объект класса **Экспортер**. Процедура обязательно должна быть вызвана - в противном случае файл экспорта не будет создан.

Пример

```
proc ExportRecord(FileName:String ;aRecord:Record);  
var Exp :Exporter;  
  Exp = Exporter.Create;  
  Exp.StartExport(FileName, 'tbc');  
  Exp.ExportRecord(aRecord);  
  Exp.EndExport;  
end;
```

Описание

```
НачатьЭкспорт(Файл:Строка; Формат:Строка; {ЭкспортироватьСсылки:Логическое}) :Экспортер;  
StartExport(File:String; Format:String; {FollowLinks:Logical}) :Exporter;
```

Аргументы

Файл - имя файла, в который будет проводиться экспорт;

Формат - формат формируемого файла в виде строки с расширением, например, "tbc" или "tbd";

ЭкспортироватьСсылки - определяет, следует ли кроме записей, экспортируемых непосредственно, также экспортировать и те, на которые имеются ссылки из числа экспортируемых записей.

Назначение

Инициализирует объект класса **Экспортер**, с помощью которого затем осуществляется экспорт выборки документов в файл.

Внимание. Вызов процедуры **НачатьЭкспорт** не открывает автоматически изоляцию на экспортируемые записи, поэтому об этом должен позаботиться разработчик.

Пример

```
proc QueryBtnClick(Sender :Object);  
var Q :Query;  
var Exp :Exporter;  
var fName :String;  
-- инициализируем выборку документов конкретного типа  
Q = Query.Create([Test.Dbl.Full]);  
-- задаем условия отбора документов  
Q.Filter = 'Номер > 300';  
-- формируем выборку  
Q.Select;  
-- предлагаем пользователю выбрать имя файла для экспорта  
if ChooseFile(fName, 'Выбор файла',  
  'Данные в формате tbc|*.tbc') = cmOk then  
  -- создаем объект-экспортер  
  Exp = Exporter.Create;  
  Exp.StartExport (fName, 'tbc');  
  -- экспортируем всю выборку  
  Exp.ExportQuery(Q);  
  Exp.EndExport;  
end;  
end;
```

Описание

```
УстановитьЭкспортируемыеПоля(ДляЧего:Класс; Поля :Строка[]);  
SetExportedFields(ForWhat:Class; Fields: String[]);
```

Аргументы

ДляЧего - класс записи или класс структуры, для которой указывается список экспортируемых полей. Таким образом, можно установить этот список как для класса документов, так и для классов вложенных в него структур;

Поля - массив с названиями экспортируемых полей для списка документов или структур.

Назначение

Устанавливает перечень экспортируемых полей. Если необходимо экспортировать все поля, вызывать процедуру не надо.

Данный метод можно вызвать только после вызова метода [НачатьЭкспорт](#).

Пример

```
proc ExportRecordName(FileName:String ; aRecord:Record);  
var Exp :Exporter;  
  Exp = Exporter.Create;  
  Exp.StartExport(FileName, 'tbc');  
  Exp.SetExportedFields(aRecord.ClassType, ['Name', 'Comment']);  
  Exp.ExportRecord(aRecord);  
  Exp.EndExport;  
end;
```

Описание

```
ЭкспортЗаписи(Зап :Запись);  
ExportRecord (Rec :Record);
```

Аргументы

Зап - ссылка на запись, которую требуется экспортировать.

Назначение

Экспортирует указанный документ (запись) в файл (файл был задан при [создании объекта Экспортер](#)). Если необходимо экспортировать не все поля записи, перечень таких полей можно задать с помощью метода [УстановитьЭкспортируемыеПоля](#).

Если осуществляется экспорт нескольких записей в цикле, то этот фрагмент кода программы рекомендуется выполнять внутри изоляции (см. [Транзакции и изоляции](#)), то есть обложить операторными скобками [НачатьИзоляцию](#) ... [ЗавершитьИзоляцию](#).

Пример

```
proc RecordsBtnClick (Sender :Object);  
var Exp :Exporter;  
var fName :String;  
var Cnt, I :Integer;  
var Doc :Record;  
-- предлагаем пользователю выбрать имя файла для экспорта  
if ChooseFile(fName, 'Выбор файла',  
  'Данные в формате tbc|*.tbc') <> cmOk then  
  return;  
end;  
-- предлагаем ввести количество экспортируемых записей  
if Input(Cnt, 'Количество записей') <> cmOk then  
  return;  
end;  
-- создаем объект-экспортер  
Exp = Exporter.Create;  
Exp.StartExport(fName, 'tbc')  
BeginIsolation([Dbl.Full]);  
try  
  -- цикл по числу экспортируемых записей  
  for I = 1..Cnt do  
    -- предлагаем выбрать запись в окне картотеки  
    if Basal.Full.ExecuteCard(Doc) = cmOk then  
      -- если запись выбрана, экспортируем ее  
      Exp.ExportRecord(Doc);  
    end;  
  end;  
finally  
  Exp.EndExport;  
  EndIsolation;  
end;  
end;
```


Описание

```
ЭкспортЗапроса(Запрос: Запрос);  
ExportQuery    (Query : Query);
```

Аргументы

Запрос - объект класса [Запрос](#). К моменту вызова процедуры запрос должен быть уже выполнен, то есть содержать набор документов.

Назначение

Экспортирует выборку документов, содержащихся в результате запроса, в файл (файл был задан при [создании объекта Экспортер](#)).

Если необходимо экспортировать не все поля классов запроса, перечень таких полей можно задать с помощью метода [УстановитьЭкспортируемыеПоля](#) (по одному вызову на каждый класс записей из запроса).

Пример

```
Назначение  
Пример  
proc QueryBtnClick(Sender :Object);  
var Q :Query;  
var Exp :Exporter;  
var fName :String;  
-- инициализируем выборку документов конкретного типа  
  Q = Query.Create([Test.Dbl.Full]);  
-- задаем условия отбора документов  
  Q.Filter = 'Номер > 300';  
-- формируем выборку  
  Q.Select;  
-- предлагаем пользователю выбрать имя файл для экспорта  
  if ChooseFile(fName, 'Выбор файла',  
    'Данные в формате tbc|*.tbc') = cmOk then  
    -- создаем объект-экспортер  
    Exp = Exporter.Create;  
    Exp.StartExport(fName, 'tbc')  
    -- экспортируем всю выборку  
    Exp.ExportQuery(Q);  
    Exp.EndExport;  
  end;  
end;
```

Перечислимый тип *ТипКодовойСтраницы / CodePageType*, определенный в классе *Экспортер* позволяет установить кодировку, в которой будет записан создаваемый файл.

В данном типе определены следующие константы:

- **cpWindows** - кодировка Windows 1251;
- **cpDOS** - DOS-кодировка (cp866);
- **cpUnicode** - Юникод;
- **cpUTF8** - кодировка UTF8.

Вышеприведенные константы используются в свойстве [КодоваяСтраница / CodePage](#). По умолчанию в свойстве **КодоваяСтраница** используется значение **cpUnicode**.

Описание

Создать :Экспортер;
Create :Exporter;

Назначение

Создает новый объект класса **Экспортер**, с помощью которого затем осуществляется экспорт выборки документов в файл.

Пример

```
proc QueryBtnClick(Sender :Object);
var Q :Query;
var Exp :Exporter;
var fName :String;
-- инициализируем выборку документов конкретного типа
  Q = Query.Create([Test.Dbl.Full]);
-- задаем условия отбора документов
  Q.Filter = 'Номер > 300';
-- формируем выборку
  Q.Select;
-- предлагаем пользователю выбрать имя файла для экспорта
  if ChooseFile(fName, 'Выбор файла',
    'Данные в формате tbc|*.tbc') = cmOk then
    -- создаем объект-экспортер
    Exp = Exporter.Create;
    Exp.StartExport(fName, 'tbc');
    -- экспортируем всю выборку
    Exp.ExportQuery(Q);
    Exp.EndExport;
  end;
end;
```

Все классы, производные от родительского класса [Объект](#), сгруппированы по своему назначению на группы.

В группу классов для работы с сервером расчетов входят следующие:

- [Признак / Sign](#)
- [СтруктураПризнака / SignStruct](#)
- [ПодтаблицаПризнака / SignSubtable](#)
- [ПланСчетов / AccPlan](#)
- [Счет / Account](#)
- [ИсточникУчета / AccSource](#)
 - [Справочник / Reference](#)
 - [Журнал / Journal](#)
- [ОбластьУчета / AccDomain](#)
- [Полупроводка / HTrans](#)
- [Проводка / Trans](#)
- [Операции / Operation](#)
- [ЗапросПолупроводок / HTransQuery](#)
- [ЗапросПроводок / TransQuery](#)
- [ЗапросСчетов / AccQuery](#)
- [ЗапросАналитики / AnalitQuery](#)

Журнал / Journal

Класс *Журнал / Journal* обеспечивает программный доступ ко всем типам [журналов](#) (текстовым, табличным и картотечным), описанных в структуре учета, и используется в качестве базового класса для всех классов журналов. Класс *доступен только на клиенте*.

Класс *Справочник / Reference* обеспечивает программный доступ к [аналитическим справочникам](#), описанным в структуре учета. Класс *доступен только на клиенте*.

Классы *Журнал* и [Справочник](#) имеют общего предка - класс [ИсточникУчета](#) - и наследуют все имеющиеся у него свойства.

Непосредственно в данном классе определены следующие свойства:

- [Поле ОбластьУчета / AccDomain](#)
- [Поле Разрешен / Enabled](#)

Описание

ОбластьУчета : [ОбластьУчета](#);
AccDomain : [AccDomain](#);

Назначение

Возвращает ссылку на объект класса [ОбластьУчета](#).

Поле доступно только на чтение.

Описание

Разрешен :Логическое;
Enabled :Logical;

Назначение

Свойство позволяет программно определять, включен или отключен журнал в расчетной сессии. Если свойство равно False, значит в [настройках расчетной базы](#) журнал отключен, т.е. его обработка запрещена и он не виден в списке журналов, а при попытке просмотра проводок в картотеке журнала выдается сообщение "Обработка журнала запрещена". Если свойство равно True, то журнал виден в списке журналов, его обработка разрешена, и можно видеть его проводки.

Свойство доступно только на клиенте.

Пример




```
func ДобавитьЗакладкуЖурнала(аЖурнал :Journal; аЗаписиЖурнала :Record[]):Logical;
var I, vCount :Integer;
var локФрейм :TemplateFrame;
var локФорма :CardForm;
var локФильтр :String;
var vQuery :Query;

Result = False;
if аЖурнал.Cardform <> nil then
    локФильтр = "";
    for I = 1..LengthOfArray(аЗаписиЖурнала) do
        локФильтр = СложитьСтрокиФильтраПоИЛИ([локФильтр,
            СложитьСтрокиФильтраПоИ([ 'ClassType='+аЗаписиЖурнала[I].ClassProject+'.'+
            аЗаписиЖурнала[I].ClassName,
        end;
    локФорма = CreateJournal(аЖурнал.Name);
    локФорма.CardFile.Filter =
        СложитьСтрокиФильтраПоИ([локФорма.CardFile.Filter, локФильтр]);

    vQuery = Query.Create(локФорма.Query.Documents);
    vQuery.Filter = локФорма.CardFile.Filter;
    vQuery.Select;
    if not vQuery.EOF then
        vCount = vQuery.Count;
    end;
    if vCount>0 then
        Result = аЖурнал.Enabled;
        локФрейм = ФреймСодержимое.AddFrame;
        локФрейм.Name = аЖурнал.Name+аЖурнал.Description;
        локФрейм.Caption = аЖурнал.Description;if(Result, "", " (Отключен)");
        локФрейм.Bevel = Template.DefaultBevel;
        LoadForm(локФрейм, локФорма);
    end;
end;
end;
```

Класс *ЗапросАналитики / AnalitQuery*, производный от родительского класса [Объект](#), предназначен для извлечения информации об аналитических признаках. С помощью данного класса формируется перечень элементов аналитических справочников, отвечающий условиям, указанным при создании запроса.

Непосредственно в данном классе определены следующие новые и переопределены наследуемые свойства:

-  [Функция Создать / Create](#)
-  [Поле Справочник / Reference](#)
-  [Поле Счета / Асс / УсловиеНаСчета](#)
-  [Поле ПараметрыСчетов / AccParam](#)
-  [Поле НачальнаяДата / BegDate](#)
-  [Поле КонечнаяДата / EndDate](#)
-  [Поле ВключатьУдаленные / IncludeDeleted](#)
-  [Поле ТолькоИспользованные / OnlyUsed](#)
-  [Функция Выбор / Select](#)
-  [Поле Количество / Count](#)
-  [Поле ВНачале / BOF](#)
-  [Поле ВКонце / EOF](#)
-  [Поле Текущий / Current](#)
-  [Процедура Первый / First](#)
-  [Процедура Последний / Last](#)
-  [Процедура Следующий / Next](#)
-  [Процедура Предыдущий / Previous](#)
-  [Процедура ПереместитьНа / MoveBy](#)

Описание

ВключатьУдаленные : Логическое;
IncludeDeleted : Logical;

Назначение

Свойство управляет видимостью в запросе удаленных элементов справочника, по умолчанию значение свойства равно False, т.е. в запрос не включаются ранее удаленные элементы справочника.

Внимание. При удалении элементов справочника их физического удаления не происходит, они только помечаются как удаленные (потерянные).

При значении свойства, равного True, происходит добавление в запрос ранее удаленных элементов справочника при условии, что свойство [ТолькоИспользованные](#) = False;.

Пример

```
var Q : AnalitQuery;  
Q = AnalitQuery.Create("Корреспондент");  
-- создаем запрос по справочнику корреспондент  
...  
Q.OnlyUsed = False;  
Q.IncludeDeleted = True;
```

Описание

ВКонец : Логическое;
EOF : Logical;

Назначение

Возвращает значение TRUE после попытки переместить внутренний указатель запроса (доступный через свойство [Current](#)) за последний аналитический признак из перечня признаков, и FALSE в противном случае.

Если в запросе 0 элементов, то оба свойства и [BOF](#), и **EOF** равны TRUE. Данный способ позволяет проверить, не пуст ли результат запроса, причем этот способ является более предпочтительным, чем проверка свойства [Count](#), поскольку выполняется гораздо быстрее.

Поле доступно только на чтение.

Пример

См. также [Примеры использования запросов по аналитике](#).

Описание

ВНачале : Логическое;
BOF : Logical;

Назначение

Возвращает значение TRUE после попытки переместить внутренний указатель запроса (доступный через свойство [Current](#)) за первый аналитический признак в перечне аналитики, сформированном по запросу, и FALSE в противном случае.

Если в запросе 0 элементов, то оба свойства и **BOF**, и [EOF](#) равны TRUE. Данный способ позволяет проверить, не пуст ли результат запроса, причем этот способ является более предпочтительным, чем проверка свойства [Count](#), поскольку выполняется гораздо быстрее.

Поле доступно только на чтение.

Пример

```
proc RunQuery(Analytics:String);  
  var q : AnalitQuery;  
  q = AnalitQuery.Create(Analytics);  
  -- делаем текущим последний признак из запроса  
  q.Last;  
  -- цикл по признакам в обратном направлении  
  while not q.Bof do  
    -- обработка  
    ...  
    q.Previous;  
  end;  
end;
```

Описание

Количество : Целое;
Count : Integer;

Назначение

Возвращает количество аналитических признаков, попавших в результаты запроса. Поле доступно только на чтение.

Пример

```
proc RunQuery(Analytics:String);  
  var q : AnalitQuery;  
  q = AnalitQuery.Create(Analytics);  
  if q.Count = 0 then  
    return;  
  end;  
  -- дальнейшая обработка  
  ...  
end;
```

Описание

КонечнаяДата :Дата;
EndDate :Date;
ДатаКонца :Дата;

Назначение

Позволяет узнать и изменить конечную дату периода для проводок, из которых будут отбираться использованные аналитические признаки.

Данное поле влияет на запрос только в том случае, если свойство [ТолькоИспользованные](#) равно TRUE.

Пример

```
var q : AnalitQuery;  
  q = AnalitQuery.Create('Корреспондент');  
  -- по справочнику корреспондент  
  q.OnlyUsed = True;  
  -- выбираем только используемые элементы  
  q.BegDate = 01.01.2007;  
  q.EndDate = 01.02.2007;  
  -- в течение января 2007 года  
  q.Select;
```

Описание

НачальнаяДата :Дата;
BegDate :Date;
ДатаНачала :Дата;

Назначение

Позволяет узнать и изменить начальную дату периода для проводок, из которых будут отбираться использованные аналитические признаки.

Данное поле влияет на запрос только в том случае, если свойство [ТолькоИспользованные](#) равно TRUE.

Пример

```
var q : AnalitQuery;  
q = AnalitQuery.Create('Корреспондент');  
-- по справочнику корреспондент  
q.OnlyUsed = True;  
-- выбираем только используемые элементы  
q.BegDate = 01.01.2007;  
q.EndDate = 01.02.2007;  
-- в течение января 2007 года  
q.Select;
```

Описание

Параметры :Строка;
Props :String;

Назначение

Позволяет узнать и изменить условие отбора по параметрам элементов аналитического справочника, указанного для текущего запроса с помощью свойства [Справочник](#).

С помощью данного поля можно, в частности, построить запрос аналитики, в который должны попасть лишь московские контрагенты: "Регион='Москва'", где Регион - поле строкового типа в справочнике контрагентов. Синтаксис условий отбора по параметрам тот же, что и в условиях отбора проводок [по параметрам счетов](#).

Пример

```
proc ИзменитьРазрез (AQ :AnalitQuery; D: Document);  
var S: Признак;  
var Условие: Строка;  
var Док: Накладная;  
    Док = Накладная (D);  
    S = ЗаписьВПризнак(Док.Контрагент);  
    Условие = "Контрагент=" + Str(S);  
    AQ.Props = Условие;  
end;
```

Описание

ПараметрыСчетов :Строка;
AccParam :String;

Назначение

Позволяет узнать и изменить условие отбора аналитических признаков по параметрам счетов, использованным в проводках с соответствующими признаками. Иными словами, в результаты запроса попадают только те признаки, которые были упомянуты в проводках вместе с заданными параметрами счетов.

Данное поле влияет на запрос только в том случае, если свойство [ТолькоИспользованные](#) равно TRUE.

См. также [Синтаксис условий отбора по параметрам счетов](#).

Пример

```
var q : AnalitQuery;  
  q = AnalitQuery.Create('Корреспондент');  
  -- по справочнику корреспондент  
  q.OnlyUsed = True;  
  -- выбираем только используемые элементы  
  q.Acc = '60';  
  -- использованные в полупроводках по 60 счету  
  q.AccParam = 'Сум=USD';  
  -- и когда сумма была в USD  
  q.BegDate = 01.01.2007;  
  q.EndDate = 01.02.2007;  
  -- в течение января 2007 года  
  q.Select;
```


Описание

Справочник :Строка;
Reference :String;

Назначение

Позволяет узнать и изменить название аналитического справочника, по которому строится запрос. Изначально данное свойство инициализируется при создании объекта значением, переданным в качестве параметра функции [Create](#).

Пример

```
Title :Label;  
proc ShowQueryResults(AQ :AnalitQuery);  
    Title.Caption = AQ.Reference;  
    -- ...  
end;
```

Описание

Счета :Строка;
Асс :String;
УсловиеНаСчета :Строка;

Назначение

Позволяет узнать и изменить условие отбора аналитических признаков по счетам, использованным в проводках с соответствующими признаками. Иными словами, в результаты запроса попадают только те признаки, которые были упомянуты в проводках вместе с заданными счетами.

Данное поле влияет на запрос только в том случае, если свойство [ТолькоИспользованные](#) равно TRUE.

См. также [Синтаксис условий отбора проводок по счетам](#).

Пример

```
var q : AnalitQuery;  
q = AnalitQuery.Create('Корреспондент');  
-- по справочнику корреспондент  
q.OnlyUsed = True;  
-- выбираем только используемые элементы  
q.Асс = '60';  
-- использованные в полупроводках по 60 счету  
q.BegDate = 01.01.2007;  
q.EndDate = 01.02.2007;  
-- в течение января 2007 года  
q.Select;
```

Описание

Текущий : [Признак](#);

Current : [Sign](#);

Назначение

Возвращает указатель на текущий аналитический признак из списка тех, что удовлетворяют условиям запроса. Значение поля имеет тип, соответствующий конкретному справочнику, например, валют или единиц измерений.

Поле доступно только на чтение. Для перемещения по аналитическим признакам запроса необходимо использовать методы [Первый](#), [Последний](#), [Следующий](#), [Предыдущий](#), [ПереместитьНа](#).

Пример

```
func ПерваяВалюта(Справочник:Строка) : Валюта;  
    var q : AnalitQuery;  
    q = AnalitQuery.Create(Справочник);  
    return q.Current;  
end;
```

См. также [Примеры использования запросов по аналитике](#).

Описание

ТолькоИспользованные :Логическое;
OnlyUsed :Logical;

Назначение

Данное поле предназначено для построения запросов по аналитике, которая должна обязательно присутствовать в проводках. Когда данное поле имеет значение TRUE, результаты запроса можно ограничить признаками, используемыми в проводках за указанный период ([НачальнаяДата](#) и [КонечнаяДата](#)), по указанным счетам ([Счета](#)), с заданными параметрами ([ПараметрыСчетов](#)).

Когда данное поле имеет значение FALSE, в запрос попадают все элементы аналитического справочника. По умолчанию (сразу после создания объекта), данное свойство имеет значение TRUE.

При использовании класса [ЗапросАналитики](#) в исходном коде типовых операций, значение свойства **ТолькоИспользованные** может быть только TRUE. В противном случае генерируется ошибка. Иными словами, в контексте типовых операций запрос всегда строится только по использованным признакам.

Пример

```
var q : AnalitQuery;  
q = AnalitQuery.Create('Корреспондент');  
-- по справочнику корреспондент  
q.OnlyUsed = True;  
-- выбираем только используемые элементы  
q.Acc = '60';  
-- использованные в полупроводках по 60 счету  
q.AccParam = 'Сум=USD';  
-- и когда сумма была в USD  
q.BegDate = 01.01.2007;  
q.EndDate = 01.02.2007;  
-- в течение января 2007 года  
q.Select;  
...  
q = nil;  
q = AnalitQuery.Create('Валюты');  
-- по справочнику валют  
q.OnlyUsed = False;  
-- выбираем все признаки  
q.Select; -- строим запрос
```

Описание

Выбор;
Select;

Назначение

Строит запрос по аналитическим признакам. Идентификатор аналитического справочника изначально указывается при создании объекта [ЗапросАналитики](#). Дополнительно можно задать и другие свойства запроса. После построения запроса объект предоставляет доступ к сформированному перечню элементов справочника, удовлетворяющих заданным (с помощью свойств объекта) условиям.

Пример

```
q = AnalitQuery.Create('Валюты');  
-- по справочнику валют  
q.OnlyUsed = False;  
-- выбираем все признаки  
q.Select; -- строим запрос
```

Описание

Первый;
First;

Назначение

Осуществляет переход к первому признаку, из числа найденных в результате запроса.

Пример

```
proc RunQuery(Analytics:String);  
  var q : AnalitQuery;  
  q = AnalitQuery.Create(Analytics);  
  -- цикл по признакам, первый заход  
  while not q.Eof do  
    -- обработка  
    ...  
    q.Next;  
  end;  
  -- возвращаем указатель на начало  
  q.First;  
  -- цикл по признакам, второй заход  
  while not q.Eof do  
    -- обработка  
    ...  
    q.Next;  
  end;  
end;
```

Описание

ПереместитьНа (Количество:Целое) ;
MoveBy (Number:Integer) ;

Аргументы

Количество - определяет, на сколько аналитических признаков (элементов аналитического справочника) и в какую сторону следует переместить указатель в перечне признаков, удовлетворяющих условиям запроса.

Назначение

Перемещает внутренний указатель запроса на заданное количество признаков по списку признаков, вошедших в результаты запроса. Аргумент может быть как положительным, так и отрицательным. В последнем случае переход осуществляется в сторону начала списка.

Если указанное количество превышает количество признаков, находящихся в списке от текущей позиции до его начала или конца (в зависимости от знака аргумента), текущим становится соответственно первый или последний признак.

Пример

```
-- перемещаемся через признак вперед  
q.MoveBy(2) ;
```

Описание

Последний;
Last;

Назначение

Осуществляет переход к последнему признаку из числа найденных в результате запроса.

Пример

```
proc RunQuery(Analytics:String);  
  var q : AnalitQuery;  
  q = AnalitQuery.Create(Analytics);  
  -- делаем текущим последний признак из запроса  
  q.Last;  
  -- цикл по признакам в обратном направлении  
  while not q.Bof do  
    -- обработка  
    ...  
    q.Previous;  
  end;  
end;
```


Описание

Предыдущий;
Previous;

Назначение

Осуществляет переход к предыдущему признаку в списке найденных в результате запроса.

Пример

```
proc RunQuery(Analytics:String);  
  var q : AnalitQuery;  
  q = AnalitQuery.Create(Analytics);  
  -- делаем текущим последний признак из запроса  
  q.Last;  
  -- цикл по признакам в обратном направлении  
  while not q.Bof do  
    -- обработка  
    ...  
    q.Previous;  
  end;  
end;
```

Описание

Следующий ;
Next ;

Назначение

Осуществляет переход к следующему признаку в списке найденных в результате запроса.

Пример

См. [Примеры использования запросов по аналитике](#).

Описание

Создать(ИдентификаторСправочника:Строка) : [ЗапросАналитики](#);
Create(ReferenceName:String) : [AnalitQuery](#);

Аргументы

ИдентификаторСправочника - строка с именем аналитического справочника, описанного в [структуре учета](#).

Назначение

Функция создает новый объект класса [ЗапросАналитики](#). В процессе создания объекта свойство [Справочник](#) объекта инициализируется значением, указанным в качестве параметра функции. Непосредственно в данной функции запрос аналитических признаков не выполняется – для построения запроса необходимо воспользоваться процедурой [Выбор](#), предварительно установив требуемые параметры запроса.











Пример

```
var q : AnalitQuery;  
q = AnalitQuery.Create("Валюта");
```

См. также [Примеры использования запросов по аналитике](#).

Класс *ЗапросПолупроводок / HTransQuery*, производный от класса [Объект](#), предназначен для извлечения информации о полупроводках. С помощью данного класса создается набор объектов класса [Полупроводка / HTrans](#), отвечающий условиям, указанным при создании запроса.

Непосредственно в классе *ЗапросПолупроводок* определены следующие свойства:

-  [Функция Создать / Create](#)
-  [Поле Количество / Count](#)
-  [Поле ВНачале / BOF](#)
-  [Поле ВКонце / EOF](#)
-  [Поле Текущая / Current](#)
-  [Процедура Первая / First](#)
-  [Процедура Последняя / Last](#)
-  [Процедура Следующая / Next](#)
-  [Процедура Предыдущая / Previous](#)
-  [Процедура ПереместитьНа / MoveBy](#)

Описание

ВКонец : Логическое;
EOF : Logical;

Назначение

Возвращает значение TRUE после попытки переместить внутренний указатель запроса (доступный через свойство [Current](#)) за конец перечня отобранных полупроводок, и FALSE в противном случае.

Если в запросе 0 элементов, то оба свойства и [BOF](#), и **EOF** равны TRUE. Данный способ позволяет проверить, не пуст ли результат запроса, причем этот способ является более предпочтительным, чем проверка свойства [Count](#), поскольку выполняется гораздо быстрее.

Поле доступно только на чтение.

Пример

См. также [Пример запроса полупроводок](#).

Описание

ВНачале : Логическое;
BOF : Logical;

Назначение

Возвращает значение TRUE после попытки переместить внутренний указатель запроса (доступный через свойство [Current](#)) за начало перечня отобранных полупроводок, и FALSE в противном случае.

Если в запросе 0 элементов, то оба свойства и **BOF**, и [EOF](#) равны TRUE. Данный способ позволяет проверить, не пуст ли результат запроса, причем этот способ является более предпочтительным, чем проверка свойства [Count](#), поскольку выполняется гораздо быстрее.

Поле доступно только на чтение.

Пример

```
proc RunQuery(AccCond:String; ParmCond:String;  
  BegDate:Date; EndDate:Date);  
  var q : HTransQuery;  
  q = HTransQuery.Create(AccCond,  
    ParmCond, BegDate, EndDate);  
  -- делаем текущей последнюю полупроводку запроса  
  q.Last;  
  -- цикл по полупроводкам в обратном направлении  
  while not q.BOF do  
    -- обработка  
    ...  
    q.Previous;  
  end;  
end;
```

Описание

Количество : Целое;

Count : Integer;

Назначение

Возвращает количество полупроводок, попавших в результаты запроса. Поле доступно только на чтение.

Пример

```
proc RunQuery(AccCond:String; ParmCond:String;
  BegDate:Date; EndDate:Date);
  var q : HTransQuery;
  q = HTransQuery.Create(AccCond,
    ParmCond, BegDate, EndDate);
  if q.Count = 0 then
    return;
  end;
  -- дальнейшая обработка
  ...
end;
```

Описание

Текущая : [Полупроводка](#);

Current : [HTrans](#);

Назначение

Возвращает указатель на текущую полупроводку из списка полупроводок, удовлетворяющих условиям запроса. Путем разыменования этой ссылки программист может получить доступ к любому параметру полупроводки.

Поле доступно только на чтение. Для перемещения по полупроводкам запроса необходимо использовать методы [Первая](#), [Последняя](#), [Следующая](#), [Предыдущая](#), [ПереместитьНа](#).

Пример

```
func ПерваяПолупроводка(Счета:Строка; Параметры:Строка;  
    Data1:Data; Data2:Data) : Полупроводка;  
    var q : HTransQuery;  
    q = HTransQuery.Create(Счета, Параметры, Data1, Data2);  
    return q.Current;  
end;
```

См. также [Пример запроса полупроводок](#).

Описание

Первая;
First;

Назначение

Осуществляет переход к первой полупроводке, из числа найденных в результате запроса.

Пример

```
proc RunQuery(AccCond:String; ParmCond:String;  
  BegDate:Date; EndDate:Date);  
  var q : HTransQuery;  
  q = HTransQuery.Create(AccCond,  
    ParmCond, BegDate, EndDate);  
  -- цикл по полупроводкам, первый заход  
  while not q.Eof do  
    -- обработка  
    ...  
    q.Next;  
  end;  
  -- возвращаем указатель на начало  
  q.First;  
  -- цикл по полупроводкам, второй заход  
  while not q.Eof do  
    -- обработка  
    ...  
    q.Next;  
  end;  
end;
```

Описание

ПереместитьНа (Количество:Целое) ;
MoveBy (Number:Integer) ;

Аргументы

Количество - определяет, на сколько полупроводок и в какую сторону следует переместить указатель в перечне полупроводок, удовлетворяющих условиям запроса.

Назначение

Перемещает внутренний указатель запроса на заданное количество полупроводок по списку полупроводок, вошедших в результаты запроса. Аргумент может быть как положительным, так и отрицательным. В последнем случае переход осуществляется в сторону начала списка.

Если указанное количество превышает количество полупроводок, находящихся в списке от текущей позиции до его начала или конца (в зависимости от знака аргумента), текущим становится соответственно первая или последняя полупроводка.

Пример

```
-- перемещаемся через полупроводку вперед  
q.MoveBy(2) ;
```

Описание

Последняя;
Last;

Назначение

Осуществляет переход к последней полупроводке из числа найденных в результате запроса.

Пример

```
proc RunQuery(AccCond:String; ParmCond:String;  
  BegDate:Date; EndDate:Date);  
  var q : HTransQuery;  
  q = HTransQuery.Create(AccCond,  
    ParmCond, BegDate, EndDate);  
  -- делаем текущей последнюю полупроводку запроса  
  q.Last;  
  -- цикл по полупроводкам в обратном направлении  
  while not q.Bof do  
    -- обработка  
    ...  
    q.Previous;  
  end;  
end;
```

Описание

Предыдущая;
Previous;

Назначение

Осуществляет переход к предыдущей полупроводке из числа найденных в результате запроса.

Пример

```
proc RunQuery(AccCond:String; ParmCond:String;  
  BegDate:Date; EndDate:Date);  
  var q : HTransQuery;  
  q = HTransQuery.Create(AccCond,  
    ParmCond, BegDate, EndDate);  
  -- делаем текущей последнюю полупроводку запроса  
  q.Last;  
  -- цикл по полупроводкам в обратном направлении  
  while not q.Bof do  
    -- обработка  
    ...  
    q.Previous;  
  end;  
end;
```

Описание

Следующая ;
Next ;

Назначение

Осуществляет переход к следующей полупроводке из числа найденных в результате запроса.

Пример

См. также [Пример запроса полупроводок](#).

Описание

Создать(Счета:Строка [; Параметры:Строка] [; Дата1:Дата] [; Дата2:Дата]) : [ЗапросПолупроводок](#);
Create(Accounts:String [; Parameters:String] [; Date1:Date] [; Date2:Date]) : [HTransQuery](#);

Аргументы

Счета - строка, содержащая условие отбора полупроводок [по счетам](#);

Параметры - строка с условием отбора полупроводок [по параметрам](#). Если данный аргумент опущен, считается, что никаких ограничений по параметрам нет;

Дата1 - начальная дата интересующего периода. Если опущен аргумент Дата1, началом периода отбора полупроводок будет считаться начало учета т.е. полупроводки запрашиваются, начиная с самой первой;

Дата2 - конечная дата интересующего отчетного периода. Причем, Дата1 входит, а Дата2 не входит в период. Если данный аргумент опущен, полупроводки отбираются до конца учетного периода, т.е. вплоть до самой последней полупроводки.

Назначение

Функция создает новый объект класса [ЗапросПолупроводок](#). В процессе создания объекта выполняется запрос полупроводок, так что проинициализированный объект предоставляет доступ к уже сформированному перечню полупроводок, удовлетворяющих заданным условиям.

Полупроводки перечисляются в хронологическом порядке.











Пример

```
var q : HTransQuery;  
q = HTransQuery.Create("+62!", ":Валюта=USD", 01.01.2007, 01.03.2007);
```

См. также [Пример запроса полупроводок](#).

Класс *ЗапросПроводок / TransQuery*, производный от класса [Объект](#), предназначен для извлечения информации о проводках. С помощью данного класса создается набор объектов класса [Проводка / Trans](#), отвечающий условиям, указанным при создании запроса.

Непосредственно в классе *ЗапросПроводок* определены следующие свойства:

-  [Функция Создать / Create](#)
-  [Поле Количество / Count](#)
-  [Поле ВНачале / BOF](#)
-  [Поле ВКонце / EOF](#)
-  [Поле Текущая / Current](#)
-  [Процедура Первая / First](#)
-  [Процедура Последняя / Last](#)
-  [Процедура Следующая / Next](#)
-  [Процедура Предыдущая / Previous](#)
-  [Процедура ПереместитьНа / MoveBy](#)

Описание

ВКонец : Логическое;
EOF : Logical;

Назначение

Возвращает значение TRUE после попытки переместить внутренний указатель запроса (доступный через свойство [Current](#)) за конец перечня отобранных проводок, и FALSE в противном случае.

Если в запросе 0 элементов, то оба свойства и [BOF](#), и **EOF** равны TRUE. Данный способ позволяет проверить, не пуст ли результат запроса, причем этот способ является более предпочтительным, чем проверка свойства [Count](#), поскольку выполняется гораздо быстрее.

Поле доступно только на чтение.

Пример

```
proc RunQuery(AccCond:String; ParmCond:String;  
    BegDate:Date; EndDate:Date);  
    var q : TransQuery;  
    q = TransQuery.Create(AccCond,  
        ParmCond, BegDate, EndDate);  
    -- цикл по проводкам  
    while not q.Eof do  
        -- обработка  
        ...  
        q.Next;  
    end;  
end;
```


Описание

ВНачале : Логическое;
BOF : Logical;

Назначение

Возвращает значение TRUE после попытки переместить внутренний указатель запроса (доступный через свойство [Current](#)) за начало перечня отобранных проводок, и FALSE в противном случае.

Если в запросе 0 элементов, то оба свойства и **BOF**, и [EOF](#) равны TRUE. Данный способ позволяет проверить, не пуст ли результат запроса, причем этот способ является более предпочтительным, чем проверка свойства [Count](#), поскольку выполняется гораздо быстрее.

Поле доступно только на чтение.

Пример

```
proc RunQuery(AccCond:String; ParmCond:String;  
  BegDate:Date; EndDate:Date);  
  var q : TransQuery;  
  q = TransQuery.Create(AccCond,  
    ParmCond, BegDate, EndDate);  
  -- делаем текущей последнюю проводку запроса  
  q.Last;  
  -- цикл по проводкам в обратном направлении  
  while not q.BOF do  
    -- обработка  
    ...  
    q.Previous;  
  end;  
end;
```

Описание

Количество : Целое;
Count : Integer;

Назначение

Возвращает количество проводок, попавших в результаты запроса. Поле доступно только на чтение.

Пример

```
proc RunQuery(AccCond:String; ParmCond:String;  
  BegDate:Date; EndDate:Date);  
  var q : TransQuery;  
  q = TransQuery.Create(AccCond,  
    ParmCond, BegDate, EndDate);  
  if q.Count = 0 then  
    return;  
  end;  
  -- дальнейшая обработка  
  ...  
end;
```

Описание

Текущая : [Проводка](#);

Current : [Trans](#);

Назначение

Возвращает указатель на текущую проводку из списка проводок, удовлетворяющих условиям запроса. Путем разыменования этой ссылки программист может получить доступ к любому параметру проводки.

Поле доступно только на чтение. Для перемещения по проводкам запроса необходимо использовать методы [Первая](#), [Последняя](#), [Следующая](#), [Предыдущая](#), [ПереместитьНа](#).

Пример

```
func ПерваяПолупроводка(Счета:Строка; Параметры:Строка;  
    Дата1:Дата; Дата2:Дата) : Проводка;  
    var q : TransQuery;  
    q = TransQuery.Create(Счета, Параметры, Дата1, Дата2);  
    return q.Current;  
end;
```

Описание

Первая;
First;

Назначение

Осуществляет переход к первой проводке, из числа найденных в результате запроса.

Пример

```
proc RunQuery(AccCond:String; ParmCond:String;  
  BegDate:Date; EndDate:Date);  
  var q : TransQuery;  
  q = TransQuery.Create(AccCond,  
    ParmCond, BegDate, EndDate);  
  -- цикл по проводкам, первый заход  
  while not q.Eof do  
    -- обработка  
    ...  
    q.Next;  
  end;  
  -- возвращаем указатель на начало  
  q.First;  
  -- цикл по проводкам, второй заход  
  while not q.Eof do  
    -- обработка  
    ...  
    q.Next;  
  end;  
end;
```

Описание

ПереместитьНа (Количество:Целое) ;
MoveBy (Number:Integer) ;

Аргументы

Количество - определяет, на сколько проводок и в какую сторону следует переместить указатель в перечне проводок, удовлетворяющих условиям запроса.

Назначение

Перемещает внутренний указатель запроса на заданное количество проводок по списку проводок, вошедших в результаты запроса. Аргумент может быть как положительным, так и отрицательным. В последнем случае переход осуществляется в сторону начала списка.

Если указанное количество превышает количество проводок, находящихся в списке от текущей позиции до его начала или конца (в зависимости от знака аргумента), текущим становится соответственно первая или последняя проводка.

Пример

```
-- перемещаемся через проводку вперед  
q.MoveBy(2) ;
```

Описание

Последняя;
Last;

Назначение

Осуществляет переход к последней проводке из числа найденных в результате запроса.

Пример

```
proc RunQuery(AccCond:String; ParmCond:String;  
  BegDate:Date; EndDate:Date);  
  var q : TransQuery;  
  q = TransQuery.Create(AccCond,  
    ParmCond, BegDate, EndDate);  
  -- делаем текущей последнюю проводку запроса  
  q.Last;  
  -- цикл по проводкам в обратном направлении  
  while not q.Bof do  
    -- обработка  
    ...  
    q.Previous;  
  end;  
end;
```

Описание

Предыдущая;
Previous;

Назначение

Осуществляет переход к предыдущей проводке из числа найденных в результате запроса.

Пример

```
proc RunQuery(AccCond:String; ParmCond:String;  
  BegDate:Date; EndDate:Date);  
  var q : TransQuery;  
  q = TransQuery.Create(AccCond,  
    ParmCond, BegDate, EndDate);  
  -- делаем текущей последнюю проводку запроса  
  q.Last;  
  -- цикл по проводкам в обратном направлении  
  while not q.Bof do  
    -- обработка  
    ...  
    q.Previous;  
  end;  
end;
```

Описание

Следующая;
Next;

Назначение

Осуществляет переход к следующей проводке из числа найденных в результате запроса.

Пример

```
proc RunQuery(AccCond:String; ParmCond:String;  
  BegDate:Date; EndDate:Date);  
  var q : TransQuery;  
  q = TransQuery.Create(AccCond,  
    ParmCond, BegDate, EndDate);  
  
  -- цикл по проводкам  
  while not q.Eof do  
    -- обработка  
    ...  
    q.Next;  
  end;  
end;
```


Описание

Создать(Счета:Строка [; Параметры:Строка] [; Дата1:Дата] [; Дата2:Дата]) : [ЗапросПроводок](#);
Create(Accounts:String [; Parameters:String] [; Date1:Date] [; Date2:Date]) : [TransQuery](#);

Аргументы

Счета - строка, содержащая условие отбора проводок [по счетам](#);

Параметры - строка с условием отбора проводок [по параметрам](#). Если данный аргумент опущен, считается, что никаких ограничений по параметрам нет;

Дата1 - начальная дата интересующего периода. Если опущен аргумент Дата1, началом периода отбора проводок будет считаться начало учета т.е. проводки запрашиваются, начиная с самой первой;

Дата2 - конечная дата интересующего отчетного периода. Причем, Дата1 входит, а Дата2 не входит в период. Если данный аргумент опущен, проводки отбираются до конца учетного периода, т.е. вплоть до самой последней проводки.

Назначение

Функция создает новый объект класса [ЗапросПроводок](#). В процессе создания объекта выполняется запрос проводок, так что проинициализированный объект предоставляет доступ к уже сформированному перечню проводок, удовлетворяющих заданным условиям.
















Проводки перечисляются в хронологическом порядке.

Пример

```
var q : TransQuery;  
q = TransQuery.Create("+62!", ":Валюта=USD", 01.01.2007, 01.03.2007);
```

Класс *ЗапросСчетов / AccQuery*, производный от родительского класса [Объект](#), предназначен для получения информации о счетах. С помощью данного класса формируется перечень всех счетов, отвечающий условиям, указанным при создании запроса.

Непосредственно в данном классе определены следующие свойства:

-  [Функция Создать / Create](#)
-  [Поле ПланСчетов / AccPlan](#)
-  [Поле МаскаСчетов / AccMask](#)
-  [Поле ТипыСчетов / AccTypes](#)
-  [Процедура Выбор / Select](#)
-  [Поле Количество / Count](#)
-  [Поле ВНачале / BOF](#)
-  [Поле ВКонце / EOF](#)
-  [Поле Текущий / Current](#)
-  [Процедура Первый / First](#)
-  [Процедура Последний / Last](#)
-  [Процедура Следующий / Next](#)
-  [Процедура Предыдущий / Previous](#)
-  [Процедура ПереместитьНа / MoveBy](#)
-  [Поле Счета / Acc / Items](#)

Описание

ВКонец : Логическое;
EOF : Logical;

Назначение

Возвращает значение TRUE после попытки переместить внутренний указатель запроса (доступный через свойство [Current](#)) за последний из счетов в результатах запроса, и FALSE в противном случае.

Если в запросе 0 элементов, то оба свойства и [BOF](#), и **EOF** равны TRUE. Данный способ позволяет проверить, не пуст ли результат запроса, причем этот способ является более предпочтительным, чем проверка свойства [Count](#), поскольку выполняется гораздо быстрее.

Поле доступно только на чтение.

Пример

См. также [Примеры использования запросов счетов](#).

Описание

ВНачале : Логическое;
BOF : Logical;

Назначение

Возвращает значение TRUE после попытки переместить внутренний указатель запроса (доступный через свойство [Current](#)) за первый счет в перечне счетов, сформированном по запросу, и FALSE в противном случае.

Если в запросе 0 элементов, то оба свойства и **BOF**, и **EOF** равны TRUE. Данный способ позволяет проверить, не пуст ли результат запроса, причем этот способ является более предпочтительным, чем проверка свойства [Count](#), поскольку выполняется гораздо быстрее.

Поле доступно только на чтение.

Пример

```
Пример
proc RunQuery(Accounts:String);
  var q : AccQuery;
  q = AccQuery.Create;
  q.AccMask = Accounts;
  q.Select;
  -- делаем текущим последний счет из запроса
  q.Last;
  -- цикл по счетам в обратном направлении
  while not q.Bof do
    -- обработка
    ...
    q.Previous;
  end;
end;
```

Описание

Количество : Целое;
Count : Integer;

Назначение

Возвращает количество счетов, попавших в результаты запроса. Поле доступно только на чтение.

Пример

```
proc RunQuery(Accounts:String);  
  var q : AccQuery;  
  q = AccQuery.Create  
  q.AccMask = Accounts;  
  q.Select;  
  if q.Count = 0 then  
    return;  
  end;  
  -- дальнейшая обработка  
  ...  
end;
```

Описание

МаскаСчетов :Строка;
AccMask :String;

Назначение

Позволяет узнать и изменить условие отбора счетов по их идентификатору и параметрам. В маске могут использоваться стандартные подстановочные символы '?', '*', '!'. Если маска не указана - в запросе анализируются все счета.

См. также [Синтаксис условий отбора проводок по счетам](#).

Пример

```
var q : AccQuery;  
  q = AccQuery.Create  
  q.AccPlan = "Balance";  
  q.AccMask = "62!";  
  q.Select;
```

Описание

ПланСчетов :Строка;
AccPlan :String;

Назначение

Позволяет узнать и изменить название плана счетов, из которого будут отбираться счета. Если план не указан - в запросе анализируются счета из всех планов.

Пример

```
var q : AccQuery;  
q = AccQuery.Create  
q.AccPlan = "Balance";  
q.AccMask = "62!";  
q.Select;
```

Описание

Счета[Индекс :Целое] : [Счет](#);
Accs [Index :Integer] : [Account](#);
Items[Индекс :Целое] : [Счет](#);

Аргументы

Индекс - номер счета из списка, построенного в результате выполнения запроса. Счета нумеруются от 1 и до общего числа счетов в результатах запроса (свойство [Количество](#)).

Назначение

Возвращает идентификатор конкретного счета по его номеру в списке. Поле доступно только на чтение, причем предварительно обязательно должна быть выполнена процедура [Выбор](#).

Данный способ доступа к результатам запроса более предпочтителен, чем использование стандартных процедур навигации по списку.

Пример

```
q = AccQuery.Create;  
q.Select;  
for i=1..q.Count do  
  trace(q.Items[i]);  
end;
```

См. также [Примеры использования запросов счетов](#).

Описание

Текущий : [Счет](#);
Current : [Account](#);

Назначение

Возвращает указатель на текущий счет из списка тех, что удовлетворяют условиям запроса.

Поле доступно только на чтение. Для перемещения по счетам запроса необходимо использовать методы [Первый](#), [Последний](#), [Следующий](#), [Предыдущий](#), [ПереместитьНа](#).

Пример

```
func ПервыйСчет(Маска:Строка) : Счет;  
    var q : AccQuery;  
    q = AccQuery.Create;  
    q.AccMask = Accounts;  
    q.Select;  
    return q.Current;  
end;
```

См. также [Примеры использования запросов счетов](#).

Описание

ТипыСчетов :Строка;
AccTypes :String;

Назначение

Позволяет узнать и изменить фильтр отбора счетов по их типам (определенным в структуре учета). Идентификаторы требуемых типов задаются через вертикальную черту '|'.

Пример

```
var q : AccQuery;  
q = AccQuery.Create  
q.AccTypes = "Товарный|Реализация";  
q.Select;
```

Описание

Выбор;
Select;

Назначение

Строит запрос по счетам. Множество анализируемых счетов может быть ограничено с помощью предварительной записи соответствующих значений в свойства [ПланСчетов](#), [МаскаСчетов](#) и [ТипыСчетов](#).

После построения запроса объект предоставляет доступ к сформированному перечню счетов, удовлетворяющих заданным условиям. Перебор счетов в перечне может производиться как с помощью традиционных для запросов процедур перемещения вперед/назад, так и через свойство [Счета](#).

Пример

```
var q : AccQuery;  
var a : Account;  
q = AccQuery.Create;  
q.Select; -- строим запрос  
-- далее анализируем результаты  
a = q.Items[1];  
q.Select; -- строим запрос
```

Описание

Первый;
First;

Назначение

Осуществляет переход к первому счету, из числа найденных в результате запроса.

Пример

```
proc RunQuery(Accounts:String);  
  var q : AccQuery;  
  q = AccQuery.Create;  
  q.AccMask = Accounts;  
  q.Select;  
  -- цикл по счетам, первый заход  
  while not q.Eof do  
    -- обработка  
    ...  
    q.Next;  
  end;  
  -- возвращаем указатель на начало  
  q.First;  
  -- цикл по счетам, второй заход  
  while not q.Eof do  
    -- обработка  
    ...  
    q.Next;  
  end;  
end;
```

Описание

ПереместитьНа (Количество:Целое) ;
MoveBy (Number:Integer) ;

Аргументы

Количество - определяет, на сколько счетов и в какую сторону следует переместить указатель в перечне счетов, построенном в результате выполнения запроса.

Назначение

Перемещает внутренний указатель запроса на заданное количество счетов по списку вошедших в результаты запроса счетов. Аргумент может быть как положительным, так и отрицательным. В последнем случае переход осуществляется в сторону начала списка.

Если указанное количество превышает количество счетов, находящихся в списке от текущей позиции до его начала или конца (в зависимости от знака аргумента), текущим становится соответственно первый или последний счет.

Пример

```
-- перемещаемся через счет назад  
q.MoveBy(-2) ;
```

Описание

Последний;
Last;

Назначение

Осуществляет переход к последнему счету из числа найденных в результате запроса.

Пример

```
proc RunQuery(Accounts:String);  
  var q : AccQuery;  
  q = AccQuery.Create;  
  q.AccMask = Accounts;  
  q.Select;  
  -- делаем текущим последний счет из запроса  
  q.Last;  
  -- цикл по счетам в обратном направлении  
  while not q.Bof do  
    -- обработка  
    ...  
    q.Previous;  
  end;  
end;
```

Описание

Предыдущий;
Previous;

Назначение

Осуществляет переход к предыдущему счету в списке найденных в результате запроса.

Пример

```
proc RunQuery(Accounts:String);  
  var q : AnalitQuery;  
  q = AccQuery.Create;  
  q.AccMask = Accounts;  
  q.Select;  
  -- делаем текущим последний счет из запроса  
  q.Last;  
  -- цикл по счетам в обратном направлении  
  while not q.Bof do  
    -- обработка  
    ...  
    q.Previous;  
  end;  
end;
```

Описание

Следующий ;
Next ;

Назначение

Осуществляет переход к следующему счету в списке найденных в результате запроса.

Пример

См. также [Примеры использования запросов счетов.](#)

Описание

Создать : [ЗапросСчетов](#);
Create : [AccQuery](#);

Назначение

Функция создает новый объект класса [ЗапросСчетов](#). Последующее использование объекта предполагает установку параметров запроса и его выполнение с помощью процедуры [Выбор](#). Непосредственно в данной процедуре запрос счетов не выполняется.

Пример

```
var q : AccQuery;  
q = AccQuery.Create;
```

Класс *ИсточникУчета / AccSource* обеспечивает программный доступ ко всем типам [журналов](#) (текстовым, табличным и картотечным) и к [аналитическим справочникам](#). Он используется в качестве базового класса для всех классов журналов и справочников, описанных в структуре учета.

Класс *ИсточникУчета* является общим предком для классов-наследников [Журнал](#) и [Справочник](#). Все классы доступны только на клиенте.

Непосредственно в данном классе определены следующие свойства:

- [Поле Name / Имя](#)
- [Поле Описание / Description](#)
- [Поле Записи / Records](#)
- [Поле ФормаКартотеки / Cardform](#)
- [Поле Виден / Visible](#)

Описание

Имя :Строка;
Name :String;

Назначение

Содержит обязательное имя [журнала](#) (идентификатор) или [аналитического справочника](#), которое задается в структуре учета при описании заголовка журнала или справочника.

Поле доступно только на чтение.

Пример

```
func ДобавитьЗакладкуЖурнала(аЖурнал :Journal; аЗаписиЖурнала :Record[]):Logical;  
var I, vCount :Integer;  
var локФрейм :TemplateFrame;  
var локФорма :CardForm;  
var локФильтр :String;  
var vQuery :Query;  
  
Result = False;  
if аЖурнал.Cardform <> nil then  
    локФильтр = "";  
    for I = 1..LengthOfArray(аЗаписиЖурнала) do  
        локФильтр = СложитьСтрокиФильтраПоИЛИ([локФильтр,  
        СложитьСтрокиФильтраПоИ([ 'ClassType='+аЗаписиЖурнала[I].ClassProject+'.'+  
        аЗаписиЖурнала[I].ClassName,  
        end;  
        локФорма = CreateJournal(аЖурнал.Name);  
        локФорма.CardFile.Filter =  
        СложитьСтрокиФильтраПоИ([локФорма.CardFile.Filter, локФильтр]);  
  
        vQuery = Query.Create(локФорма.Query.Documents);  
        vQuery.Filter = локФорма.CardFile.Filter;  
        vQuery.Select;  
        if not vQuery.EOF then  
            vCount = vQuery.Count;  
        end;  
        if vCount>0 then  
            Result = аЖурнал.Enabled;  
            локФрейм = ФреймСодержимое.AddFrame;  
            локФрейм.Name = аЖурнал.Name+аЖурнал.Description;  
            локФрейм.Caption = аЖурнал.Description+if(Result, "", " (Отключен)");  
            локФрейм.Bevel = Template.DefaultBevel;  
            LoadForm(локФрейм, локФорма);  
        end;  
    end;  
end;  
end;
```

Описание

Виден :Логическое;
Visible :Logical;

Назначение

Свойство предоставляет пользователям возможность скрывать неиспользуемые журналы или справочники. Если для конкретного журнала или справочника установить Visible = False, то он не будет показываться в диалогах выбора.

Для справочников, не привязанных к документу, это свойство по умолчанию равно False.

Свойство доступно на чтение и на запись.

Описание

Записи :Класс[] Запись;
Records :Class[] Record;

Назначение

Возвращает массив классов записей, являющихся источником данных для гетерогенного [картотечного журнала](#). Негетерогенный журнал так же, как и [аналитический справочник](#), может быть *связан только с одним источником*.

Поле доступно только на чтение.

Пример

```
proc ДобавитьЖурналы(парКлассЗаписи :class Record);  
  var I, Index :Integer;  
  var локЖурнал :Journal;  
  
  for I = 1..ОбластьУчета.JourCount do  
    локЖурнал = ОбластьУчета.Journal[I];  
    Index      = SearchInJournal(Журналы, локЖурнал);  
    if Index=-1 then  
      if SearchInArray(локЖурнал.Records, парКлассЗаписи)>0 then  
        Журналы[LengthOfArray(Журналы)+1] = локЖурнал;  
      end;  
    end;  
  end;  
end;
```

Описание

Описание :Строка;
Description :String;

Назначение

Содержит необязательное описание назначения [журнала](#) или [аналитического справочника](#), которое задается в структуре учета при описании заголовка журнала или справочника.

Поле доступно только на чтение и если назначение не задано, то поле возвращает пустую строку.

Пример

```
func ДобавитьЗакладкуЖурнала(аЖурнал :Journal; аЗаписиЖурнала :Record[]):Logical;  
var I, vCount :Integer;  
var локФрейм :TemplateFrame;  
var локФорма :CardForm;  
var локФильтр :String;  
var vQuery :Query;  
  
Result = False;  
if аЖурнал.Cardform <> nil then  
    локФильтр = "";  
    for I = 1..LengthOfArray(аЗаписиЖурнала) do  
        локФильтр = СложитьСтрокиФильтраПоИЛИ([локФильтр,  
        СложитьСтрокиФильтраПоИ([ 'ClassType='+аЗаписиЖурнала[I].ClassProject+'.'+  
        аЗаписиЖурнала[I].ClassName,  
        end;  
        локФорма = CreateJournal(аЖурнал.Name);  
        локФорма.CardFile.Filter =  
        СложитьСтрокиФильтраПоИ([локФорма.CardFile.Filter, локФильтр]);  
  
        vQuery = Query.Create(локФорма.Query.Documents);  
        vQuery.Filter = локФорма.CardFile.Filter;  
        vQuery.Select;  
        if not vQuery.EOF then  
            vCount = vQuery.Count;  
        end;  
        if vCount>0 then  
            Result = аЖурнал.Enabled;  
            локФрейм = ФреймСодержимое.AddFrame;  
            локФрейм.Name = аЖурнал.Name+аЖурнал.Description;  
            локФрейм.Caption = аЖурнал.Description+if(Result, "", " (Отключен)");  
            локФрейм.Bevel = Template.DefaultBevel;  
            LoadForm(локФрейм, локФорма);  
        end;  
    end;  
end;  
end;
```

Описание

ФормаКартотеки : Класс [ФормаКартотеки](#);
Cardform : Class [Cardform](#);

Назначение

Возвращает ссылку на класс [ФормаКартотеки](#). Поле доступно только на чтение.

Пример

```
func ДобавитьЗакладкуЖурнала(аЖурнал :Journal;  
    аЗаписиЖурнала :Record[]):Logical;  
var I, vCount :Integer;  
var локФрейм :TemplateFrame;  
var локФорма :CardForm;  
var локФильтр :String;  
var vQuery :Query;  
  
Result = False;  
if аЖурнал.Cardform <> nil then  
    локФильтр = "";  
    for I = 1..LengthOfArray(аЗаписиЖурнала) do  
        локФильтр = СложитьСтрокиФильтраПоИЛИ([локФильтр,  
            СложитьСтрокиФильтраПоИ([ 'ClassType='+аЗаписиЖурнала[I].ClassProject+'.'+  
                аЗаписиЖурнала[I].ClassName,  
            end;  
        локФорма = CreateJournal(аЖурнал.Name);  
        локФорма.CardFile.Filter =  
            СложитьСтрокиФильтраПоИ([локФорма.CardFile.Filter, локФильтр]);  
  
        vQuery = Query.Create(локФорма.Query.Documents);  
        vQuery.Filter = локФорма.CardFile.Filter;  
        vQuery.Select;  
        if not vQuery.EOF then  
            vCount = vQuery.Count;  
        end;  
        if vCount>0 then  
            Result = аЖурнал.Enabled;  
            локФрейм = ФреймСодержимое.AddFrame;  
            локФрейм.Name = аЖурнал.Name+аЖурнал.Description;  
            локФрейм.Caption = аЖурнал.Description+if(Result, "", " (Отключен)");  
            локФрейм.Bevel = Template.DefaultBevel;  
            LoadForm(локФрейм, локФорма);  
        end;  
    end;  
end;  
end;
```

Класс *ОбластьУчета / AccDomainJournal* обеспечивает программный доступ ко всем имеющимся [областям учета](#), описанным в структуре учета.

Класс *ОбластьУчета* является производным от класса [Объект](#), и наследует все имеющиеся у него свойства методы.

Непосредственно в данном классе определены следующие свойства:

- [Поле Имя / Name](#)
- [Поле КоличествоПлановСчетов / AccPlanCount](#)
- [ПланСчетов / AccPlan](#)
- [Поле КоличествоЖурналов / JourCount](#)
- [Поле Журнал / Journal](#)

Описание

Журнал[Индекс :Целое] : [Журнал](#);
Journal[Index :Integer] : [Journal](#);

Аргументы

Индекс - номер журнала;

Назначение

По заданному номеру журнала возвращает ссылку на объект класса Журнал. номер журнала изменяется в пределах от 1 до [количества журналов](#).

Пример

```
proc ДобавитьЖурналы(парКлассЗаписи :class Record);  
  var I, Index :Integer;  
  var локЖурнал :Journal;  
  
  for I = 1..ОбластьУчета.JourCount do  
    локЖурнал = ОбластьУчета.Journal[I];  
    Index      = SearchInJournal(Журналы, локЖурнал);  
    if Index=-1 then  
      if SearchInArray(локЖурнал.Records, парКлассЗаписи)>0 then  
        Журналы[LengthOfArray(Журналы)+1] = локЖурнал;  
      end;  
    end;  
  end;  
end;
```

Описание

Имя :Строка;
Name :String;

Назначение

Возвращает имя [области учёта](#), которое используется для идентификации областей учёта, описанных в структуре учета.

Пример

```
func ОбластьУчетаПоИмени(const парИмя :String) :AccDomain;  
  var I          :Integer;  
  var локОблУчета :AccDomain;  
  
  Result = nil;  
  for I = 1..AccStruct.AccDomainCount do  
    локОблУчета = AccStruct.AccDomain[I];  
    if локОблУчета.Name = парИмя then  
      Result = локОблУчета;  
      Break;  
    end;  
  end;  
end;  
end;
```

Описание

КоличествоЖурналов :Целое;
JourCount :Integer;

Назначение

Возвращает количество журналов, описанных в структуре учета и принадлежащих конкретной [области учета](#).

Пример

```
func КлассыЗаписейДляЖурналовДомена(const аДомен :AccDomain) :class[] Record;  
    var локНомерЖурнала, локНомерКласса :Integer;  
    var локЖурнал :Journal;  
    var локКлассыЗаписей :class[] Record;  
  
    Assert(аДомен <> nil);  
  
    for локНомерЖурнала = 1 .. аДомен.JourCount do  
        локЖурнал = аДомен.Journal[локНомерЖурнала];  
        локКлассыЗаписей = локЖурнал.Records;  
  
        for локНомерКласса = 1 .. LengthOfArray(локКлассыЗаписей) do  
            ДобавитьВМассив(Result, локКлассыЗаписей[локНомерКласса], True);  
        end;  
    end;  
end;
```

Описание

КоличествоПлановСчетов :Целое;
AccPlanCount :Integer;

Назначение

Возвращает количество [планов счетов](#), принадлежащих конкретной [области учета](#), описанной в структуре учета.

Пример

```
-- фрагмент исходного кода
with CardFile.ColumnByField['План'] do
  List.Clear;
  List.Add('|');
  for i = 1..AccStruct.AccDomainCount do
    if AccStruct.AccDomain[i].Name = 'Бухгалтерия' then
      for j = 1..AccStruct.AccDomain[i].AccPlanCount do
        List.Add(AccStruct.AccDomain[i].AccPlan[j].ClassInfo.Name + "|" +
          AccStruct.AccDomain[i].AccPlan[j].ClassInfo.Name);
      end;
    end;
  end;
end;
```

Описание

ПланСчетов[Номер :Целое] :Класс [ПланСчетов](#);
AccPlan[Number :Integer] :Class [AccPlan](#);

Аргументы

Номер - номер плана счетов, для которого требуется узнать имя.

Назначение






По номеру [плана счетов](#) возвращает ссылку на класс [ПланСчетов](#). Номер должен лежать в пределах от 1 до общего [количества планов счетов](#).

Поле доступно только на чтение.

Пример

```
-- фрагмент исходного кода
with CardFile.ColumnByField['План'] do
  List.Clear;
  List.Add('|');
  for i = 1..AccStruct.AccDomainCount do
    if AccStruct.AccDomain[i].Name = 'Бухгалтерия' then
      for j = 1..AccStruct.AccDomain[i].AccPlanCount do
        List.Add(AccStruct.AccDomain[i].AccPlan[j].ClassInfo.Name + "|" +
          AccStruct.AccDomain[i].AccPlan[j].ClassInfo.Name);
      end;
    end;
  end;
end;
```

Класс *Операции / Operation*, производный от класса [Объект](#), используется в качестве базового класса для всех классов типовых операций, описанных на языке типовых операций (ЯТО). Непосредственно в данном классе определены следующие свойства:

-  [Поле ЧислоОбъектов / ObjectsCount](#)
-  [Поле Объекты / Objects](#)
-  [Функция Создать / Create](#)
-  [Поле ТипКласса / ClassType](#)
-  [Процедура ПриУничтожении / OnDestroy](#)

Все вышеперечисленные методы и поля наследуются пользовательскими классами типовых операций.

Описание

```
Объекты[Индекс: Целое]: Операции;  
Objects[Index: Integer] : Operation;
```

Аргументы

Индекс - номер объекта, который может изменяться в пределах от 1 до [числа объектов](#).

Назначение

Данное свойство позволяет получить доступ к любому из существующих объектов соответствующего класса по номеру. Поле доступно только на чтение.

Описание

ТипКласса : Класс [Операции](#);
ClassType: Class [Operation](#);

Назначение

Возвращает указатель на класс [Операции](#) (именно на класс, а не на объект класса). Указатель позволяет создавать объекты на базе существующих (клонировать), проверять, принадлежат ли объекты одному классу или разным, а также определять наследственные связи в иерархии классов.

Поле доступно только на чтение.

Пример

```
func Fl(Init:Operation; Base:Operation):Logical;  
  var L : Logical;  
  if Base.ClassType = Init.ClassType then  
    Message(Init.ClassName+" = "+Base.ClassName);  
    L = TRUE;  
  else  
    Message(Init.ClassName+" <> "+Base.ClassName);  
    L = FALSE;  
  fi;  
  return L;  
end;
```


Описание

`ЧислоОбъектов` : Целое;
`ObjectsCount` : Integer;

Назначение

Поле содержит количество объектов соответствующего класса. Поле доступно только на чтение.

Описание

```
ПриУничтожении;  
OnDestroy;
```

Назначение

Данный метод автоматически вызывается ядром при уничтожении объекта. Перекрыв его в любом пользовательском классе, можно вставить туда требуемый код, который будет выполняться в момент уничтожения объекта данного класса.

Непосредственный вызов данного метода не уничтожает объект.

См. также функцию [Создать / Create](#).

Пример

```
InClass  
var OperationFinished : logical := false;  
InObject  
  
proc OnDestroy;  
  -- если удаляется последний объект,  
  -- выставляем флаг завершенности некоторой операции  
  if ObjectsCount = 1 then  
    OperationFinished = true;  
  end;  
end;
```

Описание

Создать : [Операции](#);

Create : [Operation](#);

Назначение

Функция создает новый объект класса операций. Основное назначение функции – обеспечить реализацию конструктора объектов в базовом классе **Операции**. В производных классах этот метод может перекрываться одноименной функцией для выполнения дополнительных действий при создании прикладных объектов.

См. также процедуру [ПриУничтожении / OnDestroy](#).

Пример

```
func Create : Operation;  
    Result = inherited Create;  
    -- здесь можно выполнить дополнительные действия  
end;
```

Класс *ПланСчетов / AccPlan* является базовым для всех классов, порождаемых системой на основе описаний планов счетов. Каждый план счетов из [структуры учета](#) после компиляции проекта транслируется в одноименный класс, производный от класса *ПланСчетов*.

Класс *ПланСчетов* наследует от родительского класса [Объект](#) все имеющиеся у него свойства и методы.

Непосредственно в классе *ПланСчетов* определены следующие свойства:

- [Поле Балансовый / Balanced](#)

Описание

Балансовый :Логический;
Balanced :Logical;

Назначение

Позволяет узнать, является ли план счетов балансовым (свойство равно true). Если план счетов является балансовым, то по счетам этого плана разрешается *формировать только проводки* и не допускаются полупроводки.

Пример

```
func ВзятьПланСчетовПоДомену(парИмяДомена :String) :String;
  var локОблУчета :AccDomain;
  var I :Integer;
  var локПлан :class AccPlan;

  локОблУчета = ОбластьУчетаПоИмени(парИмяДомена);
  if (локОблУчета <> nil) then
    for I = 1..локОблУчета.AccPlanCount do
      локПлан = локОблУчета.AccPlan[I];
      if локПлан.Balanced then
        Result = локПлан.ClassName;
        Break;
      elsif Result = '' then
        Result = локПлан.ClassName;
      end;
    end;
  end;
end;
```

Класс *ПодтаблицаПризнака / SignSubtable* является базовым для всех структур, описанных в справочниках структуры учета и наследует все свойства методы от родительского класса [Объект](#).

В данном классе вводятся следующие свойства:

- [Поле Количество / Count](#)
- [Поле Пункты / Items](#)
- [Поле ПунктыПоНомеру / ItemsByNumber](#)
- [Поле ИндексПоНомеру / IndexByNumber](#)

Описание

```
ИндексПоНомеру[Номер:Целое]: Вариант;  
IndexByNumber[Number:Integer]: Variant;
```

Аргументы

Номер - определяет, для какого элемент подтаблицы требуется узнать значение, использованное в качестве индекса.

Назначение

Позволяет прочесть индекс конкретного элемента подтаблицы по его порядковому номеру. Тип индекса зависит от типа подтаблицы. Если подтаблица описана в структуре учета как массив (Array) - индекс должен быть ординарного типа (Logical или Integer).

Если подтаблица описана как периодическое поле (Period) - индекс может быть любого типа, на котором определены отношения больше/меньше (String, Integer, Numeric, Logical, Date). Индексы всех элементов подтаблицы имеют один и тот же тип.

Индекс элемента в общем случае не равен его номеру. Поле доступно только на чтение.

Описание

Количество : Целое;

Count: Integer;

Назначение

Содержит количество значений (экземпляров) структур или полей, находящихся в подтаблице. Поле доступно только на чтение.

Описание

```
Пункты[Индекс:Вариант]: Вариант;  
Items[Index:Variant]: Variant;
```

Аргументы

Индекс - определяет, какой элемент подтаблицы требуется прочитать. Тип индекса определяется типом подтаблицы. Если подтаблица описана в структуре учета как массив (Array) - **Индекс** должен быть ординарного типа (Logical или Integer). Если подтаблица описана как периодическое поле (Period) - **Индекс** может быть любого типа, на котором определены отношения больше/меньше (String, Integer, Numeric, Logical, Date). Индексы всех элементов подтаблицы должны иметь один и тот же тип.

Назначение

Данное многозначное поле произвольного типа (зависит от описания конкретного справочника в структуре учета) позволяет прочитать элемент подтаблицы (это может быть структура или поле некоторого типа).

Описание

```
ПунктыПоНомеру[Номер:Целое]: Вариант;  
ItemsByNumber[Number:Integer]: Variant;
```

Аргументы

Номер - определяет, какой элемент подтаблицы требуется прочитать. Фактически это порядковый номер элемента.

Назначение

Позволяет прочитать конкретный элемент подтаблицы по его порядковому номеру. Порядковый номер в общем случае не равен индексу элемента, который используется при адресации с помощью свойства [Пункты/Items](#).

Класс **Полупроводка / HTrans**, производный от класса [Объект](#), предназначен для получения информации об элементарной (неделимой) операции по перемещению средств с/на счета. Экземпляры объектов данного класса создаются в результате выполнения запроса полупроводок с помощью класса [ЗапросПолупроводок](#) или с помощью вызова конструктора [Создать](#).

Непосредственно в данном классе определены следующие свойства и методы:

-  [Функция Создать / Create](#)
-  [Поле Дата / Date](#)
-  [Поле Запись / Record](#)
-  [Поле ЭтоДебет / IsDebet](#)
-  [Поле ЭтоКредит / IsCredit](#)
-  [Поле Счет / Acc](#)
-  [Поле КоррСчет / CorrAcc](#)
-  Процедура Очистить

Дополнительно для объекта полупроводки система динамически генерирует свойства, соответствующие тем параметрам, которые были описаны для типа счета полупроводки в структуре учета. Функция [ВзятьПоле / GetField](#) и процедура [УстановитьПоле / SetField](#) позволяют получить из полупроводки или записать в объект полупроводки значение параметра, определенного для счета данного типа в соответствии со [структурой учета](#).

Если переданная в качестве аргумента строка не соответствует ни одному параметру счета, генерируется ошибка. В связи с этим рекомендуется всегда проводить обработку исключений в тех блоках кода, где идет работа с полями полупроводки.

В результате описания любого типа счета в структуре учета в системе регистрируются два класса: не только класс этого счета (производный от класса [Счет](#)), но и связанный с этим классом пользовательский класс полупроводок, производный от класса *Полупроводка*. Такой "конкретный" класс полупроводок получает имя вида:

Полупроводка_<имя класса счета>

Описание

Дата : Дата;
Date : Date;

Назначение

Поле содержит дату полупроводки. Поле доступно только на чтение. Для редактирования полупроводок необходимо оперировать записями журналов.

Пример

```
var h: HTrans;  
...  
-- получили объект-полупроводку  
if h.Date > 01.01.2006 then  
    -- анализ  
end;
```

См. также [Пример запроса полупроводок](#).

Описание

Запись : [Запись](#);

Record : [Record](#);

Назначение

Поле содержит ссылку на документ, породивший данную полупроводку. Поле доступно только на чтение. К данному полю нельзя обращаться из кода типовых операций - в этом случае будет возникать ошибка времени исполнения.

Пример

```
var h: HTrans;  
var n: Накладная;  
...  
-- получили объект-полупроводку h  
if h.Record.ClassName = "Накладная" then  
  n = h.Record;  
  trace(n.НазваниеТовара);  
  -- анализ  
end;
```

Описание

КоррСчет : [Счет](#);
CorrAcc : [Account](#);

Назначение

Поле содержит идентификатор корреспондирующего счета, который был в полупроводке, парной текущей (две полупроводки образуют проводку). Если данная полупроводка была создана как одиночная (корреспонденции нет), данное поле содержит значение **nil**.

Поле доступно только на чтение. Для редактирования полупроводок необходимо оперировать записями журналов.

Пример

```
var ППроводка: HTrans;  
...  
-- получили объект-полупроводку  
if ППроводка.КоррСчет <> nil then  
    -- анализируем только полупроводки с корреспонденцией  
end;
```

Описание

Счет : [Счет](#);
Асс : [Account](#);

Назначение

Поле содержит идентификатор счета полупроводки. Поле доступно только на чтение. Для редактирования полупроводок необходимо оперировать записями журналов.

Пример

```
var h: HTrans;  
...  
-- получили объект-полупроводку  
if h.IsCredit AND h.Асс = 62 then  
  -- анализ 62 счета по кредиту  
end;
```

См. также [Пример запроса полупроводок](#).

Описание

ЭтоДебет : Логическое;
IsDebet : Logical;

Назначение

Поле содержит признак того, является ли счет полупроводки дебетуемым. Поле доступно только на чтение. Для редактирования полупроводок необходимо оперировать записями журналов.

Пример

```
var h: HTrans;  
...  
-- получили объект-полупроводку  
if h.IsDebet AND h.Acc = 40 then  
  -- анализ 40 счета по дебету  
end;
```

См. также [Пример запроса полупроводок](#).

Описание

ЭтоКредит : Логическое;
IsCredit : Logical;

Назначение

Поле содержит признак того, является ли счет полупроводки кредитуемым. Поле доступно только на чтение. Для редактирования полупроводок необходимо оперировать записями журналов.

Пример

```
var h: HTrans;  
...  
-- получили объект-полупроводку  
if h.IsCredit AND h.Акк = 62 then  
  -- анализ 62 счета по кредиту  
end;
```

См. также [Пример запроса полупроводок](#).

Описание

Создать : [Полупроводка](#);
Create : [HTrans](#);

Назначение

Функция создает новый объект класса **Полупроводка**. Наполнение объекта можно менять с помощью вызовов процедуры [УстановитьПоле](#).

Созданный объект всего лишь выступает в качестве временного хранилища информации о полупроводке (хранится в ОЗУ). Для того чтобы полупроводка попала в машину проводок, необходимо передать этот объект в качестве соответствующего параметра в процедуру [ПровестиПроводку](#) или [ПровестиПолупроводку](#) класса **Бухгалтерия**.

Пример






```
var h: HTrans;  
  h = Полупроводка_Базовый.Создать;  
  h.Сумма = 10^Валюта.Руб;  
  ПровестиПолупроводку(БалансовыйПлан.Счет01,h,TRUE);  
...
```



Класс аналитических справочников *Признак / Sign*, производный от класса [Объект](#), предназначен для работы с элементами аналитических справочников и является базовым для всех справочников аналитических признаков, определенных в проекте в файле [структуры учета](#). Иными словами, все справочники, описанные в структуре учета, регистрируются в иерархии как классы одноименные справочникам.

Каждый предопределенный аналитический признак любого справочника регистрируется в иерархии как статическое свойство соответствующего класса. Например, если в структуре учета описан справочник "спрВалюта" с обязательным признаком "руб", то в иерархии классов будет зарегистрирован класс "спрВалюта", унаследованный от класса "Признак" и имеющий свойство "руб" типа "спрВалюта". В связи с тем, что обязательная аналитика регистрируется как свойство класса, следует писать "спрВалюта.руб", а не просто "руб".

При компиляции структуры учета сделана проверка на совпадение имени поля справочника с именем свойства базового класса *Признак / Sign* и выдается предупреждение: "Поле ... перекрывает одноименное свойство класса ..." и программно данное поле будет недоступно.

В классе *Признак* определены следующие свойства и методы:

-  [Функция Открыть / Open](#)
-  [Функция ОткрытьПоИмени / OpenName](#)
-  [Функция ОткрытьПоЗаписи / OpenRecord](#)
-  [Поле Фильтр / Filter](#)
-  [Событие ПриОткрытииКартотеки / OnOpenCardfile](#)

-  [Поле Ключ / DocID](#)
-  [Поле Запись / Record](#)

Если справочник описан как справочник единиц измерения или валют, в классе дополнительно появляются свойства:

- **Базовый / Base** - ссылка на элемент справочника, являющийся основным;
- **Курс / Rate** - скалярное или периодическое поле числового типа, определяющее отношение данной единицей измерения к базовой;
- **Множить / Mult** - логический признак деления или умножения на курс;
- **Точность / Accur** - целое число, задающее точность величины измерителя.

Создать объект класса, производного от класса **Признак**, можно с помощью одной из этих функций. Кроме того, объекты создаются автоматически внутри системы в результате программного обращения к учетным данным, в частности, в результате запроса аналитики (класс [ЗапросАналитики](#)), запроса проводок или полупроводок (через свойства счетов проводок, попавших в результаты запроса класса [ЗапросПроводок](#), [ЗапросПолупроводок](#)).

Описание

Запись :Запись;
Record :Record;

Назначение

Возвращает объект класса **Запись (Документ)**, породивший данный признак. Поле доступно только на чтение.

Обращаться к данному полю можно только из пользовательских классов, бланков и картотек, но не из типовых операций. При попытке прочитать значение поля в типовой операции будет генерироваться исключительная ситуация.

Для выполнения обратной операции, то есть для поиска признака по записи, следует использовать метод [ОткрытьПоЗаписи/OpenRecord](#) класса **Признак**.

Пример

```
func ЕстьЛиДокумент(X: Признак): Документ;  
    var D : Документ;  
    D = X.Record;  
    if D = nil then  
        SetError(32111,"Признак не имеет документа-источника");  
    end;  
    return D;  
end;
```

Описание

Ключ ;
DocID ;

Назначение

Свойство возвращает уникальный ключ элемента справочника, по которому можно определить ссылку на элемент справочника, используя метод [Открыть / Open](#).

Свойство доступно только на чтение и *используется только в типовых операциях*.

Описание

Фильтр :Строка;

Filter :String;

Назначение

Данное свойство позволяет установить и прочитать текущее значение постоянного ограничения (фильтра), которое автоматически добавляется в запросы аналитики и в отчеты по справочникам.

Поле доступно на чтение и запись.

Описание

ПриОткрытииКартотеки :Строка;
OnOpenCardfile :String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события **OnOpenCardfile**,

Событие вызывается при попытке открыть справочник для выбора записи, например, в мастере условия отбора на параметры, в диалоге ввода проводки и т.п., и служит для того, чтобы *выбор можно было перекрыть прикладным кодом, открыв картотеку с дополнительным фильтром и настройками.*

Обработчик

```
func OnOpenCardfile(Parameter :String; var Record :Record; Filter :String; WindowStyle :  
Window.WindowStyles) :Integer;
```

Параметры:

Parameter - имя параметра счета, для которого вызывается справочник. В случае разыменованных параметров, например, в мастере условия отбора на параметры, в качестве параметра **Parameter** передается все разыменование;

Record - ссылка на запись картотеки, которая выделена в бланке-редакторе картотеки;

Filter - строковое выражение, определяющее условия отбора записей (документов), попадающих в картотеку.

WindowStyle - предопределенная [константа](#), которая определяет способ открытия картотеки.

Описание

ОткрытьПоЗаписи (Документ:Запись):Признак;
OpenRecord (Document:Record):Sign;

Аргументы

Документ - ссылка на документ любого класса, производного от класса **Запись** и использующегося в качестве источника аналитических признаков.

Назначение

По заданной ссылке на документ возвращает идентификатор аналитического признака, который определен на основе данного документа в соответствующей картотеке - источнике аналитики для справочника.

Обращаться к данной функции можно только из пользовательских классов, бланков и картотек, но не из типовых операций. При попытке выполнить функцию в типовой операции будет генерироваться исключительная ситуация.

Для выполнения обратной операции, то есть для поиска записи по признаку, следует использовать свойство [Запись/Record](#) объекта **Признак**.

Пример

```
функ ВнутреннийДокумент (Зап :Накладная): логическое;  
  перем Признак: Контрагент;  
  Признак = ОткрытьПоЗаписи (Зап);  
  if (Признак = Контрагент.Мы) then  
    return true;  
  endif;  
  return false;  
конец;
```


Описание

```
ОткрытьПоИмени (ИмяЭлемента :Строка): Признак;  
OpenName (ElementName :String): Sign;
```

Аргументы

ИмяЭлемента - идентификатор элемента указанного справочника.

Назначение

Возвращает объект класса **Признак**, содержащий ссылку на объект производного конкретного класса признаков, т.е. элемент аналитического справочника. Например, если в структуре учета был описан справочник "Валюта", то в иерархии классов после компиляции проекта появится класс **Валюта**, унаследованный от класса **Признак**, а для получения элемента справочника валют можно воспользоваться функцией **ОткрытьПоИмени**, предварительно каким-либо образом получив название валюты.

Если в функцию передается несуществующий идентификатор элемента, функция возвращает значение **nil (пусто)**.

Пример

```
var S : Sign;  
-- справочник "Валюта" должен быть  
-- описан в структуре учета  
S = Валюта. OpenName ("Руб");
```

Описание**Открыть** (Ключ :Вариант) : [Признак](#);**Open** (ID :Variant) : [Sign](#);**Аргументы****Ключ** - уникальный ключ элемента справочника.**Назначение**

Метод возвращает ссылку на элемент аналитического справочника по заданному уникальному ключу элемента справочника. Если элемент с таким ключом отсутствует в справочнике, то возвращает пустую ссылку - nil.

Функция *используется только в типовых операциях.*

Пример

```

опер Книги = "Проведение строки книги покупок/продаж"
  (аСумма20,
   аСумма18,
   аСумма10      :Numeric;
   аВалюта       :Базовый.спрЕдИзм      = nil;
   аСчетДебета,
   аСчетКредита   :Базовый.БазовыйСчет;
   аКонтрагент    :Базовый.спрКонтрагент = nil;
   аСчетФактура   :Integer              = 0;

   аНаше          :Базовый.спрНашиПредприятия;
   аДатаПроводки  :Date;
   аНомерСтроки   :Integer              = 0;
   аДопПарам      :ParamStorage         = nil;
   аДопПарам_Общ  :ParamStorage         = nil;
   аОснование     :String               = "");

var аСФ_Наша      :спрСФН;
var аСФ_Пост      :спрСФП;

if аСчетДебета<>nil and аСчетКредита<>nil then
  if аДопПарам=nil then аДопПарам = ParamStorage.Create; end;
  ОбъединитьДопПараметры(аДопПарам_Общ, аДопПарам);
  аДопПарам.Контрагент = аКонтрагент;
  if аСчетФактура<>0 then
    аСФ_Наша      = спрСФН.Open(аСчетФактура);
    if Str(аСФ_Наша)=аСФ_Наша.Name and Str(аСФ_Наша)=аСФ_Наша.Description then
      аСФ_Пост      = спрСФП.Open(аСчетФактура);
      if Str(аСФ_Пост)<>аСФ_Пост.Name or Str(аСФ_Пост)<>аСФ_Пост.Description then
        аДопПарам.СФП = аСФ_Пост;
      end;
    else
      аДопПарам.СФН   = аСФ_Наша;
    end;
  end;
  ...
end;
end;

```

Класс *Проводка / Trans*, являющийся наследником класса [Объект](#), предназначен для получения информации об операции снятия средств с одного счета и перечисления его на другой счет. Экземпляры объектов данного класса создаются в результате выполнения запроса проводок с помощью класса [ЗапросПроводок](#). Фактически одна проводка состоит из двух полупроводок.

Непосредственно в данном классе определены следующие свойства:

- [Поле Дата / Date](#)
- [Поле Дебет / Debet](#)
- [Поле Кредит / Credit](#)
- [Поле ОтобранаПоДебету / SelectedByDebet](#)
- [Поле ОтобранаПоКредиту / SelectedByCredit](#)
- [Поле Запись / Record](#)

Описание

Дата : Дата;
Date : Date;

Назначение

Поле содержит дату проводки. Поле доступно только на чтение. Для редактирования проводок необходимо оперировать записями журналов.

Пример

```
var t: Trans;  
...  
-- получили объект-проводку t  
if t.Date > 01.01.2007 then  
    -- анализ  
end;
```

Описание

Дебет : [Полупроводка](#);
Debet : [HTrans](#);

Назначение

Поле содержит ссылку на дебетовую [полупроводку](#), формирующую текущую проводку. Поле доступно только на чтение. Для изменения состава полупроводок необходимо редактировать журналы хозяйственных операций.

Пример

```
-- обработка дебетовых проводок за указанный период,  
-- отвечающих условиям отбора по счетам и параметрам  
proc ProcessDebet(BegDate:Date; EndDate:Date;  
    AccCond:String; ParamCond:String);  
var q : TransQuery;  
var t: Trans;  
var h: HTrans;  
  
-- запускаем запрос по проводкам  
q = TransQuery.Create(AccCond,  
    ParamCond, BegDate, EndDate);  
while not q.Eof do  
    t = q.Current;  
  
    if t.SelectedByDebet then  
        h = t.Debet;  
        trace(h.Acc); -- выводим счет дебета  
        -- обработка параметров счета дебета  
        ...  
    end;  
    q.Next;  
end;  
end;
```

Описание

Запись : [Запись](#);

Record : [Record](#);

Назначение

Поле содержит ссылку на документ, породивший данную проводку. Поле доступно только на чтение. К данному полю нельзя обращаться из кода типовых операций - в этом случае будет возникать ошибка времени исполнения.

Пример

```
var t: Trans;
var n: Накладная;
...
-- получили объект-проводку t
if t.Record.ClassName = "Накладная" then
  n = t.Record;
  trace(n.НазваниеТовара);
  -- анализ
end;
```

Описание

Кредит : [Полупроводка](#);
Credit : [HTrans](#);

Назначение

Поле содержит ссылку на кредитовую [полупроводку](#), формирующую текущую проводку. Поле доступно только на чтение. Для изменения состава полупроводок необходимо редактировать журналы хозяйственных операций.

Пример

```
-- обработка кредитовых проводок за указанный период,  
-- отвечающих условиям отбора по счетам и параметрам  
proc ProcessCredit(BegDate:Date; EndDate:Date;  
    AccCond:String; ParamCond:String);  
var q : TransQuery;  
var t: Trans;  
var h: HTrans;  
  
    -- запускаем запрос по проводкам  
    q = TransQuery.Create(AccCond,  
        ParamCond, BegDate, EndDate);  
    while not q.Eof do  
        t = q.Current;  
  
        if t.SelectedByCredit then  
            h = t.Credit;  
            trace(h.Acc); -- выводим счет кредита  
            -- обработка параметров счета кредита  
            ...  
        end;  
        q.Next;  
    end;  
end;
```

Описание

ОтобранаПоДебету : Логическое;
SelectedByDebet : Logical;

Назначение

Поле позволяет определить, была ли данная проводка отображена в результаты запроса проводок из-за ее соответствия дебетовой части условий на счета и их параметры.

Поле доступно только на чтение.

Пример

```
-- обработка дебетовых проводок за указанный период,  
-- отвечающих условиям отбора по счетам и параметрам  
proc ProcessDebet(BegDate:Date; EndDate:Date;  
    AccCond:String; ParamCond:String);  
var q : TransQuery;  
var t:  Trans;  
var h: HTrans;  
  
-- запускаем запрос по проводкам  
q = TransQuery.Create(AccCond,  
    ParamCond, BegDate, EndDate);  
while not q.Eof do  
    t = q.Current;  
  
    if t.SelectedByDebet then  
        h = t.Debet;  
        trace(h.Акк); -- выводим счет дебета  
        -- обработка параметров счета дебета  
        ...  
    end;  
    q.Next;  
end;  
end;
```


Описание

ОтобранаПоКредиту : Логическое;
SelectedByCredit : Logical;

Назначение

Поле позволяет определить, была ли данная проводка отобрана в результаты запроса проводок из-за ее соответствия кредитовой части условий на счета и их параметры.

Поле доступно только на чтение.

Пример

```
-- обработка кредитовых проводок за указанный период,  
-- отвечающих условиям отбора по счетам и параметрам  
proc ProcessCredit(BegDate:Date; EndDate:Date;  
    AccCond:String; ParamCond:String);  
var q : TransQuery;  
var t:  Trans;  
var h: HTrans;  
  
-- запускаем запрос по проводкам  
q = TransQuery.Create(AccCond,  
    ParamCond, BegDate, EndDate);  
while not q.Eof do  
    t = q.Current;  
  
    if t.SelectedByCredit then  
        h = t.Credit;  
        trace(h.Acc); -- выводим счет кредита  
        -- обработка параметров счета кредита  
        ...  
    end;  
    q.Next;  
end;  
end;
```

Класс *Справочник / Reference* обеспечивает программный доступ к [аналитическим справочникам](#), описанным в структуре учета. Класс *доступен только на клиенте*.

Классы *Справочник* и [Журнал](#) имеют общего предка - класс [ИсточникУчета](#) - и наследуют все имеющиеся у него свойства:

- [Поле Name / Имя](#)
- [Поле Описание / Description](#)
- [Поле Записи / Records](#)
- [Поле ФормаКартотеки / Cardform](#)
- [Поле Виден / Visible](#)

Класс *СтруктураПризнака* / *SignStruct* является базовым для всех структур, описанных в справочниках структуры учета и наследует все свойства и методы от родительского класса [Объект](#).

У классов, производных от класса *СтруктураПризнака*, в перечень свойств добавляются все поля, описанные у соответствующего аналитического справочника в структуре учета.

Класс *СтруктураУчета* / *AccStruct* является наследником класса [Объект](#) и обеспечивает программный доступ к спискам справочников и областей учета. Для этого в нем определены следующие поля:

- [Поле КоличествоСправочников / ReferenceCount](#)
- [Поле Справочник / Reference](#)
- [Поле КоличествоОбластейУчета / AccDomainCount](#)
- [Поле ОбластьУчета / AccDomain](#)

Описание

КоличествоОбластейУчета :Целое;
AccDomainCount :Integer;

Назначение

Возвращает количество [областей учета](#), описанных в структуре учета.

Пример

```
func ОбластьУчетаПоИмени(const парИмя :String) :AccDomain;
var I:Integer;
var локОблУчета :AccDomain;

Result = nil;
for I = 1..AccStruct.AccDomainCount do
    локОблУчета = AccStruct.AccDomain[I];
    if локОблУчета.Name = парИмя then
        Result = локОблУчета;
        Break;
    end;
end;
end;
```

Описание

КоличествоСправочников :Целое;

ReferenceCount :Integer;

Назначение

Возвращает количество [аналитических справочников](#), описанных в структуре учета.

Описание

ОбластьУчета[Индекс :Целое] : [ОбластьУчета](#);
AccDomain[Index :Integer] : [AccDomain](#);

Аргументы

Индекс - номер [области учета](#), описанной в структуре учета. Номер может изменяться в пределах от 1 до [количества областей учета](#), описанных в структуре учета.

Назначение

По заданному номеру области учета возвращает ссылку на объект класса [ОбластьУчета](#).

Пример

```
func ОбластьУчетаПоИмени(const парИмя :String) :AccDomain;  
  var I:Integer;  
  var локОблУчета :AccDomain;  
  
  Result = nil;  
  for I = 1..AccStruct.AccDomainCount do  
    локОблУчета = AccStruct.AccDomain[I];  
    if локОблУчета.Name = парИмя then  
      Result = локОблУчета;  
      Break;  
    end;  
  end;  
end;
```

Описание

Справочник[Индекс :Целое] : [Справочник](#);
Reference[Index :Integer] : [Reference](#);

Аргументы

Индекс - номер справочника, который может изменяться в пределах от 1 до [количества справочников](#), описанных в структуре учета.

Назначение

По заданному номеру справочника возвращает ссылку на объект класса [Справочник](#).

Класс *Счет / Account* является базовым для всех типов счетов, определенных в проекте в файле [структуры учета](#) и наследует все свойства родительского класса [Объект](#).

Типы счетов, определенные в структуре учета, показываются в иерархии классов после компиляции проекта. Состав свойств каждого типа счетов формируется на основе его описания и определяет описание другого класса, производного от класса [Полупроводка](#), который тесно связан с соответствующим типом счета. Например, если описан тип счета *Базовый* :

```
тип Базовый;  
  
    параметр Сумма :измеритель Валюта = nil;  
  
конец;
```

то в иерархии классов появится класс *Базовый*, производный от класса *Счет*, причем у этого класса (как у всех наследников класса *Счет*) имеется поле *КлассПолупроводки*, содержащее ссылку на другой автоматически регистрируемый в системе класс *Полупроводка_Базовый*. Иными словами, описание типа счета приводит к регистрации в иерархии классов сразу двух новых классов: одного, производного от класса *Счет*, и другого, производного от класса *Полупроводка*, причем в последнем случае классу дается имя вида:

```
Полупроводка_<имя класса счета>
```

Именно в классе полупроводки регистрируются все параметры из описания типа счета в структуре учета. Так, все полупроводки с участием счетов типа *Базовый* получают соответствующее свойство **Сумма**.

Непосредственно в классе *Счет* описаны следующие свойства:

- [Поле Идент / ID](#)
- [Поле Имя / Name](#)
- [Поле Описание / Description](#)
- [Поле ПланСчетов / AccPlan](#)
- [Поле СуперСчет / SuperAcc](#)
- [Поле КоличествоСубсчетов / SubaccCount](#)
- [Поле СубСчет / SubAcc](#)

Поле Идент / ID

Описание

Идент :Целое;
ID :Integer;

Назначение

Содержит уникальный номер счета. Поле доступно только на чтение.

Пример

```
proc AccID(Acc1 :Account);  
    trace(Acc1.ID);  
end;
```

Описание

Имя :Строка;
Name :String;

Назначение

Содержит идентификатор счета (он задается в структуре учета).
Поле доступно только на чтение.

Пример

```
-- процедура выводит идентификаторы и описания счетов,  
-- задействованных в имеющихся полупроводках  
proc AccQuery;  
  var q : HTransQuery;  
  var h:  HTrans;  
  
  q = HTransQuery.Create("");  
  while not q.Eof do  
    h = q.Current;  
    -- выводим информацию о каждой полупроводке  
    Trace('Date'+Str(h.Date)+' '+if(h.IsDebet,'+', '-')+  
      ' '+Str(h.Acc.Name)+ ' '+Str(h.Acc.Description));  
    q.Next;  
  end;  
end;
```

Описание

КоличествоСубсчетов :Целое;
SubaccCount :Integer;

Назначение

Возвращает количество субсчетов у текущего счета. Поле доступно только на чтение.

Пример

```
-- для заданного счета выводит все субсчета
proc ListSubAcc(Acc1 :Account);
var i :Integer;
  for i = 0..Acc1.SubaccCount-1 do
    trace(Acc1.SubAcc[i]);
  end;
end;
```

Описание

Описание :Строка;
Description :String;

Назначение

Содержит описание счета (оно задается в структуре учета и является необязательным, т.е. может отсутствовать).
Поле доступно только на чтение.

Пример

```
-- процедура выводит идентификаторы и описания счетов,  
-- задействованных в имеющихся полупроводках  
proc AccQuery;  
  var q : HTransQuery;  
  var h:  HTrans;  
  
  q = HTransQuery.Create("");  
  while not q.Eof do  
    h = q.Current;  
    -- выводим информацию о каждой полупроводке  
    Trace('Date'+Str(h.Date)+' '+if(h.IsDebet,'+', '-')+  
      ' '+Str(h.Acc.Name)+' '+Str(h.Acc.Description));  
    q.Next;  
  end;  
end;
```

Описание

```
ПланСчетов :Класс ПланСчетов;  
AccPlan :Class AccPlan;
```

Назначение

Свойство позволяет провести проверку на наличие класса [ПланСчетов](#) и в случае успеха возвращает на него ссылку, иначе nil.

Поле доступно только на чтение.

Описание

СубСчет[Индекс :Целое] : [Счет](#);
SubAcc[Index :Integer] : [Account](#);

Аргументы

Индекс - индекс (номер) субсчета, который может меняться от 0 до количества субсчетов минус один.

Назначение

По заданному по индексу возвращает объект класса [Счет](#), описывающий субсчет текущего счета.

Поле доступно только на чтение.

Пример

```
-- для заданного счета выводит все субсчета
proc ListSubAcc(Accl :Account);
var i :Integer;
  for i = 0..Accl.SubaccCount-1 do
    trace(Accl.SubAcc[i]);
  end;
end;
```

Описание

СуперСчет :Счет;
SuperAcc :Account;

Назначение

Поле содержит ссылку на суперсчет (счет верхнего уровня), если текущий объект является субсчетом. Для счетов верхнего уровня (не субсчетов) поле содержит **nil**.

Пример

```
-- для заданного субсчета функция возвращает суперсчет
-- самого верхнего уровня
func GetRootAcc(Accl :Account) :Account;
  result = Accl;
  while true do
    -- если суперсчета нет, выходим из цикла
    if result.SuperAcc = nil then
      break;
    end;
    result = result.SuperAcc;
  end;
end;
```


Все классы, производные от родительского класса [Объект](#), сгруппированы по своему назначению на группы.

В группу классов для работы с шаблонами входят следующие:

- [Шаблон / Template](#)
- [ФреймШаблона / TemplateFrame](#)
- [СтильШаблона / TemplateStyle](#)
- [СтрокаШаблона / TemplateRow](#)
- [СтолбецШаблона / TemplateColumn](#)
- [КлеткаШаблона / TemplateCell](#)
- [СекцияШаблона / TemplateSection](#)
- [ОбластьШаблона / TemplateRange](#)
- [РедакторПоля / FieldEdit](#)

А также группа классов для работы с объектами шаблона:









- [ОбъектШаблона / TemplateObject](#)
- [Кнопка / Button](#)
- [Надпись / Label](#)
- [Флаг / CheckBox](#)
- [Переключатель / RadioButton](#)
- [Редактор / Edit](#)
- [Список / ComboBox](#)
- [Рисунок / Picture](#)
- [Рамка / Bevel](#)
- [РядЗакладок / TabSet](#)
- [Проигрыватель / MediaPlayer](#)
- [График / Chart](#)
- [OLEКонтейнер / OLEContainer](#)
- [КартотекаШаблона / TemplateCardfile](#)
- [ДеревоКартотеки / CardTree](#)
- [ПодтаблицаШаблона / TemplateSubCardfile](#)











Класс *КлеткаШаблона / TemplateCell*, производный от родительского класса [Объект](#), используется для работы с клетками шаблонов Студии.

Внимание! Создать объект класса *КлеткаШаблона* напрямую нельзя. Клетки шаблонов создаются только внутри системы и используются как свойства других объектов (класса [СекцияШаблона](#)), причем такие свойства имеют атрибут "только чтение". Вместе с тем, программа на ТБ.Скрипт может получить доступ к объекту *КлеткаШаблона* и изменять свойства клетки.

Непосредственно в классе *КлеткаШаблона* определены следующие свойства:

-  [Поле Владелец / Owner](#)
-  [Поле Строка / Row](#)
-  [Поле Столбец / Column](#)
-  [Поле Кадр / Frame](#)
-  [Поле Содержимое / Contents](#)
-  [Поле Значение / Value](#)
-  [Поле Текст / Text](#)
-  [Поле Стил / Style](#)
-  [Поле Выравнивание / Alignment](#)
-  [Поле ВертВыравнивание / VertAlignment](#)
-  [Поле Свертка / Wrap](#)
-  [Поле Обязательное / Required](#)
-  [Поле ПечататьБордю / PrintBevel](#)
-  [Поле Многострочность / Multiline](#)
-  [Поле АвтоВыделение / AutoSelect](#)
-  [Поле Автоблокировка / Autolock](#)
-  [Поле Макросы / UseMacro](#)
-  [Поле МожноВыделять / CanSelect](#)
-  [Поле ТабСтоп / TabStop](#)
-  [Поле Бордю / Bevel](#)
-  [Поле Рамка / Border](#)
-  [Поле Отступ / Margins](#)
-  [Поле ПоворотТекста / TextRotation](#)
-  [Поле Цвет / Color](#)
-  [Поле Шрифт / Font](#)
-  [Поле ЦветПоля / FieldColor](#)
-  [Поле Подсказка / Hint](#)
-  [Поле КонтекстПомощи / HelpContext](#)
-  [Поле ТипПоля / FieldType](#)
-  [Поле ТипКлетки/CellType](#)
-  [Поле ФорматКлетки/CellFormat](#)
-  [Поле РазмерПоля / FieldSize](#)
-  [Поле ФорматПоля / FieldFormat](#)
-  [Поле СтилСтатическойКлетки / StaticStyle](#)
-  [Поле СтилЛогическогоПоля / LogicalStyle](#)
-  [Поле ФорматПоляДаты/DateFormat](#)
-  [Поле Надпись / Caption](#)
-  [Поле АвтоОбзор / AutoLookup](#)
-  [Поле ПолеОбзора / LookupField](#)
-  [Поле ФильтрОбзора / LookupFilter](#)
-  [Поле ИмяКартотеки / CardfileName](#)
-  [Поле ФорматМаскиПоиска / FindMaskFormat](#)
-  [Поле ДляПросмотра / ReadOnly](#)
-  [Поле Разрешен / Enabled](#)

-  [Поле Кнопка / Button](#)
-  [Поле Список / List](#)
-  [Поле ЯвляетсяМастером / IsMaster](#)
-  [Поле КлеткаМастер / MasterCell](#)
-  [Поле СвязаноПоШирине / LinkedWidth](#)
-  [Поле СвязаноПоВысоте / LinkedHeight](#)
-  [Функция ШиринаТекста / TextWidth](#)
-  [Процедура СФокусировать / SetFocus](#)

-  [Событие ПриНажатии / OnClick](#)
-  [Событие ПриПодсказке / OnHint](#)
-  [Событие ПриРисовании / OnDraw](#)
-  [Событие ПриВходе / OnEnter](#)
-  [Событие ПриПроверке / OnVerify](#)
-  [Событие ПриВыходе / OnExit](#)
-  [Событие ПриВыводе / OnOutput](#)
-  [Событие ПриВводе / OnInput](#)
-  [Событие ПриНаборе / OnType](#)
-  [Событие ПриОбзоре / OnLookup](#)

Описание

Автоблокировка :Логическое;
Autolock :Logical;

Назначение

Свойство **Автоблокировка** логического типа имеет смысл только для полей бланка-редактора.

По умолчанию Autolock = True и документ при входе в поле автоматически переводится в режим редактирования. Если Autolock = False, то документ будет переведен в режим редактирования при изменении значения при завершении редактирования поля.

Если поле вычисляемое и Autolock = False, то в случае необходимости перевод в режим редактирования осуществляется только программно.

Пример

```
Section1.Cell[2,1].Autolock = false;
```

Описание

АвтоВыделение :Логическое;
AutoSelect :Logical;

Назначение

Данное свойство позволяет выделять текст (свойство равно Истина) при входе в клетку шаблона, т.е. при открытии встроенного in-place редактора.

Если свойство равно Ложь, то текст при входе в текущую клетку выделен не будет, что позволяет избежать случайного удаления выделенного текста.

Пример

```
Section1.Cell[2,1].AutoSelect = false;
```

Описание

АвтоОбзор : Логическое;

AutoLookup : Logical;

Назначение

Данное свойство имеет смысл только для клеток, имеющих [тип поля](#): *СсылочноеПоле (ReferenceField)*.

Если в клетке включен автоматический обзор (**АвтоОбзор** = Истина), то попытка ввести в клетку какой-либо текст будет приводить к появлению [картотеки](#) для выбора требуемой записи.

Если автоматический обзор отключен, то в клетку можно ввести текст, который интерпретируется системой как значение [поля обзора](#) и непосредственно обзор данной картотеки по полю производится затем по нажатию клавиши *PgUp*.

Пример

```
Section1.Cell[2,1]. АвтоОбзор = false;
```

Описание

Бордюор : [Шаблон.СтилиБордюора](#) ;
Bevel : [Template.BevelStyles](#) ;

Назначение

Свойство позволяет определить или установить вокруг рабочей области клетки (поля) один из доступных стилей бордюора, который определен в перечислимом типе [СтилиБордюора/BevelStyles](#).

При выборе значения *NoneBevel* бордюор вокруг рабочей области (поля) клетки отсутствует, в остальных случаях бордюор прорисовывается одним из следующих стилей:

- по умолчанию | DefaultBevel
- тонкий | SingleBevel
- статический | StaticBevel
- внутренний | ClientBevel
- оконный | WindowBevel

Если [Отступ](#) равен нулю, бордюор рисуется вдоль внешней границы клетки, если же **Отступ** больше нуля, то рабочая область клетки (поле) занимает лишь часть клетки, окруженную пустым пространством, и бордюор выводится внутри клетки по периметру рабочей области.

Поле доступно на чтение и запись.

Пример

```
СекцияНакладной.Клетка[3,  
СекцияНакладной.КоличествоСтолбцов].Бордюор =  
Шаблон.SingleBevel;
```

Описание

ВертВыравнивание : Шаблон.ТипыВыравнивания;
VertAlignment : Template.AlignmentTypes;

Назначение

Позволяет узнать и изменить режим выравнивания текста в клетке по вертикали при выводе на экран или печать. Поле может принимать одно из следующих константных значений, определенных в типе [Шаблон.ТипыВыравнивания](#):

ВыравниватьВверх (ВыравниватьВлево) - выравнивание по верхнему краю
ВыравниватьВниз (ВыравниватьВправо) - выравнивание по нижнему краю
ВыравниватьПоЦентру - центрирование

Пример

СекцияПодпись.ВертВыравнивание = Ядро.Шаблон.ВыравниватьВниз;

Описание

Владелец : СекцияШаблона;
Owner : TemplateSection;

Назначение

Содержит ссылку на секцию, которой принадлежит клетка. Поле доступно только на чтение.

Пример

```
-- делаем указанную клетку и соседнюю белыми
proc WhiteSection(CC: TemplateCell);
var x,y: Integer;
var SS: TemplateSection;
  SS = CC.Owner;
  x = CC.Column;
  y = CC.Row;
  SS.Cell[x,y] = clWhite; -- эквивалентно CC.Color = clWhite;
  SS.Cell[x+1,y] = clWhite;
end;
```

Описание

Выравнивание : Шаблон.ТипыВыравнивания;
Alignment : Template.AlignmentTypes;

Назначение

Позволяет узнать и изменить режим выравнивания текста в клетке при выводе на экран или печать. Поле может принимать одно из следующих константных значений, определенных в типе [Шаблон.ТипыВыравнивания](#):

ВыравниватьВлево - выравнивание по левому краю
ВыравниватьВправо - выравнивание по правому краю
ВыравниватьПоЦентру - центрирование

Пример

```
for j=1..СекцияНакладной.КоличествоСтолбцов do
  -- первый столбец выравниваем по левому краю
  СекцияНакладной.Клетка[1,j].Выравнивание = Ядро.Шаблон.ВыравниватьВлево;
  -- второй столбец выравниваем по правому краю
  СекцияНакладной.Клетка[1,j].Выравнивание = Ядро.Шаблон.ВыравниватьВправо;
end;
```

Описание

ДляПросмотра : Логическое;
ReadOnly : Logical;

Назначение

Позволяет узнать и изменить свойство поля ввода, определяющее его доступность для редактирования. Когда свойство **ДляПросмотра** равно ИСТИНА, пользователь не может изменять значение в клетке. Данное поле имеет смысл для клеток, в которых [тип поля](#) равен любому типу кроме статического текста, и используется в случаях, когда содержимое клетки должно быть всегда недоступно пользователю и его предполагается менять только программно. Для временного запрещения поля следует использовать свойство [Разрешен](#).

Пример

```
-- проход по всем клеткам секции SS
for i=1..SS.ColumnsCount do
  for j=1..SS.RowsCount do
    -- если клетка используется как поле ввода/вывода
    if SS.Cell[i,j].FieldType = 1 then
      -- запрещаем ее редактирование
      SS.Cell[i,j].ReadOnly = true;
    end;
  end;
end;
```

Описание

Значение : Вариант;
Value : Variant;

Назначение

Поле предназначено для чтения и записи значения переменной, связанной с клеткой посредством свойства [Содержимое](#). Данное поле-свойство доступно только для клеток, представляющих собой поле ввода/вывода.

Пример

```
SS : Section;  
ДатаПриемаОплаты: Date;  
  
proc P1;  
  -- присваиваем клетке имя переменной  
  SS.Cell[3,1].Contents = "ДатаПриемаОплаты";  
  -- задаем отображение в полном формате  
  SS.Cell[3,1].FieldFormat = "dd mmmm yyyy";  
  -- устанавливаем хранимое в клетке значение  
  -- при этом автоматически изменяется значение переменной  
  SS.Cell[3,1].value = 10.06.2006;  
  -- теперь ДатаПриемаОплаты = 10.06.2006;  
  -- в клетке отображается преобразованная в соответствии с  
  -- форматом строка  
  message(SS.Cell[3,1].Text); -- выводит "10 июня 2006"  
end;
```

Описание

ИмяКартотеки : Строка;
CardfileName : String;

Назначение

Данное свойство имеет смысл только для клеток, имеющих [тип поля: СсылочноеПоле \(ReferenceField\)](#).

Свойство позволяет узнать и изменить имя картотеки, используемой для заполнения клетки (выбора конкретной записи).

Свойство может быть изменено не только программно, но и на стадии проектирования - в диалоге [свойств клетки](#).

См. также [ПолеОбзора](#).

Пример

```
-- при открытии бланка
proc BlankOnOpen(Create :Logical);
  -- назначаем для ссылочного поля выбора контрагента
  -- фильтр, отбирающий только наших покупателей
  Header.Cell[2,1].ИмяКартотеки = "Дельцы";
  Header.Cell[2,1].ПолеОбзора = "Имя";
  Header.Cell[2,1].ФильтрОбзора = "МыПродавец = True";
end;
```

Описание

Кадр : Целое;
Frame : Integer;

Назначение

Содержит индекс кадра, в котором находится данная клетка. Значение лежит в диапазоне от 1 до числа кадров в повторяющейся секции. Если секция неповторяющаяся, поле всегда содержит значение 1. Поле доступно только на чтение.

Пример

```
SS : Section;  
  
proc OnButtonRed(O:String);  
var i: integer;  
var j: integer;  
  -- делаем все клетки первого кадра в секции красными  
  for i=1..SS.ColumnsCount do  
    for j=1..SS.RowsCount do  
      if SS.Cell[i,j].Frame = 1 then  
        SS.Cell[i,1].Color = clRed;  
      fi;  
    end;  
  end;  
end;
```

Описание

КлеткаМастер :КлеткаШаблона;
MasterCell :TemplateCell;

Назначение

Поле позволяет получить ссылку на мастер-клетку в той группе объединенных клеток, к которой относится текущая клетка. Мастером-клеткой является самая верхняя, левая клетка в группе объединенных клеток.

Для не объединенных клеток (одна клетка) получаем ссылку на саму клетку.

Поле доступно только на чтение.

Описание

Кнопка : Логическое;
Button : Logical;

Назначение

Позволяет узнать и изменить свойство клетки, определяющее наличие в ней кнопки обзора (с многоточием) у правого края.

Когда свойство **Кнопка** равно ИСТИНА, клетка имеет кнопку обзора.

Пример

```
-- проход по всем столбцам секции SS
for i=1..SS.ColumnsCount do
  -- если столбец выводится на печать
  if SS.Column[i].Printed then
    -- цикл по всем клеткам столбца
    for j=1..SS.count do
      -- скрываем кнопку в клетках столбца
      SS.Cell[i,j].Button = false;
    end;
  end;
end;
end;
```


Описание

```
КонтекстПомощи : Строка;  
HelpContext : String;
```

Назначение

Свойство позволяет определить и задать путь к файлу с текстом помощи, поясняющим назначение текущей клетки шаблона.

Этот текст будет появляться в окне справке по прикладным системам, если клетка выделена и нажата клавиша **F1**. Для указанной клетки должен быть установлен флаг **Можно выделять** в диалоге "[Свойства клеток](#)".

Если поле пустое, то справка к данному объекту шаблона берется [для фрейма](#), на котором расположена секция с данной клеткой.

Пример

```
Section1.Cell[2,3].HelpContext= "C:\ПолныйПуть\ИмяФайла.htm";  
-- ПолныйПуть - полный путь к файлу ИмяФайла.htm с перечислением  
-- всех папок, начиная от папки, в которую установлена программа
```

Описание

Макросы : Логическое;
UseMacro : Logical;

Назначение

Поле определяет, разрешено ли в клетках шаблона использовать [макросы](#). Если значение свойства равно True, то любая последовательность символов, отвечающая синтаксису записи макросов, будет интерпретироваться системой как макрос (т.е. изменять способ отображения сопутствующего текста).

Если значение свойства равно False, весь текст выводится непосредственно в том виде, в каком он представлен в клетке.

Пример

```
SS : Section;  
-- ...  
SS.Cell[2,3].UseMacro = False;
```

Описание

Многострочность : Логическое;
Multiline : Logical;

Назначение

Свойство позволяет представить текст в клетке в виде нескольких строк, т.е. разрешается переносить текст на следующую строку нажатием клавиши *Enter* (перевод каретки).

Поле доступно на чтение и запись. Если свойство равно False, то переход на другую строчку невозможен.

Пример

```
Section1.Cell[3,4].Multiline = True;
```

Описание

МожноВыделять :Логическое;
CanSelect :Logical;

Назначение

Данное свойство определяет, может ли данная клетка быть выделена. Если свойство равно True, то на клетке может быть установлен курсор (т.е. она может быть выделена), а также доступны команды контекстного меню.

Поле доступно на чтение и запись.

Пример

```
Section1.Cell[2,1].CanSelect = false;
```

Описание

Надпись : Строка;
Caption : String;

Назначение

Данное поле позволяет узнать и изменить текст надписи, поясняющий назначение клетки. Поле доступно на чтение и запись.

Пример

```
Section1.Cell[3,1].Надпись = "Налоги включены";
```

Описание

Обязательное :Логическое;

Required :Logical;

Назначение

Свойство предназначено для полей, обязательных для ввода данных. Если значение поля равно True и поле не заполнено, то клетка выделяется специальным цветом.

Поле доступно на чтение и запись.

Описание

Отступ :Число;
Margins :Numeric;

Назначение

Позволяет узнать и изменить размер отступов между внешней границей клетки и внутренней рабочей областью клетки, где выводятся/вводятся данные и текст. Отступы задаются равными для всех 4-х сторон клетки. Если **Отступ** больше нуля, то рабочая область клетки (поле) занимает лишь часть клетки, окруженную пустым пространством. По умолчанию отступы равны нулю, и рабочее поле занимает всю клетку.

Значения поля задаются в миллиметрах и могут лежать в диапазоне от 0 до 1000.

Пример

```
-- делаем отступы по 10 мм  
Секция1.Клетка[1,1].Margins = 10;
```

Описание

```
ПечататьБордю :Логическое;  
PrintBevel :Logical;
```

Назначение

Свойство определяет, требуется ли или нет выводить бордю на печать, т.е. печатать обрамляющую полосу по периметру рабочей области клетки. Если свойство равно False, бордю на печать не выводится.

Поле доступно на чтение и запись.

Пример

```
-- бордю печатается  
Секция1.Клетка[1, 1].ПечататьБордю = true;
```


Описание

`ПоворотТекста` :Число;
`TextRotation` :Numeric;

Назначение

Позволяет считать или изменить угол поворота текста внутри клетки шаблона. Угол поворота текста можно также изменить с помощью поля **Поворот текста** на странице ["Поле"](#) диалога ["Свойства клетки"](#).

Примечание. Допускаются следующие значения углов поворота текста -180.0, -90.0, 0.0, 90.0, 180.0. Если провести условную горизонталь через середину клетки шаблона и указать положительное направление вправо, то угол поворота против часовой стрелки считается положительным.

Пример

```
-- выравнивание текста по вертикали  
Секция1.Клетка[1, 1].ПоворотТекста = 90;
```

Описание

```
Подсказка : Строка;  
Hint : String;
```

Назначение

Позволяет узнать и изменить строку контекстной подсказки, которая выводится во всплывающем окне, если пользователь задержал курсор мыши над клеткой.

Пример

```
Секция1.Клетка[1, Секция1.RowsCount].Hint = "Общее количество";
```

Описание

ПолеОбзора : Строка;
LookupField : String;

Назначение

Данное свойство имеет смысл только для клеток, имеющих [тип поля](#): *СсылочноеПоле (ReferenceField)*.

Свойство позволяет узнать и изменить имя поля для обзора, то есть имя поля из класса записи, для ввода экземпляров которого предназначена текущая клетка. Поле обзора позволяет осуществить быстрый ввод значений в клетку (они интерпретируются как значения указанного поля обзора и позволяют системе "на лету" искать записи с требуемым значением) и фильтрацию картотеки (в том случае, если несколько записей картотеки, используемой для заполнения клетки, содержат в поле обзора фрагмент строки, введенной пользователем).

Свойство может быть изменено не только программно, но и на стадии проектирования - в диалоге [свойств клетки](#).

См. также [ИмяКартотеки](#).

Пример

```
-- при открытии бланка
proc BlankOnOpen(Create :Logical);
  -- назначаем для ссылочного поля выбора контрагента
  -- фильтр, отбирающий только наших покупателей
  Header.Cell[2,1].ИмяКартотеки = "Дельцы";
  Header.Cell[2,1].ПолеОбзора = "Имя";
  Header.Cell[2,1].ФильтрОбзора = "МыПродавец = True";
end;
```

Описание

РазмерПоля : Целое;
FieldSize : Integer;

Назначение

С помощью этого свойства устанавливается максимальное число символов, которые можно ввести с клавиатуры при редактировании данной клетки. Формально, данное свойство можно установить у любой клетки, но как ограничитель введенных символов данное свойство работает только у тех клеток, которые редактируют строковые значения.

Если клетка в качестве поля содержит "поле записи", то максимальное количество введенных символов будет взято из MTL-описания данного поля при условии, что **FieldSize** = 0, иначе ограничителем ввода символов будет значение данного свойства.

Пример

```
var SS: TemplateSection;  
...  
SS.Cell[1,4].FieldSize = 30;
```

Описание

Разрешен :Логическое;
Enabled :Logical;

Назначение

Позволяет узнать и изменить свойство поля ввода, определяющее его общую доступность. Когда свойство **Разрешен** равно ЛОЖЬ, пользователь не может изменять значение в клетке, а кроме того становятся недоступными кнопка (обзора, истории, переключатель) или флаг, который размещен в клетке (кнопка не нажимается, в отличие от клетки доступной только [ДляПросмотра](#)).

Запрещенные элементы управления, включая поля ввода, отображаются цветом фона клетки и не генерируют никаких событий. Однако сама клетка, которая может выглядывать из-за поля (кнопки, флага) за счет широкого бордюра, по-прежнему способна генерировать события, в частности ПриНажатии.

Свойство **Разрешен** имеет смысл только для клеток, в которых [тип поля](#) равен любому типу кроме статического текста. Свойство доступно только во время исполнения проекта (к нему нет доступа во время проектирования шаблона в визуальном редакторе) и используется для временного запрещения некоторых полей в зависимости от логики работы шаблона.

Пример

```
-- пока в накладной не проставлен номер,  
-- документ нельзя сохранять и потому  
-- кнопка ОК остается недоступной  
  
-- при открытии шаблона запрещаем нажатие кнопки ОК  
-- и сохранение незаполненного документа  
proc BlankOnOpen(Create :Logical);  
    ButtonOK.Enabled = false;  
end;  
  
-- как только введен номер, разрешаем кнопку ОК  
func FieldNumberOnInput(Cell :TemplateCell; Value :Variant) :Logical;  
    ButtonOK.Enabled = true;  
    return true;  
end;
```

Описание

```
Рамка : Целое[];  
Border : Integer[];
```

Назначение

Позволяет узнать и изменить режим отрисовки рамки (тонкой или толстой линии) вокруг клетки. Свойство возвращает и принимает массив размером 4 элемента, каждый из которых отвечает за рамку с одной из 4-х сторон:

- 1 - слева
- 2 - сверху
- 3 - справа
- 4 - снизу

Каждый элемент массива может иметь следующие значения:

- 0 - нет рамки
- 1 - тонкая рамка
- 2 - толстая рамка

Пример

```
-- рисуем рамку тонкой линией слева-сверху  
-- и толстой линией справа-снизу  
Секция1.Клетка[1, Секция1.КоличествоСтолбцов].Рамка = [1,1,2,2];
```

Поле Свертка / Wrap

Описание

Свертка : Логическое;
Wrap : Logical;

Назначение

Позволяет узнать и изменить режим отрисовки длинного текста в пределах клетки. Если **Свертка** равна TRUE, то символы, не уместившиеся на одной строке, будут перенесены на следующую и так далее. (При этом высота кадра, в который входит клетка, автоматически подстраивается под наиболее высокую клетку кадра.) Если **Свертка** равна FALSE, текст выводится в одну строку и обрезается правой границей клетки.

Пример

```
Секция1.Клетка[1, 1].Свертка = ИСТИНА;
```

Описание

СвязаноПоВысоте :Целое;
LinkedHeight :Integer;

Назначение

Свойство позволяет получить количество клеток, размещенных по высоте в одной группе объединенных клеток. Причем, для всех клеток, принадлежащих одной объединенной группе свойство, возвращает одно и то же значение, как для обычной клетки, так и для мастера-клетки. Мастером-клеткой является самая верхняя, левая клетка в группе объединенных клеток.

Поле доступно только на чтение.

Для одиночной клетки, не объединенной в группу, свойство всегда равно 1.

Пример

```
var i,j,n: Integer;  
var SS: TemplateSection;  
...  
for i=1..SS.ColumnsCount do  
  for j=1..SS.RowsCount do  
    n = SS.Cell[i,j].LinkedHeight;  
    ...  
  end;  
end;  
end;
```


Описание

`СвязаноПоШирине` :Целое;
`LinkedWidth` :Integer;

Назначение

Свойство позволяет получить количество клеток, размещенных по ширине в одной группе объединенных клеток. Причем, для всех клеток, принадлежащих одной объединенной группе свойство, возвращает одно и то же значение, как для обычной клетки, так и для мастера-клетки. Мастером-клеткой является самая верхняя, левая клетка в группе объединенных клеток.

Поле доступно только на чтение.

Для одиночной клетки, не объединенной в группу, свойство всегда равно 1.

Пример

```
var i,j,n: Integer;  
var SS: TemplateSection;  
...  
for i=1..SS.ColumnsCount do  
  for j=1..SS.RowsCount do  
    n = SS.Cell[i,j].LinkedWidth;  
    ...  
  end;  
end;  
end;
```

Описание

Содержимое : Строка;
Contents : String;

Назначение

Поле предназначено для чтения и записи содержимого клетки. Если клетка имеет тип "поле ввода/вывода", то здесь указывается имя переменной произвольного типа (строка, дата, число, целое, ссылка). Непосредственно значение переменной связано со свойством [Значение \(Value\)](#). Если клетка предназначена для вывода статического текста, то в данном свойстве указывается отображаемая строка.

Пример

```
SS : Section;  
ДатаПриемаОплаты: Date;  
  
proc P1;  
  -- присваиваем клетке имя переменной  
  SS.Cell[3,1].Contents = "ДатаПриемаОплаты";  
  -- задаем отображение в полном формате  
  SS.Cell[3,1].FieldFormat = "dd mmmm yyyy";  
  -- устанавливаем хранимое в клетке значение  
  -- при этом автоматически изменяется значение переменной  
  SS.Cell[3,1].value = 10.06.2006;  
  -- теперь ДатаПриемаОплаты = 10.06.2006;  
  -- в клетке отображается преобразованная в соответствии с  
  -- форматом строка  
  message(SS.Cell[3,1].Text); -- выводит "10 июня 2006"  
end;
```

Описание

Список : [СписокСтрок](#);
List : [StringList](#);

Назначение

Данное свойство предназначено для программного управления перечнем строк в поле перечислимого типа, то есть поля с выпадающим списком. Свойство содержит указатель на объект класса [СписокСтрок](#) с массивом строк, содержащихся в данный момент в выпадающем списке.

Для просмотра и изменения списка необходимо использовать свойства и методы класса **СписокСтрок**.

Если клетка не является перечисляемым полем, то обращение к свойствам списка будет приводить к генерации ошибки.

Пример

```
SS.Cell[1,1].List.Clear;  
for i=1..КоличествоЦен do  
    SS.Cell[3,1].List.Add(Цена[i]);  
end;
```

Класс Объект : КлеткаШаблона

Поле Стил ь / Style

Описание

Стил ь : Стил ьШаблона;
Style : TemplateStyle;

Назначение

Поле предназначено для чтения и изменения стилия клетки, определяемого объектом класса Стил ьШаблона.

Пример

```
Шаблон.ТекущаяКлетка.Стил ь.Шрифт.Размер = 12;
```

Описание

СтильЛогическогоПоля : [Шаблон.СтилиЛогическогоПоля](#);

LogicalStyle : [Template.LogicalStyles](#);

Примечание. Старое название данного поля - ФорматЛогическогоПоля / LogicalFormat сохранено для совместимости данных.

Назначение

Позволяет узнать и изменить способ представления логического поля в клетке. Возможные варианты определяются с помощью перечисления [СтилиЛогическогоПоля](#).

Пример

```
SS : Section;
```

```
proc P3;
```

```
-- клетки 3-го столбца должны содержать "залипающие" кнопки-флаги
```

```
SS.Cell[3,1].LogicalStyle = Template.LogicalButton;
```

```
end;
```

Описание

СтильСтатическойКлетки : Шаблон.СтилиСтатическойКлетки;
StaticStyle : Template.StaticStyles;

Примечание. Старое название данного поля - ФорматСтатическогоТекста / StaticFormat сохранено для совместимости данных.

Назначение

Позволяет узнать и изменить способ представления статического текста в клетке. Возможные варианты определяются с помощью перечисления [СтилиСтатическойКлетки](#).

Пример

```
SS : Section;  
  
proc P3;  
    -- клетки 3-го столбца должны содержать кнопки  
    SS.Cell[3,1].StaticStyle = Template.StaticButton;  
end;
```

Описание

Столбец : Целое;
Column : Integer;

Назначение

Поле предназначено для определения номера колонки секции, в которой расположена клетка. Нумерация ведется с единицы.

Пример

```
func OnEnter(C:TemplateCell;Index:Integer):Logical;  
  if C.Column > 1 then  
    return true;  
  else  
    return false; -- запрещаем ввод в клетки первой колонки  
  end;  
end;
```

Описание

Строка : Целое;
Row : Integer;

Назначение

Поле предназначено для определения номера строки секции, в которой расположена клетка. Нумерация ведется с единицы.

Пример

```
func OnEnter(C:TemplateCell;Index:Integer):Logical;  
  if C.Row > 1 then  
    return true;  
  else  
    return false; -- запрещаем ввод в клетки первой строки  
  end;  
end;
```


Описание

TabStop :Логическое;
TabStop :Logical;

Назначение

Данное поле позволяет определить, разрешен ли переход к клетке шаблона с помощью нажатий клавиши *Tab*.

Когда значение поля равно TRUE, то его можно сделать активным, последовательно нажимая *Tab*, до тех пор, пока фокус ввода не переместится на него. Когда значение равно FALSE, объект можно сделать активным только с помощью мыши.

По умолчанию значение поля равно FALSE. При считывании старых шаблонов все поля, которые разрешено редактировать, т.е. они не имеют атрибута "только для чтения" и все специальные клетки (кнопки, заголовки или гиперссылки) получают значение TabStop = True.

Внимание. В дизайн-режиме при вводе в клетку первого символа "#" или его удаления значение поля TabStop инвертируется.

Пример

```
Секция1.Клетка[1, 1].TabStop = TRUE;
```

Описание

Текст : Строка;
Text : String;

Назначение

Поле предназначено для чтения отображаемой в клетке строки, полученной в результате заданных или используемых по умолчанию форматных преобразований того значения, которое хранится в клетке. На внешний вид данной строки влияют общие настройки программы и свойство [ФорматПоля](#) текущей клетки. Если клетка является вычисляемым полем, данное свойство содержит строку, полученную из обработчика события [ПриВыводе](#). Для поля перечисляемого типа здесь указывается одна из строк-вариантов.

Пример

```
SS : Section;  
ДатаПриемаОплаты: Date;  
  
proc P1;  
  -- присваиваем клетке имя переменной  
  SS.Cell[3,1].Contents = "ДатаПриемаОплаты";  
  -- задаем отображение в полном формате  
  SS.Cell[3,1].FieldFormat = "dd mmmm yyyy";  
  -- устанавливаем хранимое в клетке значение  
  -- при этом автоматически изменяется значение переменной  
  SS.Cell[3,1].Value = 10.06.2006;  
  -- теперь ДатаПриемаОплаты = 10.06.2006;  
  -- в клетке отображается преобразованная в соответствии с  
  -- форматом строка  
  message(SS.Cell[3,1].Text); -- выводит "10 июня 2006"  
end;
```

Описание

ТипКлетки : [Шаблон.ТипыКлетки](#);
CellType : [Template.FieldTypes](#);

Назначение

Позволяет узнать и изменить тип поля клетки. Поле может принимать одно из предопределенных значений перечислимого типа [ТипыКлетки](#) класса **Шаблон**:

- СтатическаяКлетка / StaticCell;
- КлеткаПоле / FieldCell;
- КлеткаВычисляемоеПоле / CalcFieldCell.

Пример

```
-- проход по всем клеткам секции SS
for i=1..SS.ColumnsCount do
  for j=1..SS.RowsCount do
    -- если клетка используется как поле ввода/вывода
    if SS.Cell[i,j].CellType = Kernel.Template.CommonField then
      -- запрещаем ее редактирование
      SS.Cell[i,j].ReadOnly = true;
    end;
  end;
end;
```

Описание

ТипПоля : Шаблон.ТипыПоля;
FieldType : Template.FieldTypes;

Назначение

Позволяет узнать и изменить тип поля, размещенного в клетке. Поле может принимать одно из константных значений, определенных в типе [Шаблон.ТипыПоля](#):

- СтатическийТекст / StaticText
- ОбщееПоле / CommonField
- СтроковоеПоле / StringField
- ЧисловоеПоле / NumericField
- ПолеДаты / DateField
- ЛогическоеПоле / LogicalField
- ПеречислимоеПоле / EnumField
- СсылочноеПоле / ReferenceField
- ВычисляемоеПоле / CalcField

Предупреждение. Свойство **ТипПоля** оставлено в программном интерфейсе для совместимости с предыдущими версиями. Обращаем Ваше внимание на тот факт, что различные комбинации значений полей [CellType](#) и [FieldFormat](#) не всегда однозначно преобразуются в значения поля **ТипПоля**.

Пример

```
-- проход по всем клеткам секции SS
for i=1..SS.ColumnsCount do
  for j=1..SS.RowsCount do
    -- если клетка используется как поле ввода/вывода
    if SS.Cell[i,j].FieldType = Kernel.Template.CommonField then
      -- запрещаем ее редактирование
      SS.Cell[i,j].ReadOnly = true;
    end;
  end;
end;
```

Описание

ФильтрОбзора : Строка;
LookupFilter : String;

Назначение

Данное свойство имеет смысл только для клеток, имеющих [тип поля](#): *СсылочноеПоле (ReferenceField)*.

Свойство позволяет узнать и изменить фильтр для ограничения подмножества записей, из которых можно выбирать запись для ввода в данную клетку. Если клетка связана с картотекой, то этот же фильтр накладывается и на записи картотеки перед её открытием.

Свойство может быть изменено не только программно, но и на стадии проектирования - в диалоге [свойств клетки](#).

Пример

```
-- при открытии бланка
proc BlankOnOpen(Create :Logical);
  -- назначаем для ссылочного поля выбора контрагента
  -- фильтр, отбирающий только наших покупателей
  Header.Cell[2,1].ФильтрОбзора = "МыПродавец = True";
end;
```

Описание

ФорматКлетки : [Шаблон.ФорматыКлетки](#);

CellFormat : [Template.CellFormats](#);

Назначение

Позволяет узнать и изменить формат поля клетки. Поле может принимать одно из предопределенных значений перечислимого типа [ФорматыКлетки](#) класса **Шаблон**:

- ФорматОбщий / CommonFormat;
- ФорматСтроки / StringFormat;
- ФорматЧисла / NumericFormat;
- ФорматДаты / DateFormat;
- ФорматЛогический / LogicalFormat;
- ФорматПеречислимый / EnumFormat;
- ФорматСсылочный / ReferenceFormat.

Описание

ФорматМаскиПоиска : [ФорматыМаскиПоиска](#);
FindMaskFormat : [FindMaskFormats](#);

Назначение

Свойство управляет тем, как будет формироваться маска для поиска/фильтра строки по умолчанию. Свойство может содержать только значения перечислимого типа [ФорматыМаскиПоиска](#).

Свойство доступно только программно на чтение и запись, и учитывается при поиске ссылок при inplace-редактировании в бланке.

В связи с появлением этого свойства изменилось стандартное поведение поиска ссылок при inplace редактировании. Раньше для строковых Lookup-полей (полей обзора) введенная маска безусловно обрамлялась звездочками для поиска по вхождению. Теперь, если пользователь сам ввел звездочку в состав маски, то система их добавлять не будет. Если пользователь не вводил звездочек, то они добавятся автоматически в соответствии с данным свойством FindMaskFormat.

Описание

ФорматПоля : Строка;
FieldFormat : String;

Назначение

Позволяет узнать и изменить форматное преобразование, используемое при выводе значения переменной, связанной с клеткой шаблона. Синтаксис и семантика форматных преобразований описаны в разделе [Преобразования формата](#).

Пример

```
SS : Section;  
ДатаПриемаОплаты: Date;  
  
proc P1;  
  -- присваиваем клетке имя переменной  
  SS.Cell[3,1].Contents = "ДатаПриемаОплаты";  
  -- задаем отображение в полном формате  
  SS.Cell[3,1].FieldFormat = "dd mmmm yyyy";  
  -- устанавливаем хранимое в клетке значение  
  -- при этом автоматически изменяется значение переменной  
  SS.Cell[3,1].Value = 10.06.2006;  
  -- теперь ДатаПриемаОплаты = 10.06.2006;  
  -- в клетке отображается преобразованная в соответствии с  
  -- форматом строка  
  message(SS.Cell[3,1].Text); -- выводит "10 июня 2006"  
end;
```


Описание

ФорматПоляДаты : [Шаблон.ФорматыПоляДаты](#);

DateFormat : [Template.DateFormats](#);

Назначение

Позволяет узнать и изменить формат полей, имеющих тип Дата. Поле может принимать одно из предопределенных значений перечислимого типа [ФорматыПоляДаты](#) класса **Шаблон**:

- ТолькоДата / DateOnly;
- ТолькоВремя / TimeOnly;
- ДатаИВремя / DateAndTime.

Описание

Цвет : Целое;
Color : Integer;

Назначение

Позволяет узнать цвет фона текущей клетки и изменить его. Дополнительно к цвету клетки может быть изменен и [цвет поля](#), расположенного внутри клетки. Это имеет смысл только в том случае, если размеры поля меньше, чем размер клетки (имеется отступ).

Пример

```
SS : Section;  
  
proc OnButtonRed(O:String);  
var i: integer;  
var j: integer;  
  -- делаем все клетки первого кадра в секции красными  
  for i=1..SS.ColumnsCount do  
    for j=1..SS.RowsCount do  
      if SS.Cell[i,j].Frame = 1 then  
        SS.Cell[i,1].Color = clRed;  
      fi;  
    end;  
  end;  
end;
```

Описание

ЦветПоля : Целое;
FieldColor : Integer;

Назначение

Позволяет узнать и изменить цвет поля. Не следует путать цвет поля и [цвет самой клетки](#) шаблона. В том случае, если поле расположено в клетке шаблона с некоторым отступом, то в клетке видна рамка с цветом фона клетки, а внутри нее – поле со своим цветом фона. Когда поле занимает всю клетку и ему назначен непрозрачный цвет, фон самой клетки невидим.

Пример

```
-- клетку делаем синей  
Section1.Cell[i,j].Color = clBlue;  
-- поле в клетке делаем белым  
Section1.Cell[i,j].FieldColor = clWhite;
```

Описание

Шрифт : [Шрифт](#) ;

Font : [Font](#) ;

Назначение

Позволяет узнать параметры шрифта клетки шаблона. Поле доступно только на чтение.

Пример

```
-- заголовок крупно  
Секция1.Клетка[1, 1].Шрифт.Размер = 20;
```

Описание

ЯвляетсяМастером :Логическое;
IsMaster :Logical;

Назначение

Определяет, является ли клетка мастером (свойство = True) для группы объединенных клеток или нет. Мастером-клеткой является самая верхняя, левая клетка в группе объединенных клеток.

Для не объединенных клеток (одна клетка) это свойство всегда = True.

Поле доступно только на чтение.

Пример

```
var i,j: Integer;  
var SS: TemplateSection;  
...  
for i=1..SS.ColumnsCount do  
  for j=1..SS.RowsCount do  
    if SS.Cell[i,j].IsMaster:  
      Message("Клетка i="+ Стр(i) + ", j="+ Стр(j)+ " является мастером");  
    end;  
  end;  
end;
```

Описание

СФокусировать;
SetFocus;

Назначение

Данный метод в отличии от свойства [ТекущаяКлетка](#) не просто делает заданную клетку текущей, но и фокусирует окно фрейма.

Метод не требует вычисления нужного шаблона. Достаточно, чтобы заданная клетка имелась на шаблоне.

Фокусировка не происходит, если клетки не существует или клетка и секция шаблона, которой она принадлежит, не видна экране.

Пример

```
var locCell :TemplateCell;  
...  
if (locCell.FieldType <> Template.StaticText) and  
    not locCell.ReadOnly then  
    locCell.SetFocus;  
end;
```

Описание

ПриВводе : Строка;
OnInput : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при завершении редактирования поля, а именно: после события [OnVerify](#), но перед событием [OnExit](#). Событие позволяет контролировать присвоение новых значений переменной бланка, связанной с полем.

Обработчик

func OnInput(C :TemplateCell; Value :) :Logical;

Параметр **C** содержит указатель на объект класса **КлеткаШаблона**, в котором произошло событие. Параметр **Value** содержит введенное пользователем значение.

В момент вызова события оно еще не присвоено переменной бланка (т.е. существует возможность узнать и новое, и старое значение, что иногда необходимо). Если функция вернет TRUE, то новое (только что введенное пользователем) значение будет присвоено переменной бланка, если FALSE – то нет.

С помощью этого события можно управлять процессом установки значения переменной бланка (блокировать нежелательные изменения, сделанные пользователем), а также реализовывать вычисляемые поля. В последнем случае необходимо задать для клетки вычисляемый тип поля и обрабатывать события [OnOutput](#) и, при необходимости, **OnInput**. Через последнее из них прикладная программа получает информацию о введенном пользователем значении и может его где-либо сохранить, а с помощью первого – реализует вывод на экран. Следует иметь в виду, что для вычисляемого поля возвращаемый обработчиком **OnInput** результат не имеет значения, так как за присвоение нового значения какой-либо переменной (или элементу массива) отвечает прикладная программа.

Пример

```
func Quantity_OnInput(C:TemplateCell; Sum:Numeric):Logical;  
  if C.Contents = "Количество" then  
    if Sum < 0 then -- если было введено отрицательное количество  
      Hint ("Неверное количество!");  
      return FALSE; -- запрещаем ввод некорректных данных  
    end;  
    return true; -- разрешаем присвоение нового значения  
  end;  
end;
```

Описание

ПриВходе : Строка;
OnEnter : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит каждый раз, когда пользователь начинает вводить что-либо в поле ввода клетки.

Обработчик

func OnEnter(Cell :TemplateCell; Index :Integer; Action : [Template.EnterTypes](#)) :Logical;

Параметры:

Cell - содержит указатель на объект класса **КлеткаШаблона**, в котором произошло событие.

Index - содержит номер кадра периодической секции, которой принадлежит задействованная клетка. Если секция непериодическая, второй параметр равен 1.

Action - содержит predetermined константу, описывающую тип активизации клетки.

Обработчик должен вернуть True, если редактирование поля следует разрешить, или False, если редактирование запрещено.

Если в клетку выполняется вставка из буфера обмена и обработчик возвращает False, то в данную клетку вставка не происходит. При вставке в ссылочное поле шаблона проверяется фильтр данного поля, и если вставляемая запись под фильтр не попадает, то вставка в данное поле не происходит.

Пример

```
func Section1_OnEnter(C :TemplateCell;  
  Index :Integer; Action :Template.EnterTypes):Logical;  
  if Index < 2 then  
    return false;  
    -- запрещаем ввод в первый кадр секции  
  else  
    return true;  
  end;  
end;
```


Описание

ПриВыводе : Строка;
OnOutput : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит каждый раз при перерисовке клетки.

Обработчик

```
func OnOutput(Sender :TemplateCell; Value :Variant; Action : Template.OutputTypes; var Format :String) : Variant;
```

Параметры:

Sender - указатель на объект класса **КлеткаШаблона**, в котором произошло событие;
Value - значение переменной, связанной с клеткой. Функция должна вернуть значение, которое требуется отобразить на экране. Тип параметра **Value** должен совпадать с типом переменной, связанной с данной клеткой. Тип возвращаемого значения может быть произвольным;
Action - параметр, позволяющий определить, какое действие пользователя с системой потребовало получения текущего значения клетки;
Format - параметр, позволяющий изменить текущий формат отображения значения, т.е. изменить представление числа или даты в повторяющихся секциях в зависимости от строки секции, величины значения в клетке, при экспорте и др. Изменение этого параметра имеет силу только, когда параметр Action=Output или Export. В качестве входного значения этого параметр используется свойство [TemplateCell.FieldFormat](#).

Данное событие не происходит во время редактирования поля клетки.

Предупреждение! Внутри обработчика нельзя вызывать любые функции, выводящие что-либо на экран (например, диалоги), так как это приведет к заикливанию. Кроме того, внутри обработчика не рекомендуется изменять значения переменных бланка, отображаемых в его полях, поскольку их новые значения не отразятся на экране.

Обработчик позволяет произвольным образом управлять тем, что отображается в клетке. Возвращаемое обработчиком значение не влияет собственно на значение клетки - изменяется лишь значение, предназначенное для вывода.

Если для клетки назначено какое-либо форматное преобразование, то данное событие происходит до преобразования и позволяет подменить для этой клетки строку формата.

Для ссылочных полей событие происходит после разыменования (при этом параметр **Value** содержит не ссылку на другое поле, а значение этого поля).

Внимание. В зависимости от параметра **Action** обработчик может возвращать либо строку, либо число или дату. Однако, в ряде случаев этот обработчик может возвращать в качестве результата не строку, а число или дату независимо от **Action**. За счет параметра **Format** можно добиться нужного форматирования внешнего представления на экране или в экспортируемом документе.

На самом деле при экспорте параметр **Format** можно и нужно менять только при выгрузке в HTML и только в режиме "совместимом с Excel", т.к. в этом случае программой принимаются специальные меры к тому, чтобы значение было считано программой Excel именно числом или датой, а не строкой. В остальных случаях экспорт значения из клетки делается так, как видна строка на экране.

При "совместимом с Excel" экспорте значения «ноль» записываются как «0», а не "" . Также представление вывода в HTML не содержит разделителей триад, а системный разделитель дробной части заменяется на "." . Формат даты корректируется (mmmm - значение приводится к mmm, pp - mm и убираются "[", "]").

Пример

```
-- поле ввода клетки, связанной с числом, будет отображать  
-- введенное число прописью  
func Amount_OnOutput(C:TemplateCell; Value:Numeric;  
    Action :Template.OutputTypes; var F :String) :String;
```

```
    if Action = Template.Output then  
        return Slo(Value);  
    end;  
    return "";  
end;
```

Описание

ПриВыходе : Строка;
OnExit : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит каждый раз, когда пользователь закончил ввод нового значения в поле клетки и оно уже присвоено ей. Внимание! Событие возникает, только если значение поля было изменено.

Обработчик

proc OnExit(C :TemplateCell; Index :Integer);

Параметр **C** содержит указатель на объект класса **КлеткаШаблона**, в котором произошло событие.

Параметр **Index** содержит номер кадра периодической секции, которой принадлежит задействованная клетка. Если секция непериодическая, второй параметр равен 1.

Пример

```
proc Section1_OnExit(C:TemplateCell;Index:Integer);  
  Hint("Изменение принято");  
end;
```

Описание

ПриНаборе : Строка;
OnType : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при нажатии какой-либо клавиши (пока событие происходит только при нажатии Ввод) во время редактирования поля с разрешенным ручным вводом. Обработчик позволяет выполнить нестандартные операции по присвоению вспомогательных переменных или ввести в шаблон дополнительные интерактивные возможности.

Обработчик

func OnType(C: TemplateCell; Key :String; Value :Variant; var NewValue:Variant): Logical;

Параметр **C** содержит указатель на объект класса **КлеткаШаблона**, в котором произошло событие.

Параметр **Key** зарезервирован (будет содержать код нажатой клавиши, а пока всегда равен символу с кодом 13 – клавиша Enter).

Параметр **Value** содержит введенное пользователем значение. Тип **Value** определяется типом адресуемой по ссылке переменной (либо непосредственно типу переменной, отображаемой в поле, если оно не является ссылочным).

Через **NewValue** прикладная программа имеет возможность указать ссылку на объект (если поле ссылочное) или новое значение соответствующего типа (если поле не ссылочное), которые следует присвоить полю.

При работе со ссылочными полями возможна ситуация, когда **Value** и **NewValue** имеют различный тип, например, **Value** – строка, а **NewValue** – ссылка на документ (карточку товара, контрагента и пр.).

Если функция возвращает TRUE, система выполняет стандартную обработку поля (и, в частности, ссылочного поля) с разрешенным ручным вводом (см. [Поле ДляПросмотра / ReadOnly](#)).

Стандартные действия системы описываются ниже. Если функция возвращает FALSE, то в поле вводится значение **NewValue**, которое должен установить обработчик. Обратите внимание, что в этом случае созданное внутри клетки поле ввода (inplace-редактор) автоматически не закрывается, и чтобы его закрыть, необходимо вызвать в обработчике метод: [Template.EndEdit](#)(True);

Стандартная обработка набора символов в поле подразумевает, в большинстве случаев, запись текущего значения (**Value**) в поле и отображение его на экране (пока без присвоения этого значения переменной или полю записи, связанному с клеткой). Особый случай представляет ссылочное поле. Дело в том, что если в ссылочном поле разрешен ручной ввод, то при входе в него не происходит автоматического открытия картотеки, а появляется inplace-редактор, в котором пользователь может набрать некоторый текст.

При этом, если ссылка содержит лишь имя класса (например, "Контрагент") без уточнения имени свойства этого класса через точку (например, "Контрагент.КорИмя") и в свойствах клетки (в диалоге визуального редактора шаблонов) не установлено свойство "Обзор по полю", то набранный текст интерпретируется как DocID (в формате "{DocName:DocID}").

Если ссылка содержит имя класса и, через точку, имя свойства данного класса, то набранный текст используется для поиска документа по данному свойству (полю). При точном совпадении найденный документ автоматически присваивается ссылочной переменной, в противном случае открывается картотека и позиционируется на ближайший похожий документ.

Если установлено свойство "Обзор по полю", то осуществляются аналогичные действия, но по заданному полю.

Пример

```
func ТМЦ_ПриНаборе(C:TemplateCell; Key:String; Value:String;  
    var NewValue:Пример.Справочник):Logical;  
  
    if Key <> CHR(13) then -- ввод строки еще не завершен  
        return Истина;  
    fi;
```

```
-- ...  
Template.EndEdit(Истина);  
return Ложь;  
end;
```

См. также [Пример обработчика события ПриНаборе в клетке шаблона](#).

Описание

ПриНажатии : Строка;
OnClick : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при одиночном или двойном щелчке мышью на клетке, а также при нажатии в ней клавиши Ввод. Событие генерирует только левая (основная) кнопка мыши. В случае нажатия кнопки мыши событие происходит только в клетке, которая до того не была выделена.

Если поле имеет кнопку выбора и эта кнопка была нажата, то система также генерирует событие **ПриНажатии**.

Обратите внимание, что событие вызывается, даже если поле доступно только на чтение ([ДляПросмотра/ReadOnly](#) равно True).

Если поле запрещено ([Разрешен/Enabled](#) равно False), событие **ПриНажатии** не генерируется кнопкой и флагом, размещенном в клетке, однако оно по-прежнему генерируется самой клеткой, которая может быть видна из-за элемента управления за счет наличия широкого бордюра.

Обработчик

func **OnClick**(Cell:TemplateCell; Action : [Template.ClickTypes](#)) : Logical;

Параметры:

Cell - указатель на объект класса **КлеткаШаблона**, в котором произошло событие;
Action - одна из предопределенных констант, описанных в типе [Шаблон.ТипыНажатия](#), которая определяет действие пользователя.

Для полей ввода событие вызывается до обработчика события [ПриВходе](#).

Если обработчик события **ПриНажатии** вернет True, система выполняет действие по умолчанию (вход в поле, если только это не статический текст), иначе нажатие игнорируется, и последующее событие **ПриВходе** не происходит.

Пример

```
func ЧувствительноеМесто_ПриНажатии(C:TemplateCell; A:Template.ClickTypes) :Logical;  
  if A = Шаблон.ДвойноеНажатие then  
    message("Двойной щелчок мышью здесь запрещен!");  
    return False;  
  end;  
  return True;  
end;
```

Описание

ПриОбзоре : Строка;
OnLookup : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при попытке пользователя ввести значение в ссылочное поле или поле ввода с кнопкой обзора (с многоточием).

Если в поле разрешен ручной ввод (включен флаг **Разрешить ручной ввод** в окне свойств клетки), то событие возникает при нажатии кнопки **Обзор** справа от поля (она появляется при включении соответствующей опции в [свойствах клетки](#) в визуальном редакторе шаблона) или *PgUp*.

В ссылочном поле это событие также происходит после нажатия любой клавиши при условии, что ручной ввод в него запрещен и поле в данный момент выделено.

Кроме того, следует иметь в виду, что если поле доступно только на чтение (свойство [ReadOnly](#) равно TRUE или в окне [свойств клетки](#) включен флаг **Поле только для вывода**), то событие **ПриОбзоре** в нем не возникает, однако программист может узнать о нажатии на кнопку обзора (если она есть) в таком поле с помощью обработчика события [ПриНажатии](#).

Обработчик

func OnLookup(C :TemplateCell; Value :Variant; var NewValue :Variant):Logical;

Параметр C содержит указатель на объект класса КлеткаШаблона, в котором произошло событие.

Параметр **Value** содержит текущее значение поля. Тип **Value** определяется несколькими факторами. Если поле содержит разыменованную ссылку, то есть выражение вида **Ссылка на запись.Свойство**, то тип **Value** соответствует типу адресуемого по ссылке свойства (переменной). Если в поле указана лишь ссылка на запись (то есть там фактически хранится переменная <ссылка на запись> без имени свойства), но в диалоге свойств клетки корректно заполнено поле **Обзор по полю**, то тип **Value** тот же, что и поле для обзора. Если же ссылочное поле не содержит разыменования (ни в явном виде, ни в свойствах клетки), то тип **Value** должен быть строковым, поскольку система предполагает указание записи с помощью строкового представления ее **DocID**. Во всех остальных случаях (то есть если поле не является ссылочным) тип **Value** определяется непосредственно типом переменной, отображаемой в поле.

Через **New Value** прикладная программа имеет возможность указать ссылку на объект (если поле ссылочное) или новое значение соответствующего типа (если поле не ссылочное), которое следует присвоить полю.

Если функция возвращает TRUE, то поле обрабатывается системой по умолчанию (см. ниже). Если функция возвращает FALSE, то никакой стандартной обработки не производится и функция должна присвоить переменной **New Value** требуемое значение, которое затем попадает в поле.

Стандартная обработка данного события включает:

- для полей типа "Дата" - открытие календаря;
- для полей типа "Число" - открытие калькулятора;
- для ссылочных полей - открытие картотеки;
- для перечислимых полей - открытие списка выбора;
- для остальных - открытие списка истории.

Пример

```
func Файл_ПриОбзореДиска(Cell:TemplateCell; Value:String; var NewValue:String):Logical;  
var OldValue:String;  
OldValue = NewValue;  
if (ChooseFile(NewValue,  
    'Укажите местонахождение данных', '*.*') = cmOk) then  
    Template.EndEdit(OldValue <> NewValue);  
    return Истина;  
else  
    Template.EndEdit(OldValue <> NewValue);  
    return Ложь;
```

```
fi;  
end;
```


Описание

```
ПриПодсказке : Строка;  
OnHint : String;
```

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при отображении подсказки к текущей клетке.

Обработчик

```
func OnHint(Cell :TemplateCell; var Text :String) :Logical;
```

Параметры:

Cell - указатель на объект класса [КлеткаШаблона](#), в котором произошло событие.

Text - текст подсказки к текущей клетки, который можно изменить.

Функция возвращает True, если разрешается вывести подсказку, иначе - False. Переменная **Text** содержит новый текст, если пользователь изменял текст подсказки, или прежний, заданный при входе в функцию.

Пример

```
func Реквизит_ПриПодсказке(Cell :TemplateCell; var Text :String) :Logical;  
    Result = Интерфейс.Реквизиты[Int(Cell.Section.Column[Cell.Column].Name)].  
    ПриПодсказке(Cell, Text);  
end;
```

Описание

ПриПроверке : Строка;
OnVerify : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит каждый раз, когда пользователь закончил ввод нового значения в поле клетки, но новое значение ей еще не присвоено. При обработке данного события программа может проверить введенное значение на допустимость и, в случае необходимости, откорректировать его.

Обработчик

func OnVerify(C :TemplateCell;Index :Integer; var Value :Variant) :Logical;

Параметр **C** содержит указатель на объект класса **КлеткаШаблона**, в котором произошло событие. Параметр **Index** содержит номер кадра периодической секции, которой принадлежит задействованная клетка. Если секция неперiodическая, второй параметр равен 1. Параметр **Value** содержит новое значение, которое требуется проверить на допустимость, прежде чем присваивать его полю клетки. Тип параметра **Value** должен совпадать с типом переменной, связанной с данной клеткой.

Обработчик может выполнить одно из трех действий:

1. отменить операцию ввода; при этом поле клетки получит прежнее значение, которое оно содержало до изменения; в этом случае обработчик должен вернуть значение FALSE;
2. разрешить присвоение нового значения в том виде, как оно было введено пользователем; в этом случае обработчик должен вернуть значение TRUE;
3. откорректировать введенное пользователем значение, сделав его допустимым, и разрешить присвоение этого модифицированного значения; в этом случае обработчик должен присвоить переменной Value корректное значение и вернуть TRUE.

Пример

```
func Section1_OnVerify(C:TemplateCell;Index:Integer; var Sum:Numeric):Logical;  
  if C.Contents = "Количество" then  
    if Sum < 0 then - если было введено отрицательное количество  
      Sum = 0; -- корректируем его до нуля  
    end;  
    return true; -- и выводим проверенное значение  
  else -- остальные клетки проверяем с помощью своей прикладной функции  
    if CorrectData(C,Sum) then  
      return true; -- если значение допустимо, разрешаем его присвоение  
    end;  
    message("Введено неверное значение "+Str(Sum));  
    return false; -- запрещаем ввод некорректных данных  
  end;  
end;
```

Описание

ПриРисовании : Строка;
OnDraw : String;

Назначение

Данное свойство содержит и позволяет изменить имя обработчика события **ПриРисовании**, которое происходит перед отрисовкой клетки шаблона.

Обработчик

func OnDraw(Cell :TemplateCell; Selected :Logical; var Color :Integer; var FieldColor :Integer; var Font :Font);

Параметр **Cell** содержит указатель на объект класса **КлеткаШаблона**, в котором произошло событие.

Параметр **Selected** определяет, находится ли клетка в выделенном блоке.

Параметры **Color**, **Font** и **FieldColor** определяют соответственно цвет и шрифт клетки, а также [цвет поля](#). Их можно изменить внутри обработчика, управляя тем самым внешним видом клетки шаблона. Если процедура не меняет эти параметры, то для отрисовки будут использованы значения, заданные в шаблоне.

Пример

```
proc Field_OnDraw (aCell :TemplateCell; aSelected :Logical;
  var aColor :Integer; var aFieldColor :Integer; aFont :Font);
  if aCell.Value = '' then
    aFont.Color = clGray; -- рисовать серым
  end;
end;
```

Описание

```
ШиринаТекста({Текст :Строка}) :Число;  
TextWidth({Text :String}) :Numeric;
```

Аргументы

Текст - необязательный строковый параметр, если он не указан, то возвращается ширина текущего текста ячейки.

Назначение

Функция возвращает ширину текста ячейки, с учетом шрифта и стиля текущей ячейки.

Пример

```
var SS : Section;  
var WidthT : Numeric;  
  
WidthT = SS.Cell[1,1].TextWidth("Текст в клетке");
```

Класс *ОбластьШаблона* / *TemplateRange*, производный от родительского класса [Объект](#), используется для работы с выделенным блоком секции. Объекты этого класса применяются в операциях вставки в буфер обмена или копирования из него. Программным способом, используя свойства данного класса разрешается выделять блок периодической секции (строковый, столбцовый, квадратный), внешний вид которого хранится в свойстве *Стиль*.

Для получения ссылки на объекты данного класса используется свойство [Выделение / Selection](#) класса *Шаблон*.

Непосредственно в классе *ОбластьШаблона* определены следующие свойства:

- [Поле Фрейм / Frame](#)
- [Поле Стиль / Style](#)
- [Поле ПервыйСтолбец / BeginColumn](#)
- [Поле ПерваяСтрока / BeginRow](#)
- [Поле ПоследнийСтолбец / EndColumn](#)
- [Поле ПоследняяСтрока / EndRow](#)

Описание

ПерваяСтрока : [СтрокаШаблона](#);
BeginRow : [TemplateRow](#);

Назначение

Возвращает ссылку на первую строку блока, выделенного в периодической секции шаблона. Ссылка определяется на основании ссылки на заданный шаблон.

Поле доступно только на чтение.

Пример

```
var Range :TemplateRange;  
var N      :Integer;  
  
Range = Template.Selection;  
-- определяем номер первой строки  
N = Range.BeginRow.Number;
```

См. [Пример работы с объектами класса ОбластьШаблона](#).

Описание

ПервыйСтолбец : [СтолбецШаблона](#);

BeginColumn : [TemplateColumn](#);

Назначение

Возвращает ссылку на первый столбец блока, выделенного в периодической секции шаблона. Ссылка определяется на основании ссылки на заданный шаблон.

Поле доступно только на чтение.

Пример

```
var Range :TemplateRange;
var N      :Integer;

Range = Template.Selection;
-- определяем номер первого столбца
N = Range.BeginColumn.Number;
```

См. [Пример работы с объектами класса ОбластьШаблона](#).

Описание

ПоследнийСтолбец : [СтолбецШаблона](#);
EndColumn : [TemplateColumn](#);

Назначение

Возвращает ссылку на последний столбец блока, выделенного в периодической секции шаблона. Ссылка определяется на основании ссылки на заданный шаблон.

Поле доступно только на чтение.

Пример

```
var Range :TemplateRange;  
var N      :Integer;  
  
Range = Template.Selection;  
-- определяем номер последнего столбца  
N = Range.EndColumn.Number;
```

См. [Пример работы с объектами класса ОбластьШаблона](#).

Описание

ПоследняяСтрока : [СтрокаШаблона](#);

EndRow : [TemplateRow](#);

Возвращает ссылку на последнюю строку блока, выделенного в периодической секции шаблона. Ссылка определяется на основании ссылки на заданный шаблон.

Поле доступно только на чтение.

Пример

```
var Range :TemplateRange;  
var N      :Integer;  
  
Range = Template.Selection;  
-- определяем номер последней строки  
N = Range.EndRow.Number;
```

См. [Пример работы с объектами класса ОбластьШаблона](#).

Описание

Стиль : [Шаблон.СтилиБлоков](#);
Style : [Template.BlockStyles](#);

Назначение

Позволяет определить заданный стиль в выделенном блоке. Возможные варианты стилей в блоке определяются с помощью перечислимого типа [СтилиБлоков](#);

- ПустойБлок / NoneBlock = 0 - выделение отсутствует;
- СтроковыйБлок / LineBlock = 1 - выделено нескольких строк;
- СтолбцовыйБлок / ColumnBlock = 2 - выделено несколько столбцов;2;
- КвадратныйБлок / BoxBlock = 3 - выделена группа клеток, прямоугольная область, заданная номерами начального столбца и строки, а также номерами конечного столбца и строки.

Поле доступно только на чтение.

Пример

```
var Range :TemplateRange;  
var N      :Integer;  
  
Range = Template.Selection;  
-- определяем стиль выделенного блока  
N = Range.Style;
```

Поле Фрейм / Frame

Описание

Фрейм : [ФреймШаблона](#) ;
Frame : [TemplateFrame](#) ;

Назначение

Возвращает ссылку на фрейм шаблона, которому принадлежит выделенный блок в секции, расположенной на данном фрейме.

Поле доступно только на чтение.

Пример

См. [Пример работы с объектами класса ОбластьШаблона](#).

```
func КопированиеВКлипбордин :Logical;
  var Range      :TemplateRange;
  var локДанные  :Variant[2];
  var i, j       :Integer;

  Result = True;
  Range = Template.Selection;
  ClipboardBeginModify;
  try
    НеНужнаНестандартнаяОбработкаКоманды = True;
    ExecuteCommand('Kernel.Edit.Copy');
    for i = Range.BeginRow.Number..Range.EndRow.Number do
      j = AddInArray(локДанные, []);
      ПодготовитьДанныеИзНастроекДляЗаписиВКлипбордин(i, локДанные[j]);
    end;
    PutToClipboard(локДанные, ПрефиксСодержимогоНастроекВБуфереОбмена);
  finally
    НеНужнаНестандартнаяОбработкаКоманды = False;
    ClipboardEndModify;
    Result = False;
  end;
end;

func ВыделенаКлеткаСФормулой(var аКлетка :TemplateCell) :Logical;
  var локВыделение :TemplateRange;
  var локКлетка    :TemplateCell;
  локВыделение = Template.Selection;
  Result = (локВыделение.BeginColumn = локВыделение.EndColumn) and
    (локВыделение.BeginRow = локВыделение.EndRow);
  if Result then
    локКлетка = Template.CurrentCell;
    if локКлетка <> nil then
      Result = (локКлетка.CellFormat = Template.StringFormat) and
        (локКлетка.OnOutput = 'ПолеФормула_ПриВыводе');
      if Result then
        аКлетка = локКлетка;
      end;
    else
      Result = False;
    end;
  end;
end;
```

Все классы этой группы унаследованы от абстрактного базового класса:

- [ОбъектШаблона / TemplateObject](#)







В группу классов для работы с объектами шаблонами входят следующие классы:

- [Кнопка / Button](#)
- [Надпись / Label](#)
- [Флаг / CheckBox](#)
- [Переключатель / RadioButton](#)
- [Редактор / Edit](#)
- [Список / ComboBox](#)
- [Рисунок / Picture](#)
- [Рамка / Bevel](#)
- [РядЗакладок / TabSet](#)
- [Проигрыватель / MediaPlayer](#)
- [График / Chart](#)
- [OLEКонтейнер / OLEContainer](#)
- [КартотекаШаблона / TemplateCardfile](#)
- [ДеревоКартотеки / CardTree](#)
- [ПодтаблицаШаблона / TemplateSubCardfile](#)
- [Таблица / Grid](#)

Класс *OLEКонтейнер* / *OLEContainer* позволяет работать с OLE-документами (например, в документах Microsoft Word или таблицами Microsoft Excel, видеоматериалами и т.д.) непосредственно в шаблонах Студии.

В классе *OLEКонтейнер* можно использовать все свойства и методы, определенные в родительских классах [Объект](#) и [ОбъектШаблона](#).

Непосредственно в классе *OLEКонтейнер* определены (переопределены) следующие свойства и методы:

-  [Поле РежимАктивизации / AllowInplace](#)
-  [Поле Масштабировать / Stretch](#)
-  [Поле Центрировать / Center](#)
-  [Событие ПриВходе / OnEnter](#)
-  [Событие ПриВыходе / OnExit](#)
-  [Событие ПриНажатии / OnClick](#)

Внимание! Создать объект класса *OLEКонтейнер* из программы на ТБ.Скрипт нельзя. Объекты класса *OLEКонтейнер* создаются самой системой на основе информации о шаблоне, спроектированном в визуальном редакторе бланков Студии.

Как правило, объект *OLEКонтейнер* связывается с помощью визуального редактора бланков с переменной или полем записи типа [OLEДокумент / OLEDocument](#), что позволяет вставлять и редактировать непосредственно в шаблоне практически любые документы современных программ и затем сохранять их в информационной базе.

Описание

Масштабировать :Логическое;
Stretch :Logical;

Назначение

Позволяет узнать, а также изменить способ отображения OLE-документа, находящегося в контейнере. Значение TRUE предписывает масштабировать содержимое контейнера в соответствии с размерами последнего, значение FALSE предписывает не менять размеров документа, при этом он может оказаться как больше, так и меньше окна контейнера. В последнем случае имеет смысл использовать также поле [Центрировать](#).

По умолчанию свойство **Масштабировать** равно TRUE.

Пример

```
OLE:OLEContainer;  
-- задаем способ отображения с помощью флага  
proc FlagStateChange(Sender:CheckBox);  
    OLE.Stretch = Sender.State;  
end;
```

Описание

РежимАктивизации :Логическое;
AllowInplace :Logical;

Назначение

Позволяет узнать, а также изменить текущий режим активизации OLE-документа, находящегося в контейнере. Значение TRUE разрешает активизацию по месту (в окне контейнера), в то время как FALSE – предписывает открывать окно внешнего приложения-поставщика документа. По умолчанию данное свойство равно FALSE.

Пример

```
OLE:OLEContainer;  
-- задаем метод активизации с помощью флага  
proc FlagStateChange(Sender:CheckBox);  
    OLE.AllowInplace = Sender.State;  
end;
```


Поле Центрировать / Center

Описание

Центрировать :Логическое;
Center :Logical;

Назначение

Позволяет узнать, а также изменить способ отображения OLE-документа, находящегося в контейнере. Данное поле имеет смысл только в том случае, если размер документа превосходит размеры окна контейнера, а свойство [Масштабировать](#) равно FALSE.

Значение TRUE в поле **Центрировать** предписывает системе отображать содержимое в центре окна контейнера; при значении FALSE содержимое контейнера отображается с выравниванием по левому верхнему углу окна контейнера.

По умолчанию свойство **Центрировать** равно FALSE.

Пример

```
OLE:OLEContainer;  
-- задаем способ отображения с помощью флага  
proc FlagStateChange(Sender:CheckBox);  
    OLE.Stretch = Sender.State;  
    if NOT OLE.Stretch then  
        OLE.Center = TRUE;  
    end;  
end;
```

Событие ПриВходе / OnEnter

Описание

ПриВходе: Строка;
OnEnter: String;

Назначение

Данное событие происходит, когда OLE-контейнер получает фокус ввода (то есть, становится текущим элементом управления на шаблоне). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB*. Поле **ПриВходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnEnter**(O: **OLEКонтейнер**);
O - объект класса **OLEКонтейнер**, с которым произошло событие.

Пример

```
proc Ole1_ПриВходе(O1: OLEКонтейнер);  
  -- при выделении контейнера  
  -- подготавливаем подсказку  
  O1.Hint = "Для редактирования выполните двойной щелчок мышью";  
end;
```

Событие ПриВыходе / OnExit

Описание

ПриВыходе: Строка;
OnExit: String;

Назначение

Данное событие происходит, когда OLE-контейнер теряет фокус ввода (то есть, перестает быть текущим элементом управления). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB*.

Поле **ПриВыходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnExit**(O: **OLEКонтейнер**);
O - объект класса **OLEКонтейнер**, с которым произошло событие.

Пример

```
proc Ole1_ПриВыходе(O1: OLEContainer);  
  O1.Hint = " ";  
end;
```

Описание

ПриНажатии: Строка;
OnClick: String;

Назначение

Данное событие происходит по щелчку мыши на контейнере или по нажатию клавиши Enter, если контейнер выделен.

Поле **ПриНажатии** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnClick**(O: **OLEКонтейнер**);
O - объект класса **OLEКонтейнер**, с которым произошло событие.

Пример

```
-- по щелчку переключаем свойство масштабирования
-- документа в контейнере
proc Ole1_ПриНажатии(O1: OLEContainer);
  O1.Stretch = NOT O1.Stretch;
end;
```

Класс *График / Chart*, производный от класса *ОбъектШаблона*, представляет собой программный интерфейс для встраивания в шаблоны Студии бизнес-графики.

В классе *График* можно использовать все свойства и методы, определенные в родительских классах [Объект](#) и [ОбъектШаблона](#). Непосредственно в классе *График* определены следующие свойства:

-  [Поле Настройки / Settings](#)
-  [Функция ЗагрузитьДанные / TbgLoadData\]](#)
-  [Поле ЧислоСтраниц / PageCount](#)
-  [Поле АктивнаяСтраница / ActivePage](#)
-  [Поле ЧислоТаблиц / TableCount](#)
-  [Поле АктивнаяТаблица / ActiveTable](#)
-  [Поле КоличествоСерий / SeriesCount](#)
-  [Поле КоличествоЗначений / ValuesCount](#)
-  [Поле НазваниеСерии / SeriesLabel](#)
-  [Поле НазваниеЗначения / ValueLabel](#)
-  [Поле АвтоОбновление / AutoRebuild](#)
-  [Поле Значение / Value](#)
-  [Процедура Обновить / Rebuild](#)
-  [Процедура Очистить / Clear](#)
-  [Событие ПриНажатии / OnClick](#)

Для описания области графика, в которой пользователь может производить щелчки мышью, введен перечислимый тип

-  [ОбластьНажатия / ClickedArea.](#)

Внимание! Создать объект класса *График* из программы на ТБ.Скрипт нельзя. Объекты класса *График* создаются самой системой на основе информации о шаблоне, спроектированном в визуальном редакторе бланков Студии.

Для однократного построения графика нужно выполнить следующую последовательность действий:

1. Задать значения полей **КоличествоСерий** и **КоличествоЗначений**;
2. Заполнить двумерный массив **Значение** данными для отображения;
3. Вызвать метод **Обновить**.

Константы описания областей нажатия

В классе **График** определен специальный тип **ОбластьНажатия/ ClickedArea**, задающий константы для описания областей графика, в которых возможно выполнение щелчков мышью.

В данном типе определены следующие константы:

- **Серии / Series** - щелчок на элементе графика, соответствующем визуализируемому данным;
- **Ось / Axis** - щелчок по координатной оси;
- **Легенда / Legend** - щелчок в области легенды;
- **Фон / Background** - щелчок в другой области.

Константы, определенные в данном типе, используются в обработчике события [ПриНажатии / OnClick](#) класса **ГрафикОтчета/ReportChart**.

Описание

АвтоОбновление :Логическое;
AutoRebuild :Logical;

Назначение

Позволяет включать и отключать режим автоматической перерисовки графика в процессе изменения его данных. Значение TRUE означает, что любое изменение данных будет приводить к перерисовке. Рекомендуется отключать данный режим (FALSE) в случаях, когда необходимо произвести много изменений.

При выключенном режиме автообновления можно в нужные моменты принудительно вызывать перерисовку графика с помощью процедуры [Обновить](#).

По умолчанию автообновление отключено (поле АвтоОбновление равно FALSE).

Пример

```
Chart1.AutoRebuild = TRUE;
```

Описание

АктивнаяСтраница : Целое;
ActivePage : Integer;

Назначение

Поле позволяет узнать и изменить номер текущей страницы, выводимой графиком. Значение может лежать в пределах от 1 до общего [числа страниц](#).

Пример

```
proc ButtonPageRightOnClick(Sender :Button);  
  if Chart1.АктивнаяСтраница < Chart1.ЧислоСтраниц then  
    Chart1.АктивнаяСтраница = Chart1.АктивнаяСтраница + 1;  
  end;  
end;
```


Описание

АктивнаяТаблица : Целое;
ActiveTable : Integer;

Назначение

Поле позволяет узнать и изменить номер текущей таблицы, выводимой графиком. Значение может лежать в пределах от 1 до общего [числа таблиц](#).

Пример

```
proc ButtonTableRightOnClick(Sender :Button);  
  if Chart1.АктивнаяТаблица < Chart1.ЧислоТаблиц then  
    Chart1.АктивнаяСтраница = 1;  
    Chart1.АктивнаяТаблица = Chart1.АктивнаяТаблица + 1;  
  end;  
end;
```

Описание

Значение[НомерСерии:Целое; НомерЗначения:Целое]:Число;
Value[SeriesNumber:Integer; ValueNumber:Integer]:Numeric;

Аргументы

Серии - порядковый номер серий в графике;
НомерЗначения - порядковый номер значения в серии.

Назначение

Позволяет узнать или изменить конкретное значение в указанной серии по его номеру. Для добавления серий и новых значений в серии необходимо устанавливать поля [КоличествоСерий](#) и [КоличествоЗначений](#).

Нумерация серий и значений в сериях ведется от 1.

Пример

```
for i=1..Chart1.SeriesCount do
  for j=1..Chart1.ValuesCount do
    trace("Значение["+Str(i)+", "+Str(j)+"]:"+Str(Chart1.Value[i,j]));
  end;
end;
```

Описание

КоличествоЗначений :Целое;
ValuesCount :Integer;

Назначение

Позволяет узнать и изменить количество значений во всех сериях. Нумерация ведется от 1 до [количества серий](#) в графике.

Пример

```
for i=1..Chart1.SeriesCount do
  for j=1..Chart1.ValuesCount do
    trace("Значение["+Str(i)+","+Str(j)+"]:"+Str(Chart1.Value[i,j]));
  end;
end;
```

Описание

КоличествоСерий :Целое;
SeriesCount :Integer;

Назначение

Позволяет узнать и изменить количество серий на графике. Если в графике уже были установлены какие-либо данные, то при увеличении числа серий данные из прежних серий сохраняются.

Пример

```
Message("На графике задано " + Str(Char1.SeriesCount)+ " серий");
```

Описание

НазваниеЗначения [НомерЗначения :Целое] :Строка;
ValueLabel [ValueNumber :Integer] :String;

Аргументы

НомерЗначения - номер значения в серии. Нумерация значений в сериях ведется от 1 до [количества значений](#).

Назначение

Позволяет узнать и изменить название значения для всех серий.

Пример

```
for j=1..Chart1.ValuesCount do  
    Chart1.ValueLabel[j] = Str(j+1)+"%";  
end;
```

Описание

```
НазваниеСерии[НомерСерии :Целое] :Строка;  
SeriesLabel[SeriesNumber :Integer] :String;
```

Аргументы

НомерСерии - порядковый номер серий в графике. Нумерация ведется от 1 до [количества серий](#) в графике.

Назначение

Позволяет узнать и изменить название серии по ее порядковому номеру.

Пример

```
Месяц : String[] = ["Январь", "Февраль", "Март"];  
proc SetNames;  
var i, n :Integer;  
  if Chart1.SeriesCount > 3 then  
    n = 3;  
  end;  
  if Chart1.SeriesCount < 3 then  
    n = Chart1.SeriesCount;  
  end;  
  
  for i=1..n do  
    Chart1.SeriesLabel[i] = Месяц[i];  
  end;  
end;
```

Поле Настройки / Settings

Описание

Настройки : [НастройкиГрафика](#) ;
Settings : [ChartSettings](#) ;

Назначение

Позволяет узнать и изменить настройки графика, разыменовывая хранящуюся в данном поле ссылку на внутренний объект класса [НастройкиГрафика](#) / [ChartSettings](#).

Поле доступно не только на чтение (что дает возможность изменять свойства объекта **НастройкиГрафика**), но и на запись (то есть объект с настройками можно целиком копировать из одного графика в другой, а также из объектов класса [ГрафикОтчета](#)).

Пример

```
Chart1.Settings.Pattern = true;
```

Описание

ЧислоСтраниц : Целое;
PageCount : Integer;

Назначение

Возвращает количество страниц в данных графика. Поле доступно только на чтение.

Пример

```
proc ButtonPageRightOnClick(Sender :Button);  
  if Chart1.АктивнаяСтраница < Chart1.ЧислоСтраниц then  
    Chart1.АктивнаяСтраница = Chart1.АктивнаяСтраница + 1;  
  end;  
end;
```


Описание

ЧислоТаблиц : Целое;
TableCount : Integer;

Назначение

Возвращает количество таблиц в данных графика. Поле доступно только на чтение.

Пример

```
proc ButtonTableRightOnClick(Sender :Button);  
  if Chart1.АктивнаяТаблица < Chart1.ЧислоТаблиц then  
    Chart1.АктивнаяСтраница = 1;  
    Chart1.АктивнаяТаблица = Chart1.АктивнаяТаблица + 1;  
  end;  
end;
```

Описание

Обновить;
Rebuild;

Назначение

Процедура позволяет принудительно вызывать перерисовку графика.

Если включен режим автообновления (свойство [АвтоОбновление](#) равно TRUE) вызывать данную процедуру не имеет смысла, так как график перерисовывается в нужные моменты автоматически.

Пример

```
Chart1.AutoRebuild = FALSE;  
--  
-- Заполнение графика данными...  
--  
-- Перерисовываем график с измененными данными  
Chart1.Rebuild;
```

Описание

Очистить;
Clear;

Назначение

Позволяет очистить график (удалить все серии, все значения).

Пример

```
proc OnButton1(B1: Button);  
    Chart1.Clear;  
end;
```

Описание

ПриНажатии: Строка;
OnClick: String;

Назначение

Поле позволяет узнать и изменить имя процедуры-обработчика события, которое возникает по щелчку кнопки мыши над графиком отчета.

Обработчик

proc **OnClick**(Sender :Chart; ClickedArea :Chart.ClickedArea; SeriesNo :Integer, ValueNo :Integer);
Sender - объект, в котором произошло событие;
ClickedArea - область, в которой произошло событие; коды областей перечисляются в типе [Chart.ClickedArea](#);
SeriesNo - номер серии, к которой относится элемент графика, на котором произведен щелчок. Номер изменяется от 1 до числа серий;
ValueNo - номер значения, которому соответствует элемент графика, на котором произведен щелчок. Номер изменяется от 1 до числа значений.

Если щелчок был выполнен вне какого-либо рабочего элемента графика, параметры **SeriesNo** и **ValueNo** равны нулю.

Пример

```
-- фрагмент кода класса-наследника формы отчета
proc OnClick(Sender :Chart; ClickedArea :Chart.ClickedArea; SeriesNo :Integer, ValueNo :Integer);
  if ClickedArea = Chart.Series then
    Message(Sender.Value[SeriesNo, ValueNo]);
  end;
end;
```

Описание

```
ЗагрузитьДанные(ИмяФайла :Строка [; ТипДанных :Целое [; НомерТаблицы :Целое]]) : Целое;  
TbgLoadData(FileName :String [; DataType: Integer [; TableNumber :Integer]]) :Integer;
```

Аргументы

ИмяФайла - имя файла с данными;

ТипДанных - необязательный параметр, задающий выводимый показатель; возможные значения:

0 - как сохранено в файле (значение по умолчанию);

1 - начальное сальдо;

2 - оборот;

3 - конечное сальдо.

НомерТаблицы - необязательный параметр, задающий номер таблицы, из которой выводятся данные; по умолчанию выводится первая таблица.

Назначение

Позволяет загрузить в график данные, сохраненные ранее в файле.

Пример

```
ЗагрузитьДанные("C:\Program Files\TB.Corp\Data\Данные1.tbq", 0 , 1);
```











ДеревоКартотеки / CardTree

Класс *ДеревоКартотеки* / *CardTree*, производный от класса *ОбъектШаблона*, представляет собой программный интерфейс для встраивания в шаблон Студии окна с иерархическим представлением групп картотеки. Непосредственно табличную часть окна картотеки можно встроить в шаблон с помощью класса *КартотекаШаблона*.

Внимание! Создать объект класса *ДеревоКартотеки* из программы на ТБ.Скрипт нельзя. Объекты класса *ДеревоКартотеки* создаются самой системой на основе информации о шаблоне, спроектированном в визуальном редакторе бланков Студии.

В классе *ДеревоКартотеки* можно использовать все свойства и методы, определенные в родительских классах [Объект](#) и [ОбъектШаблона](#).

Непосредственно в классе *ДеревоКартотеки* определены следующие свойства:

-  [Поле Цвет / Color](#)
-  [Поле Шрифт / Font](#)
-  [Поле Бордюры / Bevel](#)
-  [Поле Записи / Records / Документы / Documents](#)
-  [Поле ТекущаяГруппа / CurrentGroup](#)
-  [Поле ФильтрГрупп / GroupFilter](#)
-  [Поле ФильтрДерева / TreeFilter](#)
-  [Поле КореньДерева / TreeRoot](#)
-  [Поле ПолеГруппы / GroupField](#)
-  [Поле ПоказыватьУдаленные / ShowDeleted](#)
-  [Поле МожноДобавлять / CanInsert](#)
-  [Поле МожноУдалять / CanDelete](#)
-  [Поле МожноПеремещать / CanMove](#)
-  [Поле МожноКопировать / CanCopy](#)
-  [Поле МожноОткрыть / CanOpen](#)
-  [Поле МожноИзменятьПризнакГруппы / CanGroupSignModify](#)
-  [Поле Редакторы / Editors](#)
-  [Поле РедакторыГрупп / GroupsEditors](#)
-  [Поле Активно / Active / Активен](#)
-  [Процедура Обновить / Update](#)
-  [Событие ПриВходе / OnEnter](#)
-  [Событие ПриВыходе / OnExit](#)
-  [Событие ПриНажатии / OnClick](#)
-  [Событие ПриПеремещении / OnMove](#)
-  [Событие ПриСозданииЗаписи / OnCreateRecord](#)
-  [Событие ПриОткрытииБланка / OnOpenBlank](#)
-  [Событие ПередИзменением / BeforeModify](#)
-  [Событие ПриИзменении / OnModify](#)

Описание

Активно: Логическое;
Active: Logical;

Назначение

Данное поле позволяет узнать и изменить режим работы дерева картотеки на шаблоне. Если его значение равно TRUE, объект автоматически реагирует на изменения в базе данных и параметры настройки самого объекта. Если поле имеет значение FALSE, объект неактивен и не реагирует на изменения данных и/или настроек.

По умолчанию, поле равно TRUE. Устанавливать его в FALSE рекомендуется перед массивными операциями с записями картотеки, отображаемой в дереве, или перед многочисленными изменениями настроек самого дерева. Временно деактивированное дерево при этом не перерисовывается и все операции выполняются быстрее. После выполнения необходимых действий, значение поля следует вернуть в TRUE.

Пример

```
Дерево1: ДеревоКартотеки;  
-- ...  
Дерево1.Активно = FALSE;  
Дерево1.Записи = [Тест.Первичка.Счет];  
НачатьТранзакцию([Тест.Первичка.Счет]);  
-- ... добавляем, меняем, удаляем группы документов  
ЗавершитьТранзакцию;  
Дерево1.Активно = TRUE;
```

Поле Бордюр / Bevel

Описание

Бордюр :Логическое;
Bevel :Logical;

Назначение

Данное поле позволяет узнать и изменить настройку, определяющую, нужно ли выводить трехмерный бордюр вокруг объекта.

Пример

```
Дерево1: ДеревоКартотеки;  
...  
Дерево1.Bevel = true;
```


Описание

Записи : Класс [Запись\[\]](#);
Records :Class [Record\[\]](#);

Назначение

Позволяет задать или узнать классы документов, по которым строится запрос в дереве картотеке.

Пример

```
Tree: CardTree; -- объект дерево картотеки
-- по нажатию кнопки отображаем в дереве картотеки
-- записи о поставщиках
proc кнПоставщик_ПриНажатии (Sender :String);
    Tree.Записи = [Пример.Справочники.Поставщики];
end;
-- по нажатию другой кнопки отображаем в дереве картотеки
-- записи о покупателях
proc кнПокупатель_ПриНажатии (Sender :String);
    Tree.Записи = [Пример.Справочники.Покупатели];
end;
```

Поле КореньДерева / TreeRoot

Описание

КореньДерева: Строка;
TreeRoot: String;

Назначение

Данное поле позволяет узнать и изменить название корневой группы дерева картотеки. По умолчанию, корень имеет имя "Картотека".

Пример

```
Дерево1: ДеревоКартотеки;  
--...  
Дерево1.КореньДерева = "Контрагенты";
```

Поле МожноДобавлять / CanInsert

Описание

МожноДобавлять :Логическое;
CanInsert :Logical;

Назначение

Поле определяет, можно ли интерактивно добавлять записи посредством выполнения команд в объекте дерева картотеки.

Пример

```
CardTree1.CanInsert = CheckBox1.State;
```

Описание

МожноИзменятьПризнакГруппы :Логическое;
CanGroupSignModify :Logical;

Назначение

Поле определяет, можно ли интерактивно преобразовывать запись в группу и обратно.

Пример

```
CardTree1.CanGroupSignModify = CheckBox6.State;
```

Описание

МожноКопировать :Логическое;
CanCopy :Logical;

Назначение

Поле определяет, можно ли интерактивно копировать записи в дереве картотеки.

Пример

```
CardTree1.CanCopy = CheckBox4.State;
```

Поле МожноОткрыть / CanOpen

Описание

МожноОткрыть :Логическое;

CanOpen :Logical;

Назначение

Поле определяет, можно ли открыть бланк-редактор для редактирования записи из дерева картотеки.

Пример

```
CardTree1.CanOpen =CheckBox5.State;
```

Описание

МожноПеремещать :Логическое;
CanMove :Logical;

Назначение

Поле определяет, можно ли интерактивно перемещать записи в дереве картотеки.

Пример

```
CardTree1.CanMove = CheckBox3.State;
```

Описание

МожноУдалять :Логическое;
CanDelete :Logical;

Назначение

Поле определяет, можно ли интерактивно удалять записи посредством выполнения команд в объекте дерева картотеки.

Пример

```
CardTree1.CanDelete = CheckBox2.State;
```


Описание

ПоказыватьУдаленные : Логическое;
ShowDeleted : Logical;

Назначение

Данное поле позволяет узнать, включен ли режим просмотра удаленных групповых записей или нет, а также включать/отключать его.

Пример

```
Дерево1: ДеревоКартотеки;  
-- процедура включает/отключает видимость удаленных групп  
proc ButtonClick(S:String);  
    Дерево1.ПоказыватьУдаленные =  
        NOT Дерево1.ПоказыватьУдаленные;  
end;
```

Описание

ПолеГруппы: Строка;
GroupField: String;

Назначение

Данное поле позволяет узнать и изменить имя поля записи, из которого берутся значения, отображающиеся в дереве. Если это поле не задано, в дереве выводятся идентификаторы групповых документов (**DocID**).

Пример

```
Дерево1: ДеревоКартотеки;  
-- ...  
Дерево1.ПолеГруппы = "Наименование";
```

Описание

Редакторы :Класс [ФормаБланка](#)[];
Editors :Class [BlankForm](#)[];

Назначение

Данное поле предоставляет доступ к массиву классов бланков-редакторов, назначенных для данного дерева картотеки. Массив можно считывать поэлементно и целиком. Поэлементное присвоение новых значений не допускается, однако можно записать в поле целиком новый массив классов. Количество элементов в массиве можно получить с помощью функции [ДлинаМассива](#) класса Система.

Поле позволяет узнать и изменить перечень классов бланков-редакторов, использующихся для обычных (негрупповых) записей. Для того чтобы узнать или изменить перечень классов бланков-редакторов групповых записей применяется поле [РедакторыГрупп](#).

Несмотря на то, что в дереве картотеки отображаются только групповые записи, дерево поддерживает команду добавления новых негрупповых записей в текущую выделенную группу.

Пример

```
var n: integer;  
n = LengthOfArray(CardTree1.Editors);  
if n > 1 then  
    CardTree1.Editors = [ДвижениеСредств];  
end;
```

Описание

РедакторыГрупп :Класс [ФормаБланка](#)[];
GroupsEditors :Class [BlankForm](#)[];

Назначение

Данное поле предоставляет доступ к массиву классов бланков-редакторов, назначенных для данного дерева картотеки. Массив можно считывать поэлементно и целиком. Поэлементное присвоение новых значений не допускается, однако можно записать в поле целиком новый массив классов. Количество элементов в массиве можно получить с помощью функции [ДлинаМассива](#) класса Система.

Поле позволяет узнать и изменить перечень классов бланков-редакторов, использующихся для групповых записей. Для того чтобы узнать или изменить перечень классов бланков-редакторов обычных (негрупповых) записей применяется поле [Редакторы](#).

Пример

```
var n: integer;  
  n = LengthOfArray(CardTree1.GroupsEditors);  
  if n = 0 then  
    CardTree1.GroupsEditors = [ДвижениеСредств];  
  end;
```

Описание

ТекущаяГруппа : [Запись](#);
CurrentGroup: [Record](#);

Назначение

Данное свойство позволяет узнать или установить запись, задающую текущую группу в дереве иерархической картотеки на шаблоне.

Пример

```
CardGoods: TemplateCardfile; -- объект шаблона
TreeGoods: CardTree; -- объект шаблона
-- после каждого перемещения курсора по картотеке шаблона
-- проверяем, не является ли текущая запись группой, и если да -
-- устанавливаем ее текущей в дереве картотеки
proc CardGoods_OnMove(D:Document);
  try
    if D.IsGroup then
      -- входим в группу
      CardGoods.CurrentGroup = D;
      TreeGoods.CurrentGroup = D;
    end;
  except
  end;
end;
```

Описание

ФильтрГрупп : [СписокЗаписей](#);

GroupFilter : [RecordList](#);

Назначение

Данное поле позволяет установить и прочесть текущее значение фильтра групп, определяющего условие отбора записей по их расположению в иерархии групп. Иными словами, с помощью данного поля можно указать, элементы каких групп должны отображаться в картотеке шаблона.

Данное поле аналогично одноименному полю [ФильтрГрупп/GroupFilter](#) класса [Картотека/CardFile](#)

Пример

```
Tree: CardTree; -- объект шаблона
proc УстановитьОбзорГруппыТоваров(ГруппаТоваров: Тест.Справочники.Товар);
    Tree.ФильтрГрупп.Очистить;
    Tree.ФильтрГрупп.Добавить(ГруппаТоваров);
end;
```

Описание

ФильтрДерева: Строка;
TreeFilter: String;

Назначение

Данное поле позволяет установить и прочесть текущее значение дополнительного пользовательского фильтра, задающего условие отбора записей дерева.

Пример

```
CardTree1: ДеревоКартотеки;  
ФильтрНаДерево: Строка;  
...  
ФильтрНаДерево = CardTree1.TreeFilter;
```

Поле Цвет / Color

Описание

Цвет : Целое;
Color :Integer;

Назначение

Данное поле позволяет узнать и изменить цвет фона под объектом.

Пример

```
var R, G, B, C: Integer;  
-- составляющие цвета должны лежать в пределах 0-255  
R = Mod(Red,256); -- красный  
G = Mod(Green,256); -- зеленый  
B = Mod(Blue,256); -- синий  
C = R + G*256 + B*256*256;  
CardTree1.Color = C;
```


Поле Шрифт / Font

Описание

Шрифт : [Шрифт](#) ;

Font : [Font](#) ;

Назначение

Позволяет узнать параметры шрифта объекта ДеревоКартотеки. Поле доступно только на чтение.

Пример

```
ДеревоКартотеки1.Шрифт.Жирный = Истина;
```

Описание

ПередИзменением: Строка;
BeforeModify: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит перед началом изменения одной записи. Если какое-либо действие выполняется сразу над группой записей, то событие **ПередИзменением** происходит лишь один раз.

Обработчик

```
func BeforeModify(Sender :CardTree; Action :Cardfile.ModifyActions; TheRecord :Record; TheGroup :Record; var AskConfirm :Logical) :Logical;
```

Параметры:

Sender - указатель на объект класса **ДеревоКартотеки**, с которым произошло событие;
Action - действие, которое производится над записями дерева картотеки (удаление, восстановление, перемещение или копирование);
TheRecord - ссылка на запись, над которой производится действие, если действие выполняет сразу над несколькими записями, параметр **TheRecord** равен nil;
TheGroup - целевая группа, в которую переносится запись или набор записей в случае перемещения или копирования;
AskConfirm - логический параметр, позволяет указать системе, следует ли запрашивать подтверждения у пользователя. Если данный параметр при выходе из обработчика равен TRUE, то пользователю выдается запрос, а в случае FALSE - нет. По умолчанию (при входе в обработчик) **AskConfirm** равно TRUE.

Если обработчик возвращает TRUE, то тем самым разрешается выполнение действия и будут сгенерированы все последующие события - в первую очередь [ПриИзменении](#). Если обработчик **ПередИзменением** вернет FALSE, выполнение действия над записями картотеки прекращается.

При входе в обработчик Result равен FALSE (действия запрещены).

Пример

```
func Card_OnBeforeModify(Sender :CardTree; Action :Cardfile.ModifyActions; TheRecord :Record; TheGroup :Record; var AskConfirm :Logical) :Logical;  
  if Action = Cardfile.CopyRecord then  
    -- копирование запрещено  
    return FALSE;  
  elseif Action = Cardfile.DeleteRecord then  
    -- удаление требует подтверждения  
    AskConfirm = TRUE;  
  else  
    -- остальные действия выполнять без подтверждений  
    AskConfirm = FALSE;  
  end;  
end;
```

Описание

ПриВходе: Строка;
OnEnter: String;

Назначение

Данное событие происходит, когда объект дерева картотеки получает фокус ввода (то есть, становится текущим элементом управления на шаблоне). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB* или по щелчку мыши. Поле **ПриВходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnEnter**(Sender :CardTree);

Параметры:

Sender - указатель на объект класса **ДеревоКартотеки**, с которым произошло событие.

Пример

```
proc ДеревоКартотеки_ПриВходе(CT :CardTree) ;  
    CT.Font.Bold = TRUE;  
end;
```

Описание

ПриВыходе: Строка;
OnExit: String;

Назначение

Данное событие происходит, когда объект дерева картотеки теряет фокус ввода (то есть, перестает быть текущим элементом управления). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB* или по щелчку мыши.

Поле **ПриВыходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

```
proc OnExit(Sender :CardTree);
```

Параметры:

Sender - указатель на объект класса **ДеревоКартотеки**, с которым произошло событие.

Пример

```
proc ДеревоКартотеки_ПриВыходе(CT :CardTree);  
  CT.Font.Bold = FALSE;  
end;
```

Описание

ПриИзменении: Строка;
OnModify: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит перед началом изменения конкретной записи. Это событие генерируется сразу после события [ПередИзменением](#), причем если определен обработчик события **ПередИзменением**, то в нем должно быть разрешено проведение действия (функция-обработчик **ПередИзменением** должна вернуть TRUE).

Если какое-либо действие выполняется сразу над группой записей, то событие **ПриИзменении** происходит для каждой записи отдельно.

Обработчик

```
func OnModify(Sender :CardTree; Action :Cardfile.ModifyActions; TheRecord :Record; TheGroup :Record) :Logical;
```

Параметры:

Sender - указатель на объект класса **ДеревоКартотеки**, с которым произошло событие;
Action - действие, которое производится над записью картотеки (удаление, восстановление, перемещение или копирование);
TheRecord - ссылка на запись, над которой производится действие;
TheGroup - группа, в которую переносится запись в случае перемещения или копирования. При копировании записи параметр **TheRecord** содержит не исходную запись, а её копию, причем в этой копии еще не заполнено поле **GroupDoc** (значения всех остальных полей соответствуют исходной записи).

Если обработчик возвращает TRUE, система выполняет стандартную обработку выполняемого действия. Если обработчик возвращает FALSE, система не выполняет стандартную обработку, однако программист может закодировать непосредственно в теле обработчика все необходимые операции (в том числе, непредусмотренные стандартной обработкой).

В частности, обработчик может исправить часть полей записи (например, чтобы избежать конфликтов уникальных полей) и вернуть TRUE. Либо он может самостоятельно записать запись и вернуть FALSE. Если обработчик вернет FALSE, не записав запись, это будет означать, что запись не будет скопирована. При этом, если копировавшаяся запись является групповой, то не копируются и все её элементы.

При входе в обработчик Result равен FALSE (действия запрещены).

Пример

```
func Tree_OnModify(Sender: CardTree; Action :Cardfile.ModifyActions; TheRecord :Record; TheGroup :Record) :Logical;  
  if Action = Cardfile.UndeleteRecord then  
    -- восстановление запрещено  
    return FALSE;  
  elseif Action = Cardfile.CopyRecord then  
    -- изменяем значения некоторых полей  
    TheRecord.Name = "<пусто>";  
    -- соглашаемся разместить запись в группе,  
    -- куда её копирует пользователь  
    TheRecord.GroupDoc = TheGroup;  
    -- сохраняем запись  
    TheRecord.Post;  
    -- говорим системе, что дальнейшая  
    -- автоматическая обработка события не требуется  
    return FALSE;  
  end;  
  return TRUE;  
end;
```

Описание

ПриНажатии: Строка;
OnClick: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит по щелчку мышью в объекте **ДеревоКартотеки** или при нажатии клавиши **Enter**, когда фокус ввода находится в этом объекте.

Обработчик

```
func OnClick(Sender :CardTree; Action : Template.ClickTypes; Rec :Record) :Logical;
```

Параметры:

Sender - указатель на объект класса **ДеревоКартотеки**, с которым произошло событие;

Rec - текущий документ (запись);

Action - одна из предопределенных констант, описанных в типе [Шаблон.ТипыНажатия](#), которая определяет действие пользователя:

0 - одинарный щелчок мышью;

1 - двойной щелчок мышью;

2 - нажатие клавиши **Enter**.

Если функция возвращает True, выполняется стандартная обработка события, иначе - не выполняется (False).

Пример

```
-- обработчик события "щелчок мышью в дереве картотеки"  
proc CardTree_OnClick(CT :CardTree);  
    Hint(CT.CurrentGroup);  
end;
```

Описание

ПриОткрытииБланка: Строка;
OnOpenBlank: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит перед открытием бланка-редактора для текущей записи дерева картотеки.

Обработчик

```
func OnOpenBlank(Sender :CardTree; Action :Integer; Doc :Document) :Logical;
```

Параметры:

Sender - указатель на объект класса **ДеревоКартотеки**, с которым произошло событие;

Action - определяет тип события:

- 0** - редактируется существующий документ;
- 1** - добавляется новый документ;
- 2** - добавляется копия документа (документ клонируется);

Doc - редактируемый документ (запись).

Если функция возвращает TRUE, происходит стандартная обработка события, то есть открытие бланка-редактора. Если функция возвращает FALSE, стандартная обработка не выполняется.

Пример

```
func CardTree_OnOpenBlank(Sender :CardTree; Action :Integer; Document :Document):Logical;  
  if Action=0 then  
    return TRUE;  
  else  
    message('Для создания новых документов воспользуйтесь'+  
            'командами Добавить (Ins) или Дублировать');  
  fi;  
  return FALSE;  
end;
```

Описание

ПриПеремещении: Строка;
OnMove: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при перемещении текстового курсора в дереве картотеки от одной группы к другой.

Обработчик

```
proc OnMove(Sender :TemplateCardfile; D: Document);
```

Параметры:

Sender - указатель на объект класса **ДеревоКартотеки**, с которым произошло событие;

Событие позволяет управлять состоянием дерева картотеки и связанных с ним элементов управления в зависимости от контекста ввода.

Пример

```
proc CardTree_OnMove(CT :CardTree);  
  hint(CT.CurrentGroup);  
end;
```


Описание

ПриСозданииЗаписи: Строка;
OnCreateRecord: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при создании записи в дереве картотеки.

Обработчик

```
func OnCreateRecord(Sender :CardTree; var ARec :Record) :Logical;
```

Параметры:

Sender - указатель на объект класса **ДеревоКартотеки**, с которым произошло событие;
ARec - указатель на создаваемую запись. При входе в обработчик параметр **ARec** равен *nil*.

Данное событие возникает только при добавлении новой записи в дерево картотеки, но не при дублировании старой.

Обработчик должен вернуть FALSE в случае отказа от дальнейшей стандартной обработки и TRUE, если ее нужно продолжить. Если **ARec** остался равен *nil* по окончании работы обработчика и возвращаемое значение равно TRUE, то произойдет стандартный выбор класса записи и дальнейшая обработка. В случае, когда **ARec** не равен *nil* и результат равен TRUE.

Пример

```
-- На шаблоне размещен выпадающий список  
-- ComboBoxClassName с именами классов записей;  
-- При добавлении новой записи в дерево картотеки, программа  
-- автоматически выбирает класс, выделенный в списке.  
func CardTree_OnCreateRecord (Tree:CardTree; var ARec :Record) :Logical;  
  try  
    ARec = FindClass(ComboBoxClassName.Text).Create;  
  except  
  end;  
  return TRUE;  
end;
```

Класс *КартотекаШаблона/TemplateCardfile*, производный от класса *ОбъектШаблона*, представляет собой программный интерфейс для встраивания в шаблон Студии упрощенного окна картотеки (табличной части картотеки, без дерева и подтаблиц).










Дерево иерархической картотеки можно встроить в шаблон с помощью класса [ДеревоКартотеки](#). Подтаблицы картотеки можно выводить на шаблон с помощью объектов класса [ПодтаблицаШаблона](#). С точки зрения пользователя картотека на шаблоне ведет себя так же, как и табличная часть обычной картотеки. Это относится к способам [редактирования записей](#) и [настройки](#) табличной части.







Внимание! Создать объекты класса *КартотекаШаблона* из программы на встроенном языке ТБ.Скрипт нельзя. Объекты создаются автоматически на основе информации о шаблоне, спроектированном в визуальном редакторе бланков Студии.

В классе *КартотекаШаблона* можно использовать все свойства и методы, определенные в родительских классах [Объект](#) и [ОбъектШаблона](#).

Непосредственно в классе *КартотекаШаблона* определены следующие свойства:

-  [Поле Цвет / Color](#)
-  [Поле Шрифт / Font](#)
-  [Поле Бордюр / Bevel](#)
-  [Поле Запрос / Query](#)
-  [Поле Записи / Records / Документы / Documents](#)
-  [Поле Текущий / Current / Текущая](#)
-  [Поле ТекущаяГруппа / CurrentGroup](#)
-  [Поле Упорядочивание / Order](#)
-  [Поле Фильтр / Filter](#)
-  [Поле ФильтрПользователя / UserFilter](#)
-  [Поле ИспользоватьФильтрПользователя / UserFilterOn](#)
-  [Поле ФильтрГрупп / GroupFilter](#)
-  [Поле Иерархическая / Hierarchical](#)
-  [Поле ПоказыватьИерархию / ShowHierarchy](#)
-  [ПоказыватьКоличество / ShowCount](#)
-  [Поле ПоказыватьУдаленные / ShowDeleted](#)
-  [Поле ПоказыватьИтоги / ShowTotal](#)
-  [Поле ГруппыВначале / GroupsFirst](#)
-  [Поле ОписаниеИзПоля / DescrFromField](#)
-  [Поле ПоискПриНаборе / FindAtType](#)
-  [Поле МожноДобавлять / CanInsert](#)
-  [Поле МожноУдалять / CanDelete](#)
-  [Поле МожноПеремещать / CanMove](#)
-  [Поле МожноКопировать / CanCopy](#)
-  [Поле МожноРедактировать / CanEdit](#)
-  [Поле МожноОткрыть / CanOpen](#)
-  [Поле МожноИзменятьПризнакГруппы / CanGroupSignModify](#)
-  [Поле Редакторы / Editors](#)
-  [Поле РедакторыГрупп / GroupsEditors](#)
-  [Процедура ВыделитьЗаписиSelectRecords](#)
-  [Процедура СнятьВыделениеЗаписей / DeselectRecords](#)
-  [Поле КоличествоВыделенных / SelectedCount](#)
-  [Поле Выделенные / Selected](#)
-  [Поле ТекущийСтолбец / CurrentColumn](#)
-  [Поле ПозТекущегоСтолбца / CurrentColumnPos](#)
-  [Поле КоличествоСтолбцов / ColumnsCount](#)
-  [Поле Столбец / Column](#)
-  [Поле СтолбецПоПолю / ColumnByField](#)

 [Поле ФиксированныхСтолбцов / FixedColumns](#)
 [Функция ДобавитьСтолбец / AddColumn](#)
 [Функция ВставитьСтолбец / InsertColumn](#)
 [Процедура УдалитьСтолбец / DeleteColumn](#)
 [Процедура Удалить / Delete](#)
 [Процедура Восстановить / Undelete](#)
 [Процедура НачатьРедактирование / BeginEdit](#)
 [Процедура ЗавершитьРедактирование / EndEdit](#)
 [Процедура Обновить / Update](#)

 [Событие ПриВходе / OnEnter](#)
 [Событие ПриВыходе / OnExit](#)
 [Событие ПриПеремещении / OnMove](#)
 [Событие ПриНажатии / OnClick](#)
 [Событие ПриСозданииЗаписи / OnCreateRecord](#)
 [Событие ПриОткрытииБланка / OnOpenBlank](#)
 [Событие ПриРедактировании / OnEdit](#)
 [Событие ПриЗаписи / OnPost](#)
 [Событие ПриОтмене / OnCancel](#)
 [Событие ПриОформлении / OnRearrange](#)
 [Событие ПриСменеВыделения / OnChangeSelected](#)
 [Событие ПриСменеГруппы / OnChangeGroup](#)
 [Событие ПередИзменением / BeforeModify](#)
 [Событие ПриИзменении / OnModify](#)
 [Событие ПриРисованииСтроки / OnDrawRow](#)

Поле Бордюр / Bevel

Описание

Бордюр :Логическое;
Bevel :Logical;

Назначение

Данное поле позволяет узнать и изменить настройку, определяющую, нужно ли выводить трехмерный бордюр вокруг объекта.

Пример

```
КартотекаШаблона1.Bevel = true;
```

Поле Выделенные / Selected

Описание

Выделенные[Индекс: Целое] : [Запись](#);
Selected[Index: Integer] : [Record](#);

Назначение

Данное поле позволяет получить ссылку на одну из выделенных записей (документ) по ее номеру. Индекс должен лежать в пределах от 1 до [количества выделенных записей](#).

Поле доступно только на чтение.

Пример

```
proc кнУдалить_ПриНажатии (Sender :Button);  
  var i :Integer;  
  for i = 1..TemplateCardfile1.SelectedCount do  
    TemplateCardfile1.Selected[i].Delete;  
  end;  
end;
```

Описание

ГруппыВначале :Логическое;
GroupsFirst:Logical;

Назначение

По умолчанию поле имеет значение TRUE. В этом случае для иерархических картотек группы располагаются в верхней части списка перед простыми элементами.

Пример

```
КартотекаШаблона1 :КартотекаШаблона;  
proc ButtonClick(B:Button);  
  if КартотекаШаблона1.ПоказыватьИерархию:  
    КартотекаШаблона1.ГруппыВначале = Истина;  
  end;  
end;
```

Описание

Записи : Класс [Запись\[\]](#);
Records :Class [Record\[\]](#);

Назначение

Позволяет задать или узнать классы документов, по которым строится запрос в картотеке.

Использование этого поля более предпочтительно, чем разыменовывание поля [Запрос](#) и обращение к его свойству **Записи**, так как все изменения в запросе действуют лишь до первого изменения содержимого картотеки.

Пример

```
Card: TemplateCardfile; -- объект картотеки на шаблоне
-- по нажатию кнопки отображаем в картотеке приходные накладные
proc кнПриход_ПриНажатии (Sender :String);
    Card.Documents = [Пример.Накладные.Приходные];
end;
-- по нажатию другой кнопки отображаем в картотеке расходные накладные
proc кнРасход_ПриНажатии (Sender :String);
    Card.Documents = [Пример.Накладные.Расходные];
end;
```

Описание

Запрос : [Запрос](#);
Query : [Query](#);

Назначение

Поле позволяет получить доступ к объекту [Запрос/Query](#), ассоциированному с данной картотекой. Манипулируя свойствами запроса, можно управлять поведением картотеки и получать сведения о ее текущем состоянии, в частности, определять текущий (выделенный) документ.

Следует иметь в виду, что в иерархической картотеке в режиме отображения иерархии (то есть, когда одновременно в таблице картотеки выводится содержимое только одной группы, одного уровня иерархии) запрос содержит только документы открытой группы. Поэтому попытка присвоить свойству [Текущий](#) запроса некоторый документ, не входящий в открытую группу, генерирует исключение. Иными словами, в запросе иерархической картотеки можно программно менять текущий документ только в пределах открытой группы. Если необходимо переместиться из одной группы в другую, следует изменить свойство **Текущий** непосредственно у картотеки шаблона. При этом система автоматически закроет старый запрос и построит новый.

Внимание! Свойства класса **Запрос - Записи, Фильтр, Упорядочивание** - имеются также и в классе **КартотекаШаблона**. Изменение этих свойств в запросе действуют лишь до первого изменения содержащихся в картотеке записей. После каждого изменения картотеки система инициализирует вышеуказанные свойства поля **Запрос** соответствующими значениями одноименных свойства объекта **КартотекаШаблона**. Таким образом, если необходимо перманентно изменить какое-либо из свойств, это следует делать через поле **Картотека**, а не **Запрос**.

Пример

```
Card: TemplateCardfile; -- объект картотеки на шаблоне
-- по нажатию кнопки на бланке удаляем текущую запись
-- из объекта Card класса КартотекаШаблона
proc кнУдалить_ПриНажатии (Sender :String);
    Card.Query.Current.Delete;
end;
```


Описание

Иерархическая :Логическое;
Hierarchical :Logical;

Назначение

Свойство позволяет узнать, является ли картотека иерархической, или нет. Если свойство равно True, то картотека является иерархической. Поле доступно на чтение и запись.

От значения данного свойства зависит работа свойства [ShowHierarchy](#).

Пример

```
var TempCardfile1 :TemplateCardfile;  
...  
proc F1(Sender:Button);  
    if TempCardfile1.Hierarchical then  
        TempCardfile1.ShowHierarchy = True;  
    fi;  
end;
```

Описание

ИспользоватьФильтрПользователя : Логическое;
UseUserFilter : Logical;
UserFilterOn : Logical;

Назначение

Данное поле позволяет узнать, включен ли фильтр пользователя, а также включать/отключать его.

Пример

```
-- процедура включает/отключает "фильтр по Москве"
proc ButtonClick(S:String);
  if КартотекаШаблона1.ФильтрПользователя = '' then
    КартотекаШаблона1.ФильтрПользователя = 'COOTB(Адрес, "*Москва*")';
    КартотекаШаблона1.ИспользоватьФильтрПользователя = TRUE;
  else
    КартотекаШаблона1.ФильтрПользователя = '';
    -- автоматически сбрасывает ИспользоватьФильтрПользователя
  end;
end;
```

Описание

КоличествоВыделенных : Целое;
SelectedCount : Integer;

Назначение

Данное поле содержит количество выделенных в данный момент записей картотеки. Поле доступно только на чтение.

Пример

```
proc кнУдалить_ПриНажатии (Sender :Button);  
  var i :Integer;  
  for i = 1..TemplateCardfile1.SelectedCount do  
    TemplateCardfile1.Selected[i].Delete;  
  end;  
end;
```

Описание

КоличествоСтолбцов : Целое;
ColumnsCount : Integer;

Назначение

Данное свойство содержит количество столбцов в картотеке шаблона. Поле доступно только на чтение.

Для программного добавления/удаления столбцов необходимо пользоваться методами [ДобавитьСтолбец](#), [ВставитьСтолбец](#), [УдалитьСтолбец](#).

Пример

```
Card: TemplateCardfile; -- объект шаблона
-- выводим имена полей всех столбцов
proc TraceColumns(Name: String);
var i: integer;
    for i=1..Card.ColumnsCount do
        trace(Card.Column[i].FieldName);
    end;
end;
```

Описание

МожноДобавлять :Логическое;
CanInsert :Logical;

Назначение

Поле определяет, можно ли интерактивно добавлять записи в картотеку.

В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге [свойств объекта КартотекаШаблона](#) (на странице "Правка").

Пример

```
TemplateCardfile1.CanInsert = CheckBox1.State;
```

Описание

МожноИзменятьПризнакГруппы :Логическое;
CanGroupSignModify :Logical;

Назначение

Поле определяет, можно ли интерактивно преобразовывать запись картотеки в группу и обратно.

В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге [свойств объекта КартотекаШаблона](#) (на странице "Правка").

Пример

```
TemplateCardfile1.CanGroupSignModify = CheckBox7.State;
```

Описание

МожноКопировать :Логическое;
CanCopy :Logical;

Назначение

Поле определяет, можно ли интерактивно копировать записи в картотеке.

В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге [свойств объекта КартотекаШаблона](#) (на странице "Правка").

Пример

```
TemplateCardfile1.CanCopy = CheckBox4.State;
```

Описание

МожноОткрыть :Логическое;

CanOpen :Logical;

Назначение

Поле определяет, можно ли открыть бланк-редактор для редактирования записи картотеки.

В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге [свойств объекта КартотекаШаблона](#) (на странице "Правка").

Пример

```
TemplateCardfile1.CanOpen = CheckBox5.State;
```


Описание

МожноПеремещать :Логическое;

CanMove :Logical;

Назначение

Поле определяет, можно ли интерактивно перемещать записи в картотеке.

В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге [свойств объекта КартотекаШаблона](#) (на странице "Правка").

Пример

```
TemplateCardfile1.CanMove = CheckBox3.State;
```

Описание

МожноРедактировать :Логическое;
CanEdit :Logical;

Назначение

Поле определяет, можно ли интерактивно редактировать значения в клетках картотеки.

В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге [свойств объекта КартотекаШаблона](#) (на странице "Правка").

Пример

```
TemplateCardfile1.CanEdit = CheckBox5.State;
```

Описание

МожноУдалять :Логическое;
CanDelete :Logical;

Назначение

Поле определяет, можно ли интерактивно удалять записи из картотеки.

В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге [свойств объекта КартотекаШаблона](#) (на странице "Правка").

Пример

```
TemplateCardfile1.CanDelete = CheckBox2.State;
```

Описание

ОписаниеИзПоля :Строка;
DescrFromField :String;

Назначение

Поле позволяет узнать или указать имя поля записи, содержимое которого будет использовать для обозначения записей. В частности, это значение (преобразованное в строку) будет выводиться в диалоговых окнах с запросом на подтверждение операцией с записью (например, при удалении: "Валюта: рубли. Удалить текущую запись?" – здесь значение 'рубли' взято из поля описания).

Если оставить поле **ОписаниеИзПоля** пустым, то по умолчанию используется содержимое поля **DocID** записи.

Пример

```
-- процедура RadiolOnChange назначена обработчиком
-- события для двух переключателей 'Имя' и 'Код';
-- в зависимости от положения переключателей,
-- группы в дереве картотеки идентифицируются либо
-- по имени, либо по коду;
-- Caption каждого переключателя соответствует
-- имени поля записи
proc RadiolOnChange(Sender :RadioButton);
    КартотекаШаблон1.ОписаниеИзПоля = Sender.Caption;
end;
```

Описание

ПозТекущегоСтолбца : Целое;
CurrentColumnPos : Integer;

Назначение

Данное свойство позволяет узнать номер текущего (видимого) столбца картотеки, а также сделать текущим другой видимый столбец. *Текущим* (выделенным) считается столбец, в котором находится курсор. Свойство доступно на чтение и запись.

Внимание! Свойство **ПозТекущегоСтолбца**=CurrentColumn.Index, только в том случае, когда видимыми являются все столбцы картотеки. *В общем случае номер видимого столбца ПозТекущегоСтолбца не совпадает с порядковым номером столбца картотеки CurrentColumn.Index.*

Валидность диапазона при установке свойства не проверяется, поэтому, чтобы встать на последний столбец можно написать CurrentColumnPos = 999; или задать общее [количества столбцов](#) (см. пример ниже).

Пример

```
-- выделяем клетку в самом последнем столбце  
TemplateCardfile1.CurrentColumnPos = TemplateCardfile1.ColumnsCount;
```

Описание

Поле ПоискПриНаборе :Логическое;
FindAtType :Logical;

Назначение

Свойство разрешает или запрещает открывать диалог поиска в картотеке шаблона. Если свойство равно True, то в режиме редактирования (inplace-редактор) при нажатии на любую клавишу открывается диалог. Для изменения данных в ячейке необходимо нажать клавишу *Enter*.

Поле доступно на чтение и запись.

Пример

```
var TempCardfile1 :TemplateCardfile;  
...  
TempCardfile1.FindAtType = False;
```

Описание

ПоказыватьИерархию : Логическое;
ShowHierarchy: Logical;

Назначение

Данное поле позволяет узнать, включен ли режим просмотра иерархии в картотеке шаблона, а также включать/отключать его. В иерархической картотеке записи собраны в группы, причем одновременно (значение свойства равно True) в картотеке отображается лишь один уровень иерархии (корень картотеки или содержимое конкретной группы). Если значение свойства равно False, в картотеке выводятся в "плоском" виде записи всех уровней с учетом фильтра.

Пример

```
КартотекаШаблона1 :КартотекаШаблона;  
-- процедура включает/отключает режим иерархического просмотра  
proc ButtonClick(B:Button);  
    КартотекаШаблона1.ПоказыватьИерархию =  
    Not КартотекаШаблона1.ПоказыватьИерархию;  
end;
```

Описание

ПоказыватьИтоги :Логическое;
ShowTotal :Logical;

Назначение

Свойство разрешает или запрещает показывать итоговую строку в картотеке шаблона. Поле доступно на чтение и запись.

Пример

```
var TC1 :TemplateCardfile;  
...  
TC1.ShowTotal = True;
```


Описание

ПоказыватьКоличество :Логическое;
ShowCount :Logical;

Назначение

Свойство позволяет программным способом показывать (True) количество записей в картотеке или запретить показ (False).

Поле доступно на чтение и запись.

Пример

```
var TempCardfile1 :TemplateCardfile;  
...  
TempCardfile1.ShowCount = True;
```

Описание

ПоказыватьУдаленные : Логическое;
ShowDeleted : Logical;

Назначение

Данное поле позволяет узнать, включен ли режим просмотра удаленных записей или нет, а также включать/отключать его.

Пример

```
КартотекаШаблона1 :КартотекаШаблона;  
-- процедура включает/отключает видимость удаленных записей  
proc ButtonClick(S:String);  
    КартотекаШаблона1.ПоказыватьУдаленные =  
        NOT КартотекаШаблона1.ПоказыватьУдаленные;  
end;
```

Описание

Редакторы :Класс [ФормаБланка](#)[];
Editors :Class [BlankForm](#)[];

Назначение

Данное поле предоставляет доступ к массиву классов бланков-редакторов, назначенных для данной картотеки. Массив можно считывать поэлементно и целиком. Поэлементное присвоение новых значений не допускается, однако можно записать в поле целиком новый массив классов. Количество элементов в массиве можно получить с помощью функции [ДлинаМассива](#) класса Система.

Поле позволяет узнать и изменить перечень классов бланков-редакторов картотеки, использующихся для обычных (негрупповых) записей. Для того чтобы узнать или изменить перечень классов бланков-редакторов групповых записей применяется поле [РедакторыГрупп](#).

Пример

```
var n: integer;  
  n = LengthOfArray(TemplateCardfile1.Editors);  
  if n > 1 then  
    TemplateCardfile1.Editors = [ДвижениеСредств];  
  end;
```

Описание

РедакторыГрупп :Класс [ФормаБланка](#)[];
GroupsEditors :Class [BlankForm](#)[];

Назначение

Данное поле предоставляет доступ к массиву классов бланков-редакторов, назначенных для групп данной картотеки. Массив можно считывать поэлементно и целиком. Поэлементное присвоение новых значений не допускается, однако можно записать в поле целиком новый массив классов. Количество элементов в массиве можно получить с помощью функции [ДлинаМассива](#) класса Система.

Поле позволяет узнать и изменить перечень классов бланков-редакторов картотеки, использующихся для *групповых записей*. Для того чтобы узнать или изменить перечень классов бланков-редакторов *обычных (негрупповых) записей* применяется поле [Редакторы](#).

Пример

```
var n: integer;  
  n = LengthOfArray(TemplateCardfile1.GroupsEditors);  
  if n = 0 then  
    TemplateCardfile1.GroupsEditors = [ДвижениеСредств];  
  end;
```

Описание

Столбец[Индекс: Целое] :[СтолбецКартотеки](#);
Column [Index: Integer] :[CardfileColumn](#);

Аргументы

Индекс - задает позицию, в которую будет вставлен новый столбец. Индекс может изменяться в пределах от 1 до [количества столбцов](#).

Назначение

Данное поле позволяет получить доступ к конкретному столбцу картотеки по его номеру. Поле доступно только на чтение.

Пример

```
Card: TemplateCardfile; -- объект шаблона
-- симулируем метод ColumnByField
func FieldByName(Name: String):CardFileColumn;
var i: integer;
  for i=1..Card.ColumnsCount do
    if Name = Card.Column[i].FieldName then
      return Card.Column[i];
    end;
  end;
  return nil;
end;
```

Описание

СтолбецПоПолю[Имя: Строка] : [СтолбецКартотеки](#);
ColumnByField[Name: String] : [CardfileColumn](#);

Аргументы

Имя - имя столбца картотеки.

Назначение

Данное поле позволяет получить доступ к конкретному столбцу картотеки по имени поля, содержащегося в нем. Поле доступно только на чтение.

Пример

```
Card: TemplateCardfile; -- объект шаблона
-- выделить столбец жирным шрифтом
proc BoldByName;
var cc: CardFileColumn;
  cc = Card.ColumnByField["Итого"];
  cc.Font.Bold = TRUE;
end;
```

Описание

ТекущаяГруппа : [Запись](#);
CurrentGroup: [Record](#);

Назначение

Данное свойство позволяет узнать или установить запись, задающую текущую группу в иерархической картотеке шаблона.

В каждый момент времени в таблице иерархической картотеки отображается содержимое лишь одной текущей группы. Переходы из группы в группу осуществляются с помощью навигационных команд, доступных на странице "Бланк" инструментальной панели, или с помощью горячих клавиш.

Пример

```
Card: TemplateCardfile; -- объект шаблона
-- после каждого перемещения курсора по картотеке шаблона
-- проверяем, не является ли запись группой, и если да -
-- переходим в нее
proc CardfileGoods_OnMove(D:Document);
  var x: Справочники.Товары;
  try
    x = Card.Current;
    if x.IsGroup then
      Card.CurrentGroup = x;
    end;
  except
  end;
end;
```

Описание

Текущий : [Документ](#);
Current : [Record](#);
Текущая : [Запись](#);

Назначение

Данное свойство позволяет узнать и установить текущую запись картотеки шаблона.

Пример

```
Card: TemplateCardfile; -- объект шаблона
-- после каждого перемещения курсора по картотеке шаблона
-- выводим DocID текущей записи
proc CardfileFirm_OnMove(D:Document);
  var x: Справочники.ЮрЛицо;
  try
    x = Card.Current;
    hint(x.DocID);
  except
    hint('');
  end;
end;
```


Описание

ТекущийСтолбец : [СтолбецКартотеки](#);
CurrentColumn : [CardfileColumn](#);

Назначение

Поле возвращает ссылку на столбец картотеки, содержащий выделенную клетку. Свойство доступно на чтение и запись.

Внимание! При попытке выделить невидимый столбец или столбец, отсутствующий в картотеке, текущая позиция *не* **меняется**.

Пример

```
TemplateCardfile1 :TemplateCardfile;  
...  
TemplateCardfile1.CurrentColumn.Font.Bold = TRUE;
```

Описание

Упорядочивание : Строка;
Order : String;

Назначение

Позволяет задать или узнать порядок сортировки документов в картотеке. В строке указывается название одного или нескольких полей (разделенных точкой с запятой), по которым должны быть отсортированы документы. После имени каждого поля можно указать символ '+' (плюс) или '-' (минус). Плюс задает сортировку по возрастанию, минус – по убыванию.

Использование этого поля более предпочтительно, чем разыменовывание поля [Запрос](#) и обращение к его свойству **Упорядочивание**, так как все изменения в запросе действуют лишь до первого изменения содержимого картотеки.

Пример

```
Card1 :TemplateCardfile;  
-- по нажатию кнопки меняем порядок сортировки  
proc OnButtonClick(Button1 :Button);  
    Card1.Order = "дата;время;филиал";  
end;
```

Описание

ФиксированныхСтолбцов : Целое;
FixedColumns : Integer;

Назначение

Данное свойство позволяет узнать и установить количество фиксированных столбцов в картотеке. Фиксированные столбцы остаются у левого края картотеки при ее прокрутке по горизонтали.

Пример

```
proc ButtonClick(B:Button);  
    КартотекаШаблон1.ФиксированныхСтолбцов = 1;  
end;
```

Поле Фильтр / Filter

Описание

Фильтр : Строка;
Filter : String;

Назначение

Данное поле позволяет установить и прочесть текущее значение фильтра, задающего условие отбора документов, отображаемых в картотеке шаблона.

Общие правила составления фильтрующего выражения совпадают с правилами [установки пользовательских фильтров](#) в картотеках. Кроме [стандартных функций](#), которые можно использовать в выражениях фильтров, допускается применять [имена служебных полей](#) и так называемые [кванторы](#).

Фильтр, заданный с помощью этого поля, недоступен пользователю в отличие от [пользовательского фильтра](#), который может им меняться интерактивно через диалог.

Пример

```
Card: TemplateCardfile; -- объект шаблона
-- процедура включает/отключает "фильтр по Москве"
proc ButtonClick(S:String);
  if Card.Фильтр = '' then
    Card.Фильтр = 'COOTB(Адрес, "*Москва*")';
  else
    Card.Фильтр = '';
  end;
end;
```

Описание

ФильтрГрупп : [СписокЗаписей](#);

GroupFilter : [RecordList](#);

Назначение

Данное поле позволяет установить и прочесть текущее значение фильтра групп, определяющего условие отбора записей по их расположению в иерархии групп. Иными словами, с помощью данного поля можно указать, элементы каких групп должны отображаться в картотеке шаблона.

Данное поле аналогично одноименному полю [ФильтрГрупп/GroupFilter](#) класса [Картотека/CardFile](#)

Пример

```
Card: TemplateCardfile; -- объект шаблона
proc УстановитьОбзорГруппыТоваров(ГруппаТоваров: Тест.Справочники.Товар);
    Card.ФильтрГрупп.Очистить;
    Card.ФильтрГрупп.Добавить(ГруппаТоваров);
end;
```

Описание

ФильтрПользователя : Строка;
UserFilter : String;

Назначение

Данное поле позволяет установить и прочесть текущее значение пользовательского фильтра, задающего дополнительное условие отбора документов, отображаемых в картотеке. Пользовательский фильтр применяется к набору документов, отвечающих условию общего фильтра (свойство [Фильтр](#)). Пользовательский фильтр вступает в действие, только когда поле [ИспользоватьФильтрПользователя](#) содержит значение TRUE.

В отличие от общего фильтра пользовательский фильтр может быть интерактивно изменен пользователем с помощью [диалога "Установка фильтра"](#).

Пример

```
-- процедура включает/отключает "фильтр по Москве"
proc ButtonClick(S:String);
  if КартотекаШаблона1.ФильтрПользователя = '' then
    КартотекаШаблона1.ФильтрПользователя = 'COOTB(Адрес, "*Москва*")';
    КартотекаШаблона1.ИспользоватьФильтрПользователя = TRUE;
  else
    Картотека.ФильтрПользователя = '';
    -- автоматически сбрасывает ИспользоватьФильтрПользователя
  end;
end;
```

Поле Цвет / Color

Описание

Цвет : Целое;
Color :Integer;

Назначение

Данное поле позволяет задать цвет фона объекта КартотекаШаблона.

Пример

```
var Red, Green, Blue :Integer;  
Red = 128;  
Green = 128;  
Blue = 128;  
КартотекаШаблона1.Color = Red + Green*256 + Blue*256*256;
```

Поле Шрифт / Font

Описание

Шрифт : [Шрифт](#) ;

Font : [Font](#) ;

Назначение

Позволяет узнать параметры шрифта объекта КартотекаШаблона.

Поле доступно только на чтение.

Пример

```
КартотекаШаблона1.Шрифт.Жирный = Истина;
```


Описание

Восстановить;
Undelete;

Назначение

Восстанавливает все выделенные записи. Если выделенных записей нет, восстанавливает текущую запись. Если нет ни выделенных, ни текущей записей, происходит ошибка.

Восстанавливаемые записи должны быть помечены как удаленные. В противном случае также возникает ошибка. Иными словами, нельзя восстановить неудаленную запись.

Удаленные записи видны в окне картотеки только при ее соответствующей настройке.

Пример

```
proc ButtonUnDelOnClick(Sender :Кнопка);  
  try  
    TemplateCardfile1.UnDelete;  
  except  
    -- ...  
  end;  
end;
```

Описание

```
ВыделитьЗаписи([Записи : Запись\[\] ]);  
SelectRecords([Records : Record\[\] ]);
```

Аргументы

Записи - необязательный параметр, массив записей, которые должны быть выделены.

Назначение

Выделяет массив записей, заданный в первом параметре, или все записи, если параметр опущен.

Пример

```
var локКартотека           :TemplateCardfile;  
var локКолВоОбработанных  :Integer;  
var локОбработанныеПроцессы :Данные.Процесс[];  
  
proc ВыделениеКартотеки;  
  if локОбработанныеПроцессы <> nil then  
    локКартотека.SelectRecords(локОбработанныеПроцессы);  
  end;  
end;  
end;
```

Описание

```
ЗавершитьРедактирование[ (ПринятьИзменения :Логическое) ] ;  
EndEdit[ (AcceptChanges: Logical) ] ;
```

Аргументы

ПринятьИзменения - определяет, следует ли принять или отклонить сделанные изменения.

Назначение

Закрывает встроенный (in-place) редактор, открытый ранее с помощью [BeginEdit](#) или пользователем. Если **ПринятьИзменения** равно TRUE, изменения (если они были) передаются в обработчик **OnVerify** соответствующего столбца картотеки (если этот обработчик установлен) и затем сохраняются во временном представлении текущей записи. Впоследствии, изменения будут окончательно приняты, если будет вызван метод [Post](#), или отменены, если будет вызван метод [Cancel](#) для этой записи (объекта **Document**).

Если **ПринятьИзменения** равно FALSE, изменения не попадают из встроенного редактора во временный буфер, где хранится текущая редактируемая запись. Если параметр **ПринятьИзменения** опущен, он считается равным TRUE.

Пример

```
proc ButCancelClick(S:String);  
    TemplateCardfile1.EndEdit(FALSE);  
end;
```

Описание

```
НачатьРедактирование;  
BeginEdit;
```

Назначение

Открывает в выделенной клетке встроенный (in-place) редактор, позволяющий изменить значение в ней.
Редактирование возможно только в том случае, если в настройках картотеки разрешена ее модификация.

Пример

```
-- обработчик события "щелчок мышью в колонке картотеки"  
func TempCardfile_OnClick(Sender :TemplateCardFile; Action :Integer; Column :CardfileColumn; Document :Document):Logical;  
  if Action>0 then  
    Sender.BeginEdit; -- приступить к редактированию клетки  
  end;  
  return Ложь;  
end;
```

Описание

```
Обновить[(Только1строка:Логическое)];  
Update[(Only1row:Logical)];
```

Аргументы

Только1строка - логическое выражение, которое определяет, следует ли обновить только одну текущую строку или все окно картотеки.

Назначение

Иницирует перерисовку окна картотеки, если параметр равен FALSE, или только одной текущей строки картотеки, в противном случае. Если параметр опущен, то он по умолчанию принимается равным FALSE, то есть обновляется все окно целиком.

Данная процедура может быть полезна при выполнении длительных расчетов, влияющих на содержимое картотеки и когда требуется визуально отображать ход процесса. Кроме того, обновление может потребоваться, если внешний вид некоторых вычисляемых полей зависит не только от их значений, но и внешних факторов. Дело в том, что картотека автоматически перерисовывается только при изменении значений в ее полях и не может реагировать на какие-либо другие события вне ее, даже если они по логике программы связаны с картотечкой.

Пример

```
proc OnEditChange(E1:Edit);  
  -- при изменении в редакторе обновляем строку картотеки  
  TemplateCarfile1.Update(TRUE);  
end;
```

Процедура СнятьВыделениеЗаписей / DeselectRecords

Описание

```
СнятьВыделениеЗаписей([Записи : Запись\[\] ]);  
DeselectRecords([Records : Record\[\] ]);
```

Аргументы

Записи - необязательный параметр, массив записей, у которых требуется снять выделение.

Назначение

Снимает выделение массива записей, заданных в первом параметре, или всех записей, если параметр опущен. Записи могут выделены с помощью процедуры [ВыделитьЗаписи](#).

Пример

```
var локКартотека           :TemplateCardfile;  
var локКолВоОбработанных  :Integer;  
var локОбработанныеПроцессы :Данные.Процесс[];  
  
proc СнятьВыделениеКартотеки;  
    локКартотека.DeselectRecords(локОбработанныеПроцессы);  
    локОбработанныеПроцессы = nil;  
end;
```

Процедура Удалить / Delete

Описание

Удалить;
Delete;

Назначение

Удаляет все выделенные записи. Если записи не выделены, удаляет текущую запись. Если нет ни выделенных, ни текущей записей, происходит ошибка.

Удаляемые записи не должны быть помеченными как удаленные. В противном случае также возникает ошибка. Иными словами, нельзя повторно удалить уже удаленную запись.

Пример

```
proc ButtonDelOnClick(Sender :Кнопка);  
  try  
    TemplateCardfile1.Delete;  
  except  
    -- ...  
  end;  
end;
```

Описание

```
УдалитьСтолбец(Номер :Целое);  
DeleteColumn(Position :Integer);
```

Аргументы

Номер - задает номер столбца, который следует удалить. Номер должен лежать в пределах от 1 до [количества столбцов](#).

Назначение

Процедура удаляет указанный столбец из картотеки.

Пример

```
Card: TemplateCardfile; -- объект шаблона  
-- ...  
-- удаляем первый столбец  
Card.DeleteColumn(1);
```


Описание

ПередИзменением: Строка;
BeforeModify: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит перед началом изменения одной или нескольких выделенных записей. Если какое-либо действие выполняется сразу над группой записей, то событие **ПередИзменением** происходит лишь один раз – при этом перечень выделенных записей можно получить из обработчика, обратившись к полю [Выделенные](#).

Обработчик

```
func BeforeModify(Sender :TemplateCardfile; Action :Cardfile.ModifyActions; TheRecord :Record; TheGroup :Record; var AskConfirm :Logical) :Logical;
```

Параметры:

Sender - указатель на объект класса **КартотекаШаблона**, с которым произошло событие;

Action - действие, которое производится над записями картотеки (удаление, восстановление, перемещение или копирование);

TheRecord - ссылка на запись, над которой производится действие, если действие выполняет сразу над несколькими записями, параметр **TheRecord** равен nil;

TheGroup - целевая группа, в которую переносится запись или набор записей в случае перемещения или копирования;

AskConfirm - логический параметр, позволяет указать системе, следует ли запрашивать подтверждения у пользователя. Если данный параметр при выходе из обработчика равен TRUE, то пользователю выдается запрос, а в случае FALSE - нет. По умолчанию (при входе в обработчик) AskConfirm равно TRUE.

Если обработчик возвращает TRUE, то тем самым разрешается выполнение действия и будут сгенерированы все последующие события - в первую очередь [ПриИзменении](#) (это событие будет генерироваться для каждой записи из общего числа выделенных). Если обработчик **ПередИзменением** вернет FALSE, выполнение действия над записями картотеки прекращается.

При входе в обработчик Result равен FALSE (действия запрещены).

Пример

```
func Card_OnBeforeModify(Sender :TemplateCardfile; Action :Cardfile.ModifyActions; TheRecord :Record; TheGroup :Record; var AskConfirm :Logical) :Logical;  
  if Action = Cardfile.CopyRecord then  
    -- копирование запрещено  
    return FALSE;  
  elseif Action = Cardfile.DeleteRecord then  
    -- удаление требует подтверждения  
    AskConfirm = TRUE;  
  else  
    -- остальные действия выполнять без подтверждений  
    AskConfirm = FALSE;  
  end;  
end;
```

Событие ПриВходе / OnEnter

Описание

ПриВходе: Строка;
OnEnter: String;

Назначение

Данное событие происходит, когда картотека шаблона получает фокус ввода (то есть, становится текущим элементом управления на шаблоне). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB* или по щелчку мыши. Поле **ПриВходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnEnter**(TC :TemplateCardfile);
TC - объект класса **КартотекаШаблона**, с которым произошло событие.

Пример

```
proc КартотекаШаблона_ПриВходе(TC :TemplateCardfile);  
    TC.Font.Bold = TRUE;  
end;
```

Описание

ПриВыходе: Строка;
OnExit: String;

Назначение

Данное событие происходит, когда картотека шаблона теряет фокус ввода (то есть, перестает быть текущим элементом управления). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB* или по щелчку мыши. Поле **ПриВыходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnExit**(TC :TemplateCardfile);
TC - объект класса **КартотекаШаблона**, с которым произошло событие.

Пример

```
proc КартотекаШаблона_ПриВыходе(TC :TemplateCardfile);  
    TC.Font.Bold = FALSE;  
end;
```

Описание

ПриЗаписи: Строка;
OnPost: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при попытке сохранить измененную запись картотеки.

Обработчик

```
func OnPost(Sender :TemplateCardfile; ARec: Document) :Logical;
```

Параметры:

Sender - указатель на объект класса **КартотекаШаблона**, с которым произошло событие;
ARec - редактируемая запись (документ).

Событие позволяет контролировать запись новых значений в поля документа, указанного с помощью параметра **ARec**. Если обработчик возвращает TRUE, документ сохраняется, иначе – нет.

Пример

```
func ПриЗаписи (Табл1 :КартотекаШаблона; ARec :Документ) :Логическое;  
  if Табл1.Имя = "ТаблицаОтгрузок" then  
    if (ARec as Операции.Отгрузка).Номер = 0 then  
      Hint("Укажите номер документа");  
      return FALSE;  
    end;  
  end;  
  return TRUE;  
end;
```

Описание

ПриИзменении: Строка;
OnModify: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит перед началом изменения конкретной записи. Это событие генерируется сразу после события [ПередИзменением](#), причем если определен обработчик события **ПередИзменением**, то в нем должно быть разрешено проведение действия (функция-обработчик **ПередИзменением** должна вернуть TRUE).

Если какое-либо действие выполняется сразу над группой записей, то событие **ПриИзменении** происходит для каждой записи отдельно.

Обработчик

func **OnModify**(Sender :TemplateCardfile; Action :Cardfile.ModifyActions; TheRecord :Record; TheGroup :Record) :Logical;

Параметры:

Sender - указатель на объект класса **КартотекаШаблона**, с которым произошло событие;

Action - действие, которое производится над записью картотеки (удаление, восстановление, перемещение или копирование);

TheRecord - ссылка на запись, над которой производится действие;

TheGroup - целевая группа, в которую переносится запись в случае перемещения или копирования. При копировании записи параметр **TheRecord** содержит не исходную запись, а её копию, причем в этой копии еще не заполнено поле **GroupDoc** (значения всех остальных полей соответствуют исходной записи).

Если обработчик возвращает TRUE, система выполняет стандартную обработку выполняемого действия. Если обработчик возвращает FALSE, система не выполняет стандартную обработку, однако программист может закодировать непосредственно в теле обработчика все необходимые операции (в том числе, непредусмотренные стандартной обработкой).

В частности, обработчик может исправить часть полей записи (например, чтобы избежать конфликтов уникальных полей) и вернуть TRUE. Либо он может самостоятельно записать запись и вернуть FALSE. Если обработчик вернет FALSE, не записав запись, это будет означать, что запись не будет скопирована. При этом, если копировавшаяся запись является групповой, то не копируются и все её элементы.

При входе в обработчик Result равен FALSE (действия запрещены).

Пример

```
func Card_OnModify(Sender: TemplateCardfile; Action :Cardfile.ModifyActions; TheRecord :Record; TheGroup :Record) :Logical;  
  if Action = Cardfile.UndeleteRecord then  
    -- восстановление запрещено  
    return FALSE;  
  elseif Action = Cardfile.CopyRecord then  
    -- изменяем значения некоторых полей  
    TheRecord.Name = "<пусто>";  
    -- соглашаемся разместить запись в группе,  
    -- куда её копирует пользователь  
    TheRecord.GroupDoc = TheGroup;  
    -- сохраняем запись  
    TheRecord.Post;  
    -- говорим системе, что дальнейшая  
    -- автоматическая обработка события не требуется  
    return FALSE;  
  end;  
  return TRUE;  
end;
```

Описание

ПриНажатии: Строка;
OnClick: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при одиночном или двойном щелчке мышью в ячейке картотеки на шаблоне или при нажатии клавиши **Enter**, когда фокус ввода находится в этой картотеке.

Обработчик

```
func OnClick(Sender :TemplateCardfile; Action : Template.ClickTypes; Column :CardfileColumn;  
Rec :Record) :Logical;
```

Параметры:

Sender - указатель на объект класса **КартотекаШаблона**, с которым произошло событие;
Action - одна из предопределенных констант, описанных в типе [Шаблон.ТипыНажатия](#), которая определяет действие пользователя:

- 0** - одинарный щелчок мышью;
- 1** - двойной щелчок мышью;
- 2** - нажатие клавиши **Enter**;

Column - текущий столбец картотеки;

Rec - текущий документ (запись).

Если функция возвращает True, происходит стандартная обработка события, то есть открытие бланка-редактора, открытие встроенного (in-place) редактора, вход в группу или выбор текущего документа, в зависимости от настроек картотеки. Если функция возвращает False, стандартная обработка не выполняется.

Пример

```
-- обработчик события "щелчок мышью в колонке картотеки"  
func Card_OnClick(TC :TemplateCardfile; Action :Integer;  
Column :CardfileColumn; Document :Document):Logical;  
  if Action = 0 then  
    Hint(Document.DocID);  
  fi;  
  return Ложь;  
end;
```

Описание

ПриОткрытииБланка: Строка;
OnOpenBlank: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит перед открытием бланка-редактора для текущей записи картотеки.

Обработчик

```
func OnOpenBlank(TC :TemplateCardfile; Action :Integer; Rec :Record) :Logical;
```

Параметры:

TC - объект класса **КартотекаШаблона**, с которым произошло событие;

Action - определяет тип события:

0 - редактируется существующий документ;

1 - добавляется новый документ;

2 - добавляется копия документа (документ клонируется).

Rec - редактируемый документ (запись).

Если функция возвращает TRUE, происходит стандартная обработка события, то есть открытие бланка-редактора. Если функция возвращает FALSE, стандартная обработка не выполняется.

Пример

```
func Card_OnOpenBlank(TC :TemplateCardfile;  
Action :Integer; Rec :Record):Logical;  
  if Action=0 then  
    return TRUE;  
  else  
    message('Для создания новых документов '+  
    'воспользуйтесь командами Добавить (Ins) или Дублировать');  
  fi;  
  return FALSE;  
end;
```

Описание

ПриОтмене: Строка;
OnCancel: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при попытке отменить сделанные в картотечной записи изменения и вернуть ее в первоначальное состояние.

Обработчик

```
func OnCancel(Sender :TemplateCardfile; ARec :Document) :Logical;
```

Параметры:

Sender - указатель на объект класса **КартотекаШаблона**, с которым произошло событие;
ARec - исходный документ (запись).

Событие позволяет контролировать "откат" к старому состоянию записи. Если обработчик возвращает TRUE, запись, заданная с помощью параметра **ARec**, возвращается в исходное состояние; если функция возвращает FALSE – ничего не происходит.

Пример

```
func ПриОтмене(Отправитель :КартотекаШаблона; Операция :Операции.Оплата) :Логическое;  
-- если в новых записях какие-то поля проставляются  
-- программой по умолчанию в момент их создания,  
-- то при "сбросе" такой записи  
-- нужно повторить инициализацию  
if Операция.НовыйДокумент then  
    Операция.УстановитьЗначенияПоУмолчанию; -- прикладная процедура  
end;  
return TRUE;  
end;
```


Описание

ПриОформлении: Строка;
OnRearrange: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при изменении атрибутов столбца пользователем. Системой отслеживаются изменения ширины, положения, видимости и сортировки.

Обработчик

proc **OnRearrange**(Sender :TemplateCardfile; Action : [Cardfile.RearrangeActions](#); Column :CardfileColumn);

Параметры:

Sender - указатель на объект класса **КартотекаШаблона**, с которым произошло событие;

Action - параметр, детализирующий, какой именно атрибут столбца был изменен и принимающий одно из значений перечислимого типа [RearrangeActions](#);

Column - указатель на объект **СтолбецКартотеки**, принадлежащего объекту **Sender** из класса **КартотекаШаблона**.

Внимание. Если параметр Action=Cardfile.OtherRearrangeActions, параметр Column = nil. В этом случае событие может вызываться, например, при изменении ширины панелей или включении/выключении видимости удаленных записей.

Пример

```
proc Card_OnRearrange(Sender :TemplateCardfile;  
Action :Cardfile.RearrangeActions; Column :CardfileColumn);  
-- в случае сортировки по какому-либо столбцу  
if Action = Cardfile.Sort then  
-- вызываем прикладную функцию для сортировки  
-- служебных массивов  
SortAuxArrays(Column);  
end;  
end;
```

Описание

ПриПеремещении: Строка;
OnMove: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при перемещении текстового курсора в картотеке от одной записи к другой.

Обработчик

```
proc OnMove(Sender :TemplateCardfile; D: Document);
```

Параметры:

Sender - указатель на объект класса **КартотекаШаблона**, с которым произошло событие;
D - текущая запись.

Событие позволяет управлять состоянием и значением полей записи картотеки в зависимости от контекста ввода.

В некоторых случаях, например, при переходе из группы в группу, может возникнуть ситуация, когда ни один документ не выделен. Тогда параметр **D** будет равен nil. Поэтому в обработчике данного события всегда требуется проверять **D** на ноль, прежде чем обращаться к какому-либо свойству документа.

Пример

```
proc (TC :TemplateCardfile);  
Пример  
proc Card_OnMove(TC :TemplateCardfile; D:Document);  
  if D <> nil then  
    hint(D.Номер);  
  end;  
end;
```

Описание

ПриРедактировании: Строка;
OnEdit: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит в момент начала редактирования записи в картотеке, минуя бланк-редактор. Иными словами, событие происходит после того, как пользователь вызвал команду редактирования или вставки записи (например, нажал кнопку Ins), но до того, как началось ее редактирование. Событие позволяет проинициализировать требуемые поля записи или вообще запретить ее создание.

Обработчик

```
func OnEdit(Sender :TemplateCardfile; Action :Integer; ARec :Record): Logical;
```

Параметры:

Sender - указатель на объект класса **КартотекаШаблона**, с которым произошло событие;

Action - выполняемое действие:

0 - редактируется имеющаяся запись;

1 - добавляется новая запись;

2 - добавляется копия записи (запись клонируется);

ARec - добавляемая запись.

Если функция возвращает True, происходит стандартная обработка (переход в режим редактирования или вставка новой записи) и позиционирование на нее. Если функция возвращает False, редактирование не начинается (в случае попытки добавить новую запись, она не добавляется), то есть для новой записи автоматически делается [Cancel](#) (так же как и при событии [OnOpenBlank](#)).

Необходимо отметить, что запись в момент наступления события еще не вставлена в Запрос картотеки (хотя и отображается в окне картотеки) – это происходит только после завершения редактирования, то есть после вызова метода [Post](#).

Пример

```
func CardfileOrders_OnEdit(Sender :TemplateCardfile; Kind :Integer; D :Document):Logical;  
  if Kind = 1 then  
    СквознойНомер = СквознойНомер + 1  
    D.Номер = СквознойНомер;  
  end;  
  return true;  
end;
```

Описание

ПриРисованииСтроки : Строка;
OnDrawRow : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит перед изменением цвета иконки картотеки.

Обработчик

проц **OnDrawRow**(Record :Record; Selected :Logical; var Color :Integer; Image :Image);

Параметры:

Record - указатель на редактируемую запись (документ) объекта КартотекаШаблона;
Selected - логический параметр. Если он равен True, запись картотеки (строка) выделена;
Color - цвет, который будет иметь иконка после срабатывания процедуры-обработчика;
Image - параметр не используется.

Пример

```
проц картотека_ПриРисованииСтроки(Record :Справочники.СтатусДвижения;  
Selected :Logical; var Color :Integer; Image :Image);  
    Color = Record.Цвет;  
end;
```

Описание

ПриСменеВыделения: Строка;
OnChangeSelected: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при изменении выделения нескольких документов.

Обработчик

```
proc OnChangeSelected(Sender :TemplateCardfile; Count :Integer);
```

Параметры:

Sender - указатель на объект класса **КартотекаШаблона**, с которым произошло событие;
Count - количество выделенных документов.

Когда выделение снимается, **Count** равен нулю.

Количество выделенных документов и ссылки на сами документы можно получить соответственно с помощью свойств [КоличествоВыделенных](#) и [Выделенные](#).

Пример

```
proc CardfileOrders_OnChangeSelected(Sender :TemplateCardfile; Count : Integer);  
  hint("Выделено:" + Str(Count));  
end;
```

Описание

ПриСменеГруппы: Строка;
OnChangeGroup: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при переходе из одной группы в другую.

Обработчик

```
func OnChangeGroup(Sender :TemplateCardfile; Group :Record) :Logical;
```

Параметры:

Sender - указатель на объект класса **КартотекаШаблона**, с которым произошло событие;
Group - ссылку на запись той группы, в которую осуществляется переход.

Если обработчик возвращает True – выполняется стандартная обработка, если False – ничего не происходит. Подразумевается, что в последнем случае смену текущей группы, если это разрешено алгоритмом, сделает сам обработчик.

Внимание! В обработчике **OnChangeGroup** нельзя менять текущие группы через свойство картотеки **TemplateCardfile.CurrentGroup**, так как при этом обработчик вызовется снова и произойдет заикливание. Для смены группы программным образом следует использовать свойство **CurrentGroup** из запроса картотеки (см. пример).

Пример

```
func Card_OnChangeGroup(Sender :TemplateCardfile; Group :Record) :Logical;  
  -- Действия до смены группы  
  --...  
  Sender.Query.CurrentGroup = Group;  
  -- Действия после смены группы  
  --...  
  return False;  
end;
```

Описание

ПриСозданииЗаписи: Строка;
OnCreateRecord: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при создании записи в картотеке шаблона.

Данное событие возникает только при добавлении новой записи в картотеку, но не при дублировании старой. Оно предназначено для управления созданием новой записи и позволяет, например, автоматизировать добавление новой записи в гетерогенных картотеках. Там, как известно, при попытке добавить запись открывается стандартный диалог выбора класса добавляемой записи, что не всегда удобно.

Обработчик

```
func OnCreateRecord(Sender :TemplateCardfile; var ADoc :Record) :Logical;
```

Параметры:

Sender - указатель на объект класса **КартотекаШаблона**, с которым произошло событие;

ADoc - выходной параметр для создаваемой записи.

Обработчик должен вернуть FALSE в случае отказа от дальнейшей стандартной обработки и TRUE, если ее нужно продолжить. При входе в обработчик параметр **ADoc** равен *nil*. Если **ADoc** остался равен *nil* по окончании работы обработчика и возвращаемое значение равно TRUE, то произойдет стандартный выбор класса записи и дальнейшая обработка. В случае, когда **ADoc** не равен *nil* и результат равен TRUE, стандартный диалог выбора класса записи не выводится на экран, однако дальнейшая обработка будет продолжена (генерируются события [ПриОткрытииБланка](#) или [ПриРедактировании](#), в зависимости от способа редактирования записи, и т.д.).

Пример

```
-- На шаблоне размещен выпадающий список
-- ComboBoxClassName с именами классов записей;
-- При добавлении новой записи в картотеку, программа
-- автоматически выбирает класс, выделенный в списке.
func TemplateCardfile_OnCreateRecord (Card: TemplateCardfile; var ADoc :Record) :Logical;
begin
    try
        ADoc = FindClass(ComboBoxClassName.Text).Create;
    except
    end;
    return TRUE;
end;
```

Описание

ВставитьСтолбец(Номер :Целое) :[СтолбецКартотеки](#);
InsertColumn(Position :Integer) :[CardfileColumn](#);

Аргументы

Номер - задает позицию, в которую будет вставлен новый столбец. Номер должен лежать в пределах от 1 до [количества столбцов](#).

Назначение

Функция вставляет новый столбец в картотеку. Позиция нового столбца задается аргументом **Номер**.

Пример

```
Card: TemplateCardfile; -- объект шаблона
proc PlusColumn1(FieldName :String);
var Column: CardfileColumn;
    -- создаем столбец, который должен располагаться первым
    Column = Card.InsertColumn(1);
    -- настраиваем столбец
    Column.FieldName = FieldName;
    Column.Width = 200;
    -- ...
end;
```


Описание

ДобавитьСтолбец : [СтолбецКартотеки](#);
AddColumn : [CardfileColumn](#);

Назначение

Функция добавляет новый столбец в картотеку. Столбец становится последним в массиве столбцов.

Пример

```
Card: TemplateCardfile; -- объект шаблона
proc PlusColumn(FieldName :String);
var Column: CardfileColumn;
  -- создаем столбец и получаем ссылку на новый объект
  Column = Card.AddColumn;
  -- настраиваем столбец
  Column.FieldName = FieldName;
  Column.Width = 200;
  -- ...
end;
```









Кнопка / Button





Класс *Кнопка/Button*, производный от класса *ОбъектШаблона*, представляет собой программный интерфейс для объекта *Кнопка*, используемого в шаблонах.

В классе *Кнопка* можно использовать все свойства и методы, определенные в родительских классах [Объект](#) и [ОбъектШаблона](#).

Внимание! Создать объект класса *Кнопка/Button* из программы, написанной на языке ТБ.Скрипт, нельзя. Кнопки создаются самой системой на основе информации о шаблоне, спроектированном в визуальном редакторе шаблонов.

Непосредственно в классе *Кнопка/Button* определены следующие свойства:

-  [Поле Надпись / Caption](#)
-  [Поле Шрифт / Font](#)
-  [Поле Состояние / State](#)
-  [Поле Выравнивание / Alignment](#)
-  [Поле ПоложениеИзображения / ImageAlign](#)
-  [Поле ТипКнопки / ButtonKind](#)
-  [Поле ИспользоватьМакросы / UseMacro](#)
-  [Поле Изображение / Image](#)

-  [Событие ПриВходе / OnEnter](#)
-  [Событие ПриВыходе / OnExit](#)
-  [Событие ПриНажатии / OnClick](#)
-  [Событие ПриИзменении / OnChange](#)

и перечислимые типы:

-  [Тип ВидыКнопок / ButtonKinds](#)

Константы видов кнопок ([ВидыКнопок](#) / [ButtonKinds](#))

Перечислимый тип *ВидыКнопок* / *ButtonKinds*, определенный в классе *Кнопка*, используется в дизайнерских целях, чтобы изменить внешний вид кнопки. Независимо от вида кнопки для генерации событий в программе с помощью объекта *Кнопка* осуществляется одной и той же процедурой-обработчиком **ПриНажатии**.

В данном типе определены следующие константы:

- **КакКнопка** / **asButton** - традиционное изображение кнопки;
- **КакСсылка** / **asLink** - кнопка без рамки.

Константы, определенные в данном типе, используются в свойстве [ТипКнопки](#) класса *Кнопка*.

Описание

Выравнивание : [Шаблон.ТипыВыравнивания](#);
Alignment : [Template.AlignmentTypes](#);

Назначение

Поле позволяет узнать и изменить выравнивание надписи и рисунка (если он задан), выводимых на кнопке.

Пример

```
-- по нажатию кнопки меняем выравнивание текста на ней
-- с левого на правое и обратно
proc Button1OnClick(Sender :Button);
  if Sender.Alignment = Template.LeftAlign then
    Sender.Alignment = Template.RightAlign;
  elsif Sender.Alignment = Template.RightAlign then
    Sender.Alignment = Template.LeftAlign;
  end;
end;
```

Описание

Изображение : [Изображение](#);

Image : [Image](#);

Назначение

Поле позволяет получить ссылку на объект с рисунком, выводимым на кнопке. Назначение рисунка кнопке производится в диалоге свойств кнопки во время проектирования шаблона. Кнопка может одновременно содержать и надпись, и рисунок – в этом случае может потребоваться добавить в текст надписи несколько пробелов для визуального разнесения надписи и рисунка.

В качестве рисунка допустимо использовать файлы форматов JPEG, BMP, ICO, EMF, WMF. Картинка масштабируется (кроме иконок) и выравнивается в соответствии со значением поля **Выравнивание**.

Пример

```
Рисунок1 :Рисунок;
```

```
-- по нажатию одной из нескольких кнопок копируем изображение
-- её рисунка в объект Рисунок1
proc Button1OnClick(Sender :Button);
    Рисунок1.Изображение.Присвоить(Sender.Image);
end;
```

Описание

ИспользоватьМакросы :Логическое;
UseMacro :Logical;

Назначение

Свойство определяет, разрешено ли на кнопках шаблона использовать [макросы](#). Если значение свойства равно True, то любая последовательность символов, отвечающая синтаксису записи макросов, будет интерпретироваться системой как макрос, т.е. изменять способ отображения сопутствующего текста.

Если значение свойства равно False, весь текст выводится непосредственно в том виде, в каком он представлен на кнопке.

Пример

```
Button1 : Button;  
-- ...  
Button1.UseMacro = False;
```

Описание

Надпись :Строка;
Caption :String;

Назначение

Данное поле позволяет узнать и изменить текст, который выводится на кнопке.

Поле доступно на чтение и запись.

Пример

```
Кнопка1 : Button;  
proc P1(Object:String);  
    Кнопка1.Caption = "Перерасчет";  
end;
```

Описание

ПоложениеИзображения : [Шаблон.ТипыРасположения](#);
ImageAlign [Template.AlignTypes](#);

Назначение

Свойство позволяет узнать и изменить местоположение рисунка (вверху, внизу, слева, справа) относительно надписи, выводимой на кнопке. Поле может принимать одно из predetermined значений, установленных в перечислимом типе [ТипыРасположения](#) класса *Шаблон*.

Пример

```
Button1 : Button;  
-- ...  
Button1.ImageAlign = Template.AlignTop;
```


Описание

Состояние :Логическое;
State :Logical;

Назначение

Позволяет использовать кнопку как флаг, переключающийся между "нажатым" и "отжатым" состоянием. Значение FALSE соответствует ненажатому состоянию (на экране кнопка отображается как обычно), значение TRUE – нажатому (поверхность кнопки "осветляется" точечным шахматным узором). Система может автоматически переключать состояние кнопки из одного состояния в другое по нажатию на кнопку, если в свойствах последней включить флаг **Работает как флаг**.

Следует иметь в виду, что присвоение свойству **Состояние** нового значения вызывает генерацию события [ПриИзменении](#).

Пример

```
-- Пример 1
-- Пример программного переключения кнопки из
-- отжатого в нажатое состояние и наоборот
-- Внимание! В свойствах кнопки Button1 флаг
-- "Работает как флаг" должен быть сброшен!
proc Button1OnClick(Sender :Button);
    Sender.State = NOT Sender.State;
    trace(Sender.State);
end;

-- Пример 2
-- Пример использования автоматического переключения кнопки
-- из отжатого в нажатое состояние и наоборот.
-- Внимание! В свойствах кнопки Button2 флаг
-- "Работает как флаг" должен быть включен!
proc Button2OnClick(Sender :Button);
    -- система автоматически уже изменила состояние кнопки
    -- в ответ на нажатие пользователем
    trace(Sender.State);
end;
```

Описание

ТипКнопки; [Кнопка.ВидыКнопок](#);
ButtonKind; [Button.ButtonKinds](#);

Назначение

Свойство позволяет узнать и изменить внешний вид изображения кнопки на шаблоне. В ряде случаев возникают ситуации, когда в дизайнерских целях требуется изменить традиционный вид кнопки. Поле может принимать одно из предопределенных значений, установленных в перечислимом типе [ВидыКнопок](#).

Пример

```
Button1 : Button;  
-- ...  
Button1.ButtonKind = Button.asLink;
```

Поле Шрифт / Font

Описание

Шрифт : [Шрифт](#) ;

Font : [Font](#) ;

Назначение

Позволяет узнать параметры шрифта объекта Кнопка. Поле доступно только на чтение.

Пример

```
Кнопка1.Шрифт.Жирный = Истина;
```

Описание

ПриВходе: Строка;
OnEnter: String;

Назначение

Данное событие происходит, когда кнопка получает фокус ввода (то есть, становится текущим элементом управления на шаблоне). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *Tab*. Поле **ПриВходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnEnter**(B: Button);
B - объект класса **Кнопка**, с которым произошло событие.

Пример

```
proc Кнопка_ПриВходе(B1: Button);  
  if B1.Name = "Расчет" then  
    СтрокаСподсказкой = "Нажимайте на кнопку осторожно!";  
  end;  
end;
```

Описание

```
ПриВыходе: Строка;  
OnExit: String;
```

Назначение

Данное событие происходит, когда кнопка теряет фокус ввода (то есть, кнопка перестает быть текущим элементом управления). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *Tab*. Поле **ПриВыходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

```
proc OnExit(B: Button);  
B - объект класса Кнопка, с которым произошло событие.
```

Пример

```
proc Кнопка_ПриВыходе(B1: Button);  
  if B1.Name = "Расчет" then  
    СтрокаПодсказкой = "";  
    -- удаляем подсказку для данной кнопки  
  end;  
end;
```

Описание

ПриИзменении: Строка;
OnChange: String;

Назначение

Данное событие происходит при изменении состояния кнопки (поле [Состояние](#)).

Изменение может быть как результатом нажатия кнопки пользователем (при условии, что кнопка используется как флаг), так и результатом программной установки поля **Состояние**.

Если событие происходит в кнопке, работающей как флаг, и вызвано действием пользователя (щелчок мышью или нажатие клавиши *Пробел*), то оно предваряет событие [ПриНажатии](#) (иными словами, система автоматически меняет состояние кнопки, "сообщает" об этом с помощью события **ПриИзменении**, и лишь затем сигнализирует, что кнопка была нажата). В момент вызова обработчика события состояние кнопки уже имеет новое значение.

Поле **ПриИзменении** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnChange**(B: Button);
B - объект класса **Кнопка**, с которым произошло событие.

Пример

```
proc Кнопка1_ПриИзменении(Кн1: Кнопка);  
  trace(Кн1.Состояние);  
  if Кн1.Состояние then  
    Кн1.Надпись = "Нажата";  
  else  
    Кн1.Надпись = "Отжата";  
  end;  
end;
```

Описание

ПриНажатии: Строка;
OnClick: String;

Назначение

Данное событие происходит при нажатии кнопки. Если кнопка используется как флаг, то событие происходит после события [ПриИзменении](#).

Поле **ПриНажатии** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnClick**(B: Button);
B - объект класса Кнопка, с которым произошло событие.

Пример


```
proc Кнопка1_ПриНажатии(B1: Button);  
  B1.Visible = FALSE;  
  -- в ответ на нажатие кнопка становится невидимой.  
end;
```

Класс *Надпись/Label*, производный от класса *ОбъектШаблона*, представляет собой программный интерфейс для элемента управления типа "статический текст", используемого в шаблонах Студии.

Внимание! Создать объект класса *Надпись* из программы на ТБ.Скрипт нельзя. Надписи создаются самой системой на основе информации о шаблоне, спроектированном в визуальном редакторе бланков Студии.

В классе *Надпись* можно использовать все свойства и методы, определенные в родительских классах [Объект](#) и [ОбъектШаблона](#).

Непосредственно в классе *Надпись/Label* определены следующие свойства:

-  [Поле Надпись / Caption](#)
-  [Поле Шрифт / Font](#)
-  [Поле Цвет / Color](#)
-  [Поле Выравнивание / Alignment](#)
-  [Событие ПриНажатии / OnClick](#)

Описание

Выравнивание : [Шаблон.ТипыВыравнивания](#);
Alignment : [Template.AlignmentTypes](#);

Назначение

Данное поле позволяет узнать и изменить выравнивание текста в надписи.

Пример

```
proc OnClickRight(B1 :Buttons);  
  -- по нажатию кнопки применяем выравнивание по правому краю  
  Label1.Alignment = Kernel.Template.RightAlign;  
end;
```

Поле Надпись / Caption

Описание

Надпись :Строка;
Caption :String;

Назначение

Данное поле позволяет узнать и изменить текст надписи.

Пример

```
proc ПриИзмененииФлага(О:String);  
  if Флаг1.Состояние then  
    Надпись1.Caption = "Цена с НДС";  
  else  
    Надпись1.Caption = "Цена без НДС";  
  end;  
end;
```

Описание

Цвет : Целое;
Color :Integer;

Назначение

Данное поле позволяет узнать и изменить цвет фона надписи.

Пример

```
proc SetColor(Надпись1 :Надпись; Red :Integer; Green :Integer; Blue :Integer);
var R, G, B, C: Integer;
  -- составляющие цвета должны лежать в пределах 0-255
  R = Mod(Red,256); -- красный
  G = Mod(Green,256); -- зеленый
  B = Mod(Blue,256); -- синий
  C = R + G*256 + B*256*256;
  Надпись1.Цвет = C;
end;
```

Поле Шрифт / Font

Описание

Шрифт : [Шрифт](#) ;

Font : [Font](#) ;

Назначение

Позволяет узнать параметры шрифта объекта Надпись. Поле доступно только на чтение.

Пример

```
Надпись1.Шрифт.Жирный = Истина;
```

Описание

ПриНажатии: Строка;
OnClick: String;

Назначение

Данное событие происходит, когда пользователь щелкает мышью по надписи. Поле **ПриНажатии** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnClick**(Lab: Label);
Lab - объект класса **Надпись**, с которым произошло событие.

Пример

```
proc Надпись1_ПриНажатии(L1: Label);  
  -- в ответ на нажатие надписи она будет становиться  
  -- то печатаемой, то нет, что отображается начертанием шрифта  
  L1.Печатать = NOT L1.Печатать;  
  L1.Шрифт.Жирный = NOT L1.Шрифт.Жирный;  
end;
```

Класс *ОбъектШаблона* (*TemplateObject*), производный от родительского класса [Объект](#), инкапсулирует базовые свойства интерфейсных элементов управления, используемых на шаблонах Студии.

Непосредственно в классе *ОбъектШаблона* определены следующие свойства:

-  [Поле Имя / Name](#)
-  [Поле Владелец / Owner](#)
-  [Поле РодИндекс / ParentIndex](#)
-  [Поле Виден / Видна / Visible](#)
-  [Поле Печатать / Printed](#)
-  [Поле Разрешен / Enabled](#)
-  [Поле ТабСтоп / TabStop](#)
-  [Поле Выравнивать / Align](#)
-  [Поле Подсказка / Hint](#)
-  [Поле КонтекстПомощи / HelpContext](#)
-  [Поле Слева / Left](#)
-  [Поле Сверху / Top](#)
-  [Поле Ширина / Width](#)
-  [Поле Высота / Height](#)
-  [Процедура Сфокусировать / SetFocus](#)
-  [Событие ПриПодсказке / OnHint](#)

Внимание! Создать объект класса *ОбъектШаблона* напрямую нельзя.

Класс *ОбъектШаблона* - это *абстрактный класс*, являющийся базовым для производных классов, объекты которых наследуют свойства данного класса. К наследникам класса *ОбъектШаблона* относятся следующие классы:

- [Кнопка / Button](#)
- [Надпись / Label](#)
- [Флаг / CheckBox](#)
- [Переключатель / RadioButton](#)
- [Редактор / Edit](#)
- [Список / ComboBox](#)
- [Рисунок / Picture](#)
- [Рамка / Bevel](#)
- [РядЗакладок / TabSet](#)
- [Проигрыватель / MediaPlayer](#)
- [График / Chart](#)
- [OLEКонтейнер / OLEContainer](#)
- [КартотекаШаблона / TemplateCardfile](#)
- [ДеревоКартотеки / CardTree](#)
- [ПодтаблицаШаблона / TemplateSubCardfile](#)

Описание

Виден :Логическое;
Видна :Логическое;
Visible :Logical;

Назначение

Данное поле позволяет узнать, виден ли в данный момент объект, а также управлять его видимостью. Если значение поля равно TRUE, объект виден, иначе - скрыт.

Пример

```
proc ButtonPress(O:String);  
    Кнопка1.Видна = TRUE;  
end;
```

Описание

Владелец : [Шаблон](#);
Owner : [Template](#);

Назначение

Поле позволяет получить ссылку на шаблон, в котором расположен данный объект. Поле доступно только на чтение.

Пример

```
-- по нажатию кнопки вызываем метод владельца
-- для создания копии переданной кнопки
proc OnButtonPressed(aButton :Button);
var newButton :Button;
  newButton = aButton.Owner.AddObject(aButton.ClassType);
  newButton.Top = newButton.Top + newButton.Height;
end;
```


Описание

Выравнивать :Логическое;
Align :Logical;

Назначение

Свойство позволяет разрешить (True) или запретить (False) выравнивание текста на объекте шаблона.

Поле доступно на чтение и запись.

Пример

```
picImage      :Picture;  
...  
proc Pl(Object:String);  
    picImage.Align = True;  
end;
```

Описание

Высота :Число;
Height :Numeric;

Назначение

Позволяет получить и установить высоту объекта в миллиметрах.

Пример

```
proc ПриИзмененииФлага1(О:String);  
  if Флаг1.Состояние then  
    Кнопка1.Ширина = 75;  
    Кнопка1.Высота = 10;  
  else  
    Кнопка1.Ширина = 150;  
    Кнопка1.Высота = 15;  
  end;  
end;
```

Описание

Имя : Строка;
Name: String;

Назначение

Данное поле содержит имя объекта, уникальное в контексте шаблона.

Поле доступно только на чтение и позволяет идентифицировать элементы управления. Имя совпадает с именем переменной-объекта, описанной в коде.

Пример

```
Класс "БланкРедактор", Редактор Пример.Накладная;  
  
Кнопка1: Button; -- кнопка с именем 'Кнопка1'  
  
proc OnButtonPressed(But :Button);  
  if But.Name = 'Кнопка1' then  
    -- аналогичную проверку можно было бы сделать,  
    -- сравнив параметр с указателем на объект кнопки:  
    -- if But = Кнопка1 then  
      ...  
    end;  
  end;  
end;
```

Описание

```
КонтекстПомощи : Строка;  
HelpContext    : String;
```

Назначение

Свойство позволяет определить и задать путь к файлу с текстом помощи, поясняющим назначение объекта шаблона.

Этот текст будет появляться в окне справке по прикладным системам, если на объекте установлен фокус и нажата клавиша *F1*.

Если поле пустое, то справка к данному объекту шаблона берется [для фрейма](#), на котором размещается данный объект.

Пример

```
Кнопка1 : Button;  
....  
proc F1(Sender:Button);  
    if Кнопка1.HelpContext='' then  
        Message("Помощь к кнопке отсутствует.");  
    fi;  
end;
```

Описание

Печатать :Логическое;
Printed :Logical;

Назначение

Данное поле позволяет узнать, выводится ли данный объект на печать, а также управлять этим аспектом. Если значение поля равно TRUE, объект будет выводиться на печать, иначе – нет.

Пример

```
Кнопка1.Печатать = FALSE;
```

Описание

```
Подсказка : Строка;  
Hint : String;
```

Назначение

Данное поле позволяет установить текст контекстной подсказки, которая выводится во всплывающем окне, когда пользователь задерживает курсор мыши над объектом.

Пример

```
Кнопка1 : Button;  
proc P1(Object:String);  
    Кнопка1.Hint = "Нажмите "+  
        " на эту кнопку, чтобы провести расчет";  
end;
```

Описание

Разрешен :Логическое;
Enabled :Logical;

Назначение

Данное поле предназначено для управления доступом к объектам. Когда значение поля равно TRUE, объект доступен. Когда значение равно FALSE, объект недоступен (находится в отключенном состоянии). На недоступных объектах текст отображается серым цветом.

Пример

```
proc ButtonPress(O:String);  
    Кнопка1.>Enabled = FALSE;  
end;
```

Описание

РодИндекс :Целое;
ParentIndex :Integer;

Назначение

Поле содержит номер данного объекта в массиве объектов шаблона (массив представлен свойством [Объект/Object](#) класса **Шаблон**).

Поле доступно только на чтение и позволяет идентифицировать элементы управления по их номеру. Объекты нумеруются, начиная с 1.

Пример

```
if Кнопка.РодИндекс = 1 then  
    Кнопка.Видна = Ложь;  
end;
```


Описание

РодФрейм : [ФреймШаблона](#);

Parent : [TemplateFrame](#);

Назначение

Свойство позволяет получить ссылку на родительский фрейм. Если родительский фрейм не существует возвращает пустую ссылку (nil).

Поле доступно только на чтение.

Описание

Сверху :Число;
Тор :Numeric;

Назначение

Позволяет получить и установить положение верхней границы объекта в миллиметрах. Значение берется относительно верхней границы шаблона.

Пример

```
proc ПриИзмененииФлага1(O:String);  
  if Флаг1.Состояние then  
    Кнопка1.Слева = 150; -- двигаем кнопку  
    Кнопка1.Сверху = 20;  
  else  
    Кнопка1.Слева = 100; -- двигаем кнопку  
    Кнопка1.Сверху = 30;  
  end;  
end;
```

Описание

Слева :Число;
Left :Numeric;

Назначение

Позволяет получить и установить положение левой границы объекта в миллиметрах. Значение берется относительно левой границы шаблона.

Пример

```
proc ПриИзмененииФлага1(O:String);  
  if Флаг1.Состояние then  
    Кнопка1.Left = 150; -- двигаем кнопку  
    Кнопка1.Сверху = 20;  
  else  
    Кнопка1.Left = 100; -- двигаем кнопку  
    Кнопка1.Сверху = 30;  
  end;  
end;
```

Описание

TabStop :Логическое;
TabStop :Logical;

Назначение

Данное поле позволяет определить, следует ли системе разрешить переход к объекту с помощью нажатий клавиши *Tab* (прямая последовательность табулируемых объектов) или *Shift+Tab* (обратная последовательность).

Когда значение поля равно TRUE, объект можно сделать активным, последовательно нажимая *Tab*, до тех пор, пока фокус ввода не переместится на него. Когда значение равно FALSE, объект можно сделать активным только с помощью мыши.

По умолчанию значение поля равно FALSE.

Для некоторых типов объектов (**Надпись**, **Рамка**) поле не действует. Такие объекты никогда не получают фокус ввода.

Пример

```
proc BlankOnOpen(Create :Logical);  
  Button1.TabStop = TRUE;  
end;
```

Описание

Ширина :Число;
Width :Numeric;

Назначение

Позволяет получить и установить ширину объекта в миллиметрах.

Пример

```
proc ПриИзмененииФлага1(О:String);  
  if Флаг1.Состояние then  
    Кнопка1.Ширина = 75;  
    Кнопка1.Высота = 10;  
  else  
    Кнопка1.Ширина = 150;  
    Кнопка1.Высота = 15;  
  end;  
end;
```

Описание

СФокусировать ;
SetFocus ;

Назначение

Данный метод в отличии от свойства [ТекущийОбъект](#) не просто делает заданный объект текущим, но и фокусирует окно фрейма. Метод не требует вычисления нужного шаблона. Достаточно, чтобы объект существовал.

Фокусировка игнорируется, если объект был уничтожен или он и его "владельцы" невидимы.

Пример

```
if Button5 <> nil then  
    Button5.SetFocus;  
end;
```

Описание

```
ПриПодсказке : Строка;  
OnHint : String;
```

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при выводе подсказки к текущему объекту.

Обработчик

```
func OnHint(Cell :TemplateCell; var Text :String) :Logical;
```

Параметры:

Cell - указатель на объект класса [ОбъектШаблона](#), в котором произошло событие.

Text - текст подсказки к текущей клетки, который можно изменить.


Функция возвращает True, если разрешается вывести подсказку, иначе - False. Переменная **Text** содержит новый текст, если пользователь изменял текст подсказки, или прежний, заданный при входе в функцию.

Класс *Переключатель* (*RadioButton*), производный от класса *ОбъектШаблона*, представляет собой программный интерфейс для элемента управления типа "зависимый переключатель", который используется в шаблонах Студии.

В классе *Переключатель* можно использовать все свойства и методы, определенные в родительских классах [Объект](#) и [ОбъектШаблона](#).

Внимание! Создать объект класса *Переключатель* из программы на ТБ.Скрипт нельзя. Переключатели создаются самой системой на основе информации о шаблоне, спроектированном в визуальном редакторе бланков Студии.

Непосредственно в классе *Переключатель* определены следующие свойства:

-  [Поле Надпись / Caption](#)
-  [Поле Шрифт / Font](#)
-  [Поле Цвет / Color](#)
-  [Поле Состояние / State](#)
-  [Поле Индекс / Index](#)
-  [Событие ПриВходе / OnEnter](#)
-  [Событие ПриВыходе / OnExit](#)
-  [Событие ПриИзменении / OnChange](#)
-  [Событие ПриНажатии / OnClick](#)

Описание

Индекс : Целое;
Index: Integer;

Назначение

Данное поле позволяет узнать и изменить состояние группы переключателей, в которую входит данный переключатель. Поле содержит индекс включенного в текущий момент переключателя.

Пример

```
proc ПриИзмененииФлага1(O:String);  
  if Флаг1.Состояние then  
    Переключатель1.Index = 2;  
  end;  
end;
```

Поле Надпись / Caption

Описание

Надпись :Строка;
Caption :String;

Назначение

Данное поле позволяет узнать и изменить текст, который выводится в рабочей области переключателя.

Пример

```
proc ПриИзмененииФлагов1(O:String);  
  if Флаг1.Состояние then  
    Переключатель1.Caption = "Цена с НДС";  
  else  
    Переключатель1.Caption = "Цена без НДС";  
  end;  
end;
```

Поле Состояние / State

Описание

Состояние :Логическое;
State :Logical;

Назначение

Данное поле позволяет узнать и установить состояние переключателя: включенное или выключенное. Если переключатель входит в группу переключателей и его состояние меняется на включенное, то все остальные переключатели автоматически отключаются.

Пример

```
proc ПриИзмененииФлага1(O:String);  
  if Флаг1.Состояние then  
    Переключатель1.Состояние = TRUE;  
  end;  
end;
```

Описание

Цвет : Целое;
Color :Integer;

Назначение

Данное поле позволяет узнать и изменить цвет фона под объектом.

Пример

```
var R, G, B, C: Integer;  
-- составляющие цвета должны лежать в пределах 0-255  
R = Mod(Red,256); -- красный  
G = Mod(Green,256); -- зеленый  
B = Mod(Blue,256); -- синий  
C = R + G*256 + B*256*256;  
Переключатель1.Цвет = C;
```

Поле Шрифт / Font

Описание

Шрифт : [Шрифт](#) ;

Font : [Font](#) ;

Назначение

Позволяет узнать параметры шрифта объекта Переключатель.

Поле доступно только на чтение.

Пример

```
Переключатель1.Шрифт.Жирный = Истина;
```

Событие ПриВходе / OnEnter

Описание

ПриВходе: Строка;

OnEnter: String;

Назначение

Данное событие происходит, когда переключатель получает фокус ввода (то есть, становится текущим элементом управления на шаблоне). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB*.

Кроме того, переключатель может получить фокус ввода, если пользователь выделил его щелчком мыши. Поле **ПриВходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

```
proc OnEnter(R: RadioButton);
```

R - объект класса **Переключатель**, с которым произошло событие.

Пример

```
proc Переключатель1_ПриВходе(R1: RadioButton);  
  R1.Font.Bold = TRUE; -- описание текущего переключателя делаем жирным  
end;
```

Описание

ПриВыходе: Строка;
OnExit: String;

Назначение

Данное событие происходит, когда переключатель теряет фокус ввода (то есть, переключатель перестает быть текущим элементом управления).

Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB* или в результате выделения элемента мышью. Поле **ПриВыходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnExit**(R: RadioButton);
R - объект класса **Переключатель**, с которым произошло событие.

Пример

```
proc Переключатель1_ПриВыходе(R1: RadioButton);  
  R1.Font.Bold = FALSE; -- когда фокус ввода уходит с переключателя,  
  -- задаем отрисовку его описания нежирным шрифтом  
end;
```

Описание

ПриИзменении: Строка;
OnChange: String;

Назначение

Данное событие происходит, когда переключатель меняет свое состояние (*только с выключенного на включенное, но не наоборот*). Изменение состояния может происходить как программно, так и в результате действий пользователя. В последнем случае событие происходит до генерации события [ПриНажатии](#). Новое состояние переключателя можно определить с помощью свойства [Состояние](#).

Поле **ПриИзменении** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnChange** (R: RadioButton);
R - объект класса **Переключатель**, с которым произошло событие.

Пример

```
proc Переключатель1_ПриИзменении(R1: RadioButton);  
-- в зависимости от состояния переключателя  
if R1.Name = "Radiol" AND NOT R1.State then  
    Button1.Enabled = TRUE;  
    -- делаем кнопку доступной  
else  
    Button1.Enabled = FALSE;  
    -- делаем кнопку недоступной  
end;  
end;
```


Описание

ПриНажатии: Строка;
OnClick: String;

Назначение

Данное событие происходит при нажатии переключателя (щелчок мышью или клавиша пробел на клавиатуре). Событие происходит после события [ПриИзменении](#).

Поле **ПриНажатии** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnClick**(RB: RadioButton);
RB - объект класса **Переключатель**, с которым произошло событие.

Пример


```
proc Переключателии2_ПриНажатии(RB: RadioButton);  
  if RB = RB1 then  
    RB2.State = NOT RB2.State;  
  else  
    RB1.State = NOT RB1.State;  
  end;  
end;
```

Класс *ПодтаблицаШаблона* / *TemplateSubCardfile*, производный от класса *ОбъектШаблона*, используется в шаблонах для работы с [многозначными полями \(массивами\)](#), в том числе и [структурными](#), определенными с помощью MTL-описания в прикладных классах записей.

Внимание! Создать объект класса *ПодтаблицаШаблона* напрямую нельзя. Подтаблицы на шаблоне создаются самой системой на основе данных о шаблоне, спроектированном в визуальном редакторе.

Класс *ПодтаблицаШаблона* наследует все свойства и методы от родительских классов [Объект](#) и [ОбъектШаблона](#).

Непосредственно в этом классе определены следующие свойства и методы:

-  [Поле Цвет / Color](#)
-  [Поле Шрифт / Font](#)
-  [Поле Бордю / Bevel](#)
-  [Поле Записи / Records / Документы / Documents](#)
-  [Поле ИмяПодтаблицы / SubtableName](#)
-  [Поле ФильтрПодтаблицы / SubtableFilter](#)
-  [Поле Подтаблица / Subtable](#)
-  [Поле ТекущаяЗапись / CurrentRecord](#)
-  [Поле Текущий / Current / Текущая](#)
-  [Поле МожноДобавлять / CanInsert](#)
-  [Поле МожноУдалять / CanDelete](#)
-  [Поле МожноПеремещать / CanMove](#)
-  [Поле МожноРедактировать / CanEdit](#)
-  [Поле КоличествоВыделенных / SelectedCount](#)
-  [Поле Выделенные / Selected](#)
-  [Поле ТекущийСтолбец / CurrentColumn](#)
-  [ПозТекущегоСтолбца / CurrentColumnPos](#)
-  [Поле ФиксированныхСтолбцов / FixedColumns](#)
-  [Поле КоличествоСтолбцов / ColumnsCount](#)
-  [Поле Столбец / Column](#)
-  [Поле СтолбецПоПолю / ColumnByField](#)
-  [Функция ДобавитьСтолбец / AddColumn](#)
-  [Функция ВставитьСтолбец / InsertColumn](#)
-  [Процедура УдалитьСтолбец / DeleteColumn](#)
-  [Процедура НачатьРедактирование / BeginEdit](#)
-  [Процедура ЗавершитьРедактирование / EndEdit](#)
-  [Процедура Обновить / Update](#)

-  [Событие ПриВходе / OnEnter](#)
-  [Событие ПриВыходе / OnExit](#)
-  [Событие ПриНажатии / OnClick](#)
-  [Событие ПриВставке / OnInsert](#)
-  [Событие ПриУдалении / OnDelete](#)
-  [Событие ПриСменеВыделения / OnChangeSelected](#)
-  [Событие ПриОформлении / OnRearrange](#)
-  [Событие ПриСменеПозиции / OnChangePosition](#)

Описание

Бордюр :Логическое;
Bevel :Logical;

Назначение

Данное поле позволяет узнать и изменить настройку, определяющую, нужно ли выводить трехмерный бордюр вокруг объекта.

Пример

```
TemplateSubCardfile1: TemplateSubCardfile;  
...  
TemplateSubCardfile1.Bevel = true;
```

Описание

Выделенные[Индекс: Целое] : [Запись](#);
Selected[Index: Integer] : [Record](#);

Назначение

Поле позволяет по номеру получить ссылку на конкретную структуру из числа выделенных в подтаблице. Индекс меняется от 1 до числа выделенных структур, которое определяется с помощью свойства [КоличествоВыделенных](#).

Поле доступно только на чтение.

Пример

```
TemplateSubCardfile1 :TemplateSubCardfile;  
  
proc Пересчет;  
var x :Документы.ПриходРасход.Позиция;  
-- цикл по выделенным строкам подтаблицы  
for i=1..TemplateSubCardfile1.SelectedCount do  
  x = TemplateSubCardfile1.Selected[i]  
  as Документы.ПриходРасход.Позиция;  
  -- обработка  
  CountUp(x.Value);  
end;  
end;
```

Описание

Записи : Класс [Запись\[\]](#);
Records :Class [Record\[\]](#);

Назначение

Данное поле содержит массив классов записей, подтаблицы которых могут обрабатываться объектом рассматриваемого класса **ПодтаблицаШаблона**.

В поле можно записать новый массив классов записей, однако поэлементное присваивание не допускается.

Пример

```
TemplateSubCardfile.Records = [Документы.Накладные];
```

Описание

ИмяПодтаблицы : Строка;
SubtableName : String;

Назначение

Поле позволяет узнать или задать имя подтаблицы, которая будет связана с данным объектом класса **ПодтаблицаШаблона**. Подтаблица с указанным именем должна существовать в классах записей, отобранных с помощью поля [Записи](#).

Пример

```
TemplateSubCardfile1.Records = [Документы.Накладные];  
TemplateSubCardfile1.SubtableName = "Позиции";
```

Описание

КоличествоВыделенных : Целое;
SelectedCount : Integer;

Назначение

Данное поле содержит количество выделенных в данный момент в подтаблице структур. Поле доступно только на чтение.

Пример

```
TemplateSubCardfile1 :TemplateSubCardfile;  
...  
-- цикл по выделенным строкам подтаблицы  
for i=1..TemplateSubCardfile1.SelectedCount do  
    -- обработка  
end;
```

Описание

КоличествоСтолбцов : Целое;
ColumnsCount : Integer;

Назначение

Поле позволяет узнать количество столбцов в подтаблице. Поле доступно только на чтение.

Пример

```
-- цикл по всем столбцам подтаблицы
for i=1..TemplateSubCardfile11.ColumnsCount do
  if Шаблон.ПолеДаты = TemplateSubCardfile11.Column[i].FieldType then
    -- если дата
    TemplateSubCardfile11.Column[i].Format = "dd mmmm yyyy";
  end;
end;
```


Описание

МожноДобавлять :Логическое;
CanInsert :Logical;

Назначение

Поле определяет, можно ли интерактивно добавлять строки в подтаблицу. В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге [свойств подтаблицы](#) (на странице "Правка").

Пример

```
SubCardfile1.CanInsert = CheckBox1.State;
```

Описание

МожноПеремещать :Логическое;
CanMove :Logical;

Назначение

Поле определяет, можно ли интерактивно перемещать строки в подтаблице. В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге [свойств подтаблицы](#) (на странице "Правка").

Пример

```
SubCardfile1.CanMove = CheckBox3.State;
```

Описание

МожноРедактировать :Логическое;
CanEdit :Logical;

Назначение

Поле определяет, можно ли интерактивно редактировать значения в клетках подтаблицы. В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге [свойств подтаблицы](#) (на странице "Правка").

Пример

```
SubCardfile1.CanEdit = CheckBox4.State;
```

Описание

МожноУдалять :Логическое;
CanDelete :Logical;

Назначение

Поле определяет, можно ли интерактивно удалять строки в подтаблице. В дизайн-режиме данное свойство задается с помощью соответствующего флага в диалоге [свойств подтаблицы](#) (на странице "Правка").

Пример

```
SubCardfile1.CanDelete = CheckBox2.State;
```

Описание

Подтаблица : [Подтаблица](#);
Subtable : [Subtable](#);

Назначение

Поле содержит ссылку на объект класса **Подтаблица**, связанного с данной подтаблицей шаблона. Имя этой подтаблицы можно узнать с помощью свойства [ИмяПодтаблицы](#).

Поле доступно только на чтение.

Пример

```
TemplateSubCardfile1.Records = [Документы.Накладные];  
TemplateSubCardfile1.SubtableName = "Позиции";  
TemplateSubCardfile1.SubtableFilter = "Код <> 0";  
-- выводим отладочную информацию о количестве записей  
-- в подтаблице (с учетом наложения фильтра)  
trace(TemplateSubCardfile1.Subtable.Count);
```

Описание

ПозТекущегоСтолбца : Целое;
CurrentColumnPos : Integer;

Назначение

Поле содержит номер видимого на экране столбца, содержащего выделенную клетку подтаблицы, а также позволяет назначить выделение, указав другой текущий столбец.

Внимание! Свойство **ПозТекущегоСтолбца**=CurrentColumn.Index, только в том случае, когда видимыми являются все столбцы текущей подтаблицы. В общем случае номер видимого столбца ПозТекущегоСтолбца не совпадает с порядковым номером столбца подтаблицы CurrentColumn.Index.

Валидность диапазона при установке свойства не проверяется, поэтому, чтобы встать на последний столбец можно написать CurrentColumnPos = 999; или задать общее [КоличествоСтолбцов](#), (см. пример ниже).

Пример

```
-- выделяем клетку в самом последнем столбце  
TemplateSubCardfile1.CurrentColumnPos = TemplateSubCardfile1.ColumnsCount;
```

Описание

Столбец[Индекс: Целое] : [СтолбецПодтаблицы](#);
Column [Index: Integer] : [SubCardfileColumn](#);

Аргументы

Индекс - номер столбца. Индекс может изменяться в пределах от 1 до [количества столбцов](#).

Назначение

Данное поле позволяет получить доступ к конкретному столбцу подтаблицы по его номеру. Поле доступно только на чтение.

Пример

```
-- цикл по всем столбцам подтаблицы
for i=1..TemplateSubCardfile11.ColumnsCount do
  if Шаблон.ПолеДаты = TemplateSubCardfile11.Column[i].FieldType then
    -- если дата
    TemplateSubCardfile11.Column[i].Format = "dd mmmm yyyy";
  end;
end;
```

Описание

СтолбецПоПолю[Имя: Строка] : [СтолбецПодтаблицы](#);
ColumnByField[Name: String] : [SubCardfileColumn](#);

Аргументы

Имя - имя столбца подтаблицы.

Назначение

Данное поле позволяет получить доступ к конкретному столбцу подтаблицы по имени поля, содержащегося в нем. Поле доступно только на чтение.

Пример

```
proc BoldByName(S:SubCardfile);  
var cc: SubCardFileColumn;  
  cc = S.ColumnByField["Сумма"];  
  cc.Font.Bold = TRUE;  
end;
```


Описание

ТекущаяЗапись : [Запись](#) ;
CurrentRecord : [Record](#) ;

Назначение

Поле содержит указатель на текущую запись, структура которой в данный момент отображается в подтаблице. Поле доступно на запись.

Пример

```
TemplateSubCardfile1 :TemplateSubCardfile;  
TemplateCardfile1 :TemplateCardfile;  
  
-- при выделении новой записи в таблице картотеки на шаблоне  
-- выводим структуру из этой записи в подтаблицу  
proc Card_OnMove(TC :TemplateCardfile; D:Document);  
  if TC = TemplateCardfile1 then  
    TemplateSubCardfile1.CurrentRecord = TemplateCardfile1.Current;  
  end;  
end;
```

Поле Текущая / Current

Описание

Текущая : [Запись](#);

Current : [Record](#);

Назначение

Данное поле содержит указатель на текущую структуру в подтаблице. Изменяя значение поля можно менять текущую строку в подтаблице.

Следует отметить, что текущая строка визуально отличается от остальных строк подтаблицы – либо отдельная клетка строки, либо вся строка (в зависимости от типа выделения) отображается контрастным цветом.

Пример

```
-- выводим название товара из текущей позиции в строку состояния  
Hint(SubCardfileGoods.Current.Товар);
```

Описание

ТекущийСтолбец : [СтолбецПодтаблицы](#);
CurrentColumn : [SubCardfileColumn](#);

Назначение

Поле возвращает ссылку на столбец подтаблицы, содержащий выделенную клетку. Свойство доступно на чтение и запись.

Внимание! При попытке выделить невидимый столбец или столбец, отсутствующий в подтаблице, текущая позиция *не* **меняется**.

Пример

```
TemplateSubCardfile1 :TemplateSubCardfile;  
...  
TemplateSubCardfile1.CurrentColumn.Font.Bold = TRUE;
```

Описание

ФиксированныхСтолбцов : Целое;
FixedColumns : Integer;

Назначение

Данное свойство позволяет узнать и установить количество фиксированных столбцов подтаблице картотеки на шаблоне. Фиксированные столбцы остаются у левого края подтаблицы при ее прокрутке по горизонтали.

Пример

```
ПодтаблицаШаблона1 :ПодтаблицаШаблона;  
  
proc ButtonClick(B:Button);  
    ПодтаблицаШаблона1.ФиксированныхСтолбцов = 2;  
end;
```

Описание

ФильтрПодтаблицы : Строка;
SubtableFilter : String;

Назначение

Поле **ФильтрПодтаблицы** позволяет установить на отображаемые записи подтаблицы произвольный фильтр – строковое выражение, отвечающее общим правилам написания фильтров (см. раздел [Установка фильтра](#)). В подтаблице будут отображаться только те записи, для которых выражение фильтра принимает истинное значение.

Пример

```
TemplateSubCardfile1.Records = [Документы.Накладные];  
TemplateSubCardfile1.SubtableName = "Позиции";  
TemplateSubCardfile1.SubtableFilter = "Код <> 0";
```

Описание

Цвет : Целое;
Color :Integer;

Назначение

Данное поле позволяет узнать и изменить цвет фона под объектом.

Пример

```
var R, G, B, C: Integer;  
-- составляющие цвета должны лежать в пределах 0-255  
R = Mod(Red,256); -- красный  
G = Mod(Green,256); -- зеленый  
B = Mod(Blue,256); -- синий  
C = R + G*256 + B*256*256;  
ПодтаблицаШаблона1.Цвет = C;
```

[Класс Объект](#) : [ОбъектШаблона](#) : [ПодтаблицаШаблона](#)

Поле Шрифт / Font

Описание

Шрифт : [Шрифт](#) ;

Font : [Font](#) ;

Назначение

Позволяет узнать параметры шрифта объекта ПодтаблицаШаблона. Поле доступно только на чтение.

Пример

```
ПодтаблицаШаблона1.Шрифт.Жирный = Истина ;
```

Описание

```
ЗавершитьРедактирование[ (ПринятьИзменения :Логическое) ] ;  
EndEdit[ (AcceptChanges: Logical) ] ;
```

Аргументы

ПринятьИзменения - определяет, следует ли принять или отклонить сделанные изменения.

Назначение

Закрывает встроенный (in-place) редактор, открытый ранее с помощью [BeginEdit](#) или пользователем. Если **ПринятьИзменения** равно TRUE, изменения (если они были) передаются в обработчик [OnVerify](#) (если он установлен) и затем сохраняются во временном представлении текущей записи. Впоследствии, изменения будут окончательно приняты, если будет вызван метод [Post](#) или если пользователь выполнит соответствующую команду (**Записать**). Изменения не проходят, если будет вызван метод [Cancel](#) для записи, содержащей подтаблицу, или пользователь выполнит соответствующую команду (**Отменить**).

Если **ПринятьИзменения** равно FALSE, изменения не попадают из встроенного редактора во временный буфер, где хранится текущая редактируемая запись. Если параметр **ПринятьИзменения** опущен, он считается равным TRUE.

Пример

```
proc ButtonOKClick(B:Button);  
    TemplateSubCardfile1.EndEdit;  
end;
```


Описание

НачатьРедактирование;
BeginEdit;

Назначение

Открывает в выделенной клетке подтаблицы встроенный (in-place) редактор, позволяющий изменить значение в ней. При этом как подтаблица, так и вся запись, содержащая данную подтаблицу, переводится в состояние редактирования.

Редактирование возможно только в том случае, если в настройках подтаблицы разрешена ее модификация.

Пример

```
proc B1Click(B:Button);  
    TemplateSubCardfile1.BeginEdit;  
end;
```

Описание

```
УдалитьСтолбец(Номер :Целое);  
DeleteColumn(Position :Integer);
```

Аргументы

Номер - задает номер столбца, который следует удалить. Номер должен лежать в пределах от 1 до [количества столбцов](#).

Назначение

Процедура удаляет указанный столбец из подтаблицы картотеки на шаблоне.

Пример

```
-- код класса формы бланка  
-- ...  
-- удаляем первый столбец из подтаблицы  
TemplateSubCardfile1.DeleteColumn(1);
```

Описание

ПриВставке : Строка;
OnInsert: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при вставке новой структуры в подтаблицу. Событие происходит после того, как пользователь вызвал команду вставки структуры (например, нажал кнопку Ins), но до того, как началось ее редактирование. Событие позволяет проинициализировать требуемые поля структуры или вообще запретить ее создание.

Обработчик

```
func OnInsert(Sender :TemplateSubCardfile;Index :Integer ) :Logical;
```

Параметры:

Sender - указатель на объект класса **ПодтаблицаШаблона**, с которым произошло событие;
Index - номер добавляемой структуры.

Если функция возвращает TRUE, происходит стандартная вставка структуры и позиционирование на нее.
Если функция возвращает FALSE, вставка не выполняется.

Пример

```
func CardfileOrder_OnInsert(Sender :TemplateSubCardfile; Index :Integer) :Logical;  
var S :Structure;  
    S = Sender.Subtable.Items[Index];  
    -- заполняем поле структуры значением по умолчанию  
    (S As Документы.ПриходРасход.Позиции).Количество = 1;  
    return true;  
end;
```

Описание

ПриВходе: Строка;
OnEnter: String;

Назначение

Данное событие происходит, когда объект подтаблицы получает фокус ввода (то есть, становится текущим элементом управления на шаблоне). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB* или по щелчку мыши.

Поле **ПриВходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

```
proc OnEnter(Sender :TemplateSubCardfile);
```

Параметры:

Sender - указатель на объект класса **ПодтаблицаШаблона**, с которым произошло событие.

Пример

```
proc ПодтаблицаШаблона_ПриВходе(Sender :TemplateSubCardfile);  
  Sender.CurrentColumn = 1;  
end;
```

Описание

ПриВыходе: Строка;
OnExit: String;

Назначение

Данное событие происходит, когда объект подтаблицы теряет фокус ввода (то есть, перестает быть текущим элементом управления). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB* или по щелчку мыши.

Поле **ПриВыходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnExit**(Sender :TemplateSubCardfile);

Параметры:

Sender - указатель на объект класса **ПодтаблицаШаблона**, с которым произошло событие.

Пример

```
proc ПодтаблицаШаблона_ПриВыходе(ПШ :ПодтаблицаШаблона);  
  if ПШ.КоличествоВыделенных = 0 then  
    message("Выделите строки для обработки");  
  end;  
end;
```

```
proc КартотекаШаблона_ПриВыходе(ТС :TemplateCardfile);  
  ТС.Font.Bold = FALSE;  
end;
```

Описание

ПриНажатии: Строка;
OnClick: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при одиночном или двойном щелчке мышью на любой клетке подтаблицы , а также при нажатии клавиши **Enter**.

Обработчик

```
func OnClick(Sender :TemplateSubCardfile; Action : Template.ClickTypes; Column :SubCardfileColumn;  
Struct :Structure) :Logical;
```

Параметры:

Sender - указатель на объект класса **ПодтаблицаШаблона**, с которым произошло событие;
Action - одна из предопределенных констант, описанных в типе [Шаблон.ТипыНажатия](#), которая определяет тип события:
 0 - одинарный щелчок мышью;
 1 - двойной щелчок мышью;
 2 - нажатие клавиши **Enter**.
Rec - текущий документ (запись);
Column - текущий столбец подтаблицы;
Struct - текущая структура (строка) подтаблицы.

Если функция возвращает True, происходит стандартная обработка события, то есть открытие бланка-редактора, открытие встроенного (in-place) редактора, вход в группу или выбор текущего документа, в зависимости от настроек картотеки. Если функция возвращает False, стандартная обработка не выполняется.

Если событие не обрабатывается, то это эквивалентно возврату значения True.

Пример

```
-- обработчик события "щелчок мышью в колонке подтаблицы"  
func CardfileSubTable_OnClick(Sender :TemplateSubCardfile;  
Action:Integer; Column:SubCardfileColumn; Struct:Structure):Logical;  
  if Action>0 then  
    TemplateSubCardFile1.BeginEdit;  
    -- приступить к редактированию клетки  
  end;  
  return Ложь;  
end;
```

Описание

ПриОформлении: Строка;
OnRearrange: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при изменении пользователем атрибутов столбца. Системой отслеживаются изменения ширины, положения, видимости и сортировки.

Обработчик

```
proc OnRearrange(Sender      :TemplateSubCardfile;      Column      : SubCardfileColumn;      Action      :  
Cardfile.RearrangeActions);
```

Параметры:

Sender - указатель на объект класса **ПодтаблицаШаблона**;
Action - константа, описанная в перечислимом типе [Картотека.ДействияОформления](#) и характеризующая тип события, связанного с изменением какого-либо атрибута столбца;
Column - указатель на объект класса **СтолбецПодтаблицы**, в котором произошло событие.

Внимание. Если параметр Action=Cardfile.OtherRearrangeActions, параметр Column = nil. В этом случае событие может вызываться, например, при изменении ширины панелей или включении/выключении видимости удаленных записей.

Пример

```
proc SubCard_OnRearrange(Sender :TemplateSubCardfile;  
  Action :Cardfile.Rearrange; ActionsColumn :SubCardfileColumn);  
  -- в случае сортировки по какому-либо столбцу  
  if Action = Cardfile.Sort then  
    -- вызываем прикладную функцию для сортировки  
    -- служебных массивов  
    SortAuxArrays(Column);  
  end;  
end;
```

Описание

ПриСменеВыделения: Строка;
OnChangeSelected: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при изменении выделения нескольких структур в подтаблице.

Обработчик

```
proc OnChangeSelected(Sender :TemplateSubCardfile; Count :Integer);
```

Параметры:

Sender - указатель на объект класса **ПодтаблицаШаблона**, с которым произошло событие;
Count - количество выделенных структур.

Когда выделение снимается, **Count** равен нулю.

Количество выделенных документов и ссылки на сами документы можно получить соответственно с помощью свойств [КоличествоВыделенных](#) и [Выделенные](#).

Пример

```
proc Order_OnChangeSelected(Sender :TemplateSubCardfile; Count :Integer);  
  hint("Выделено:" + Str(Count));  
end;
```


Описание

ПриСменеПозиции :Строка;
OnChangePosition :String;

Назначение

Данное событие позволяет управлять процессом перемещения строк в подтаблице.

Обработчик

```
func OnChangePosition(Sender :TemplateSubCardfile; Index :Integer; MoveUp :Logical) :Logical;
```

Параметры:

Sender - указатель на объект класса **ПодтаблицаШаблона**, с которым произошло событие;

Index - номер перемещаемой в подтаблице строки, нумерация начинается с 1;

MoveUp - логический параметр, равный равно TRUE, если строка перемещается вверх, и FALSE - в противном случае.

Возвращаемый функцией результат разрешает (TRUE) либо запрещает (FALSE) перемещение. В случае, если обработчик вернул FALSE, перемещение можно осуществить, запрограммировав его вручную (с учетом специфики выполняемой задачи).

Пример

```
ТаблицаТоварныхПозиций :TemplateSubCarfile;  
-- ...  
func ПодТабл_ПриСменеПозиции(Sender :TemplateSubCarfile;  
N :Integer; Up :Logical): Logical;  
  if Sender = ТаблицаТоварныхПозиций then  
    return TRUE; -- разрешаем перемещение кадра  
  else  
    return FALSE; -- запрещаем перемещение кадра  
  end;  
end;
```

Описание

ПриУдалении : Строка;
OnDelete : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит перед удалением структуры (строки) подтаблицы.

Обработчик

```
func OnDelete(Sender :TemplateSubCardfile; Index :Integer) :Logical;
```

Параметры:

Sender - указатель на объект класса **ПодтаблицаШаблона**, с которым произошло событие;
Index - номер элемента подтаблицы, то есть номер структуры, которая удаляется.

Если функция возвращает TRUE, происходит стандартная обработка события, то есть структура удаляется. Если функция возвращает FALSE, стандартная обработка не выполняется.

Пример

```
func Order_OnDelete(Sender :TemplateSubCardfile; Index :Integer) :Logical;  
  -- любую строку кроме первой можно удалить  
  if Index <> 1 then  
    return TRUE;  
  end;  
  return FALSE;  
end;
```

Описание

ВставитьСтолбец[Номер: Целое] : [СтолбецПодтаблицы](#);
InsertColumn[Index: Integer] : [SubCardfileColumn](#);

Аргументы

Номер - задает позицию, в которую будет вставлен новый столбец. Номер должен лежать в пределах от 1 до [количества столбцов](#).

Назначение

Функция вставляет новый столбец в подтаблицу картотеки на шаблоне. Позиция нового столбца задается аргументом **Номер**.

Пример

```
-- код класса формы бланка

proc PlusColumn1(FieldName :String);
var Column: SubCardfileColumn;
  -- вставляем новый столбец в первую подтаблицу
  -- новый столбец должен располагаться первым
  Column = TemplateSubCardfile1.InsertColumn(1);
  -- настраиваем столбец
  Column.FieldName = FieldName;
  Column.Width = 200;
  -- ...
end;
```

Описание

ДобавитьСтолбец : [СтолбецПодтаблицы](#);
AddColumn : [SubCardfileColumn](#);

Назначение

Функция добавляет новый столбец в подтаблицу картотеки на шаблоне. Столбец становится последним в массиве столбцов.

Пример

```
-- код класса формы бланка

proc PlusColumn(FieldName :String);
var Column: SubCardfileColumn;
  -- добавляем столбец в первую подтаблицу
  Column = TemplateSubCardfile1.AddColumn;
  -- настраиваем столбец
  Column.FieldName = FieldName;
  Column.Width = 200;
  -- ...
end;
```

Класс *Проигрыватель / MediaPlayer*, производный от класса *ОбъектШаблона*, позволяет работать с мультимедиа-материалами (например, в форматах WAV, MIDI, AVI и других) непосредственно в шаблонах Студии.

Внимание! Создать объект класса *Проигрыватель* из программы на ТБ.Скрипт нельзя. Объекты класса *Проигрыватель* создаются самой системой на основе информации о шаблоне, спроектированном в визуальном редакторе бланков Студии.

Класс *Проигрыватель* наследует все свойства и методы от родительских классов [Объект](#) и [ОбъектШаблона](#).

Непосредственно в классе *Проигрыватель* определены следующие свойства:

-  [Поле Файл / File](#)
-  [Поле Состояние / State](#)
-  [Процедура Воспроизведение / Play](#)
-  [Процедура Стоп / Stop](#)
-  [Процедура Пауза / Pause](#)
-  [Процедура Извлечение / Eject](#)
-  [Событие ПриНажатии / OnClick](#)
-  [Событие ПриЗавершении / OnStop](#)

Поле Состояние / State

Описание

Состояние :Целое;
State :Integer;

Назначение

Позволяет определить текущее состояние проигрывателя. Возможны следующие значения:

- 0** - проигрыватель не готов;
- 1** - проигрыватель остановлен;
- 2** - проигрыватель в режиме воспроизведения;
- 3** - проигрыватель в состоянии записи;
- 4** - проигрыватель в состоянии поиска;
- 5** - проигрыватель приостановлен;
- 6** - лоток устройства мультимедиа открыт.

Пример

```
proc ButtonStateClick(Sender :String);  
    message(StateName[MediaPlayer2.State]);  
end;
```

Поле Файл / File

Описание

Файл :Строка;
File :String;

Назначение

Данное поле позволяет определить мультимедиа-файл для воспроизведения.

Пример

```
MediaPlayer1.File = "C:\Windows\Media\PlayLogo.avi";  
MediaPlayer1.Play;
```

Описание

Воспроизведение;
Play;

Назначение

Начинает воспроизведение файла, указанного ранее с помощью поля [Файл](#).

Пример

```
proc Player1Click(Sender :String);  
  if MediaPlayer1.State = 2 :  
    MediaPlayer1.Stop;  
  else  
    MediaPlayer1.Play;  
  end;  
end;
```


Описание

Извлечение;
Eject;

Назначение

Выгружает мультимедиа-носитель (если тот поддерживает такую операцию).

Пример

```
proc ButtonEjectClick(Sender :String);  
    MediaPlayer2.Eject;  
end;
```

Процедура Пауза / Pause

Описание

Пауза;
Pause;

Назначение

Приостанавливает воспроизведение мультимедиа-файла.

Пример

```
proc ButtonPauseClick(Sender :String);  
    MediaPlayer2.Pause;  
end;
```

Процедура Стоп / Stop

Описание

Стоп;
Stop;

Назначение

Прекращает воспроизведение файла, указанного ранее с помощью поля [Файл](#).

Пример

```
proc Player1Click(Sender :String);  
  if MediaPlayer1.State = 2 :  
    MediaPlayer1.Stop;  
  else  
    MediaPlayer1.Play;  
  end;  
end;
```

Описание

ПриЗавершении: Строка;
OnStop: String;

Назначение

Данное событие происходит при завершении воспроизведения.

Поле **ПриЗавершении** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnStop** (M: MediaPlayer);
M - объект класса **Проигрыватель**, с которым произошло событие.

Пример

```
proc Проигрыватель1_OnStop(M1: MediaPlayer);  
  -- по завершении воспроизведения разрешаем снова его запустить  
  StartPlayButton.Enabled = TRUE;  
end;
```

Описание

ПриНажатии: Строка;
OnClick: String;

Назначение

Данное событие происходит при нажатии клавиши *Enter* или кнопки мыши на выделенном проигрывателе.
Поле **ПриНажатии** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnClick**(M: MediaPlayer);
M - объект класса **Проигрыватель**, с которым произошло событие.

Пример

```
proc Проигрыватель1_ПриНажатии(M1: MediaPlayer);  
  M1.Play;  
  -- запускаем воспроизведение  
end;
```

Рамка / Bevel

Класс *Рамка* (*Bevel*), производный от класса *ОбъектШаблона*, предназначен для работы с рамками, расположенными на шаблонах Студии. Как правило, рамки используются для визуального выделения групп логически взаимосвязанных элементов управления.

Внимание! Создать объект класса *Рамка* из программы на ТБ.Скрипт нельзя. Объекты класса *Рамка* создаются самой системой на основе информации о шаблоне, спроектированном в визуальном редакторе бланков Студии.

Класс *Рамка* наследует все свойства и методы от родительских классов [Объект](#) и [ОбъектШаблона](#).

Непосредственно в этом классе определены следующие свойства:



[Поле Надпись / Caption](#)



[Поле Шрифт / Font](#)

Поле Надпись / Caption

Описание

Надпись :Строка;
Caption :String;

Назначение

Данное поле позволяет узнать и изменить текст, который выводится в качестве заголовка рамки.

Пример

```
Рамка1.Надпись = "Группа";
```

Поле Шрифт / Font

Описание

Шрифт : [Шрифт](#) ;

Font : [Font](#) ;

Назначение

Позволяет узнать параметры шрифта объекта Рамка. Поле доступно только на чтение.

Пример








```
Рамка1.Шрифт.Жирный = Истина;
```





Редактор / Edit

Класс *Редактор/Edit*, производный от класса *ОбъектШаблона*, представляет собой программный интерфейс для элемента управления типа "поле ввода", который используется в шаблонах Студии.

Внимание! Создать объект класса *Редактор* из программы на ТБ.Скрипт нельзя. Объекты *Редактор* создаются самой системой на основе информации о шаблоне, спроектированном в визуальном редакторе бланков Студии.

Класс *Редактор* наследует все свойства и методы от родительских классов [Объект](#) и [ОбъектШаблона](#). Непосредственно в этом классе определены следующие свойства:

-  [Поле Текст / Text](#)
-  [Поле Шрифт / Font](#)
-  [Поле Цвет / Color](#)
-  [Поле Выравнивание / Alignment](#)
-  [Поле ВыделенныйТекст / SelText](#)
-  [Поле НачалоВыделения / SelStart](#)
-  [Поле ДлинаВыделения / SelLength](#)

-  [Событие ПриВходе / OnEnter](#)
-  [Событие ПриВыходе / OnExit](#)
-  [Событие ПриИзменении / OnChange](#)

Описание

ВыделенныйТекст: Строка;
SelText: String;

Назначение

При обращении на чтение данное поле позволяет получить текущий выделенный фрагмент текста. Если выделения в тексте редактора нет - возвращается пустая строка.

При изменении данного свойства (т.е. при записи в поле новой строки) текущий выделенный фрагмент текста заменяется на указанный. Если выделения в тексте редактора нет – новый текст вставляется в редактор, начиная с текущей позиции курсора.

Пример

```
Editor1 :Edit;

proc Button1Press(Sender :Button)
var S :String;
  -- если есть выделенный текст в редакторе
  if Editor1.SelLength > 0 then
    -- получаем текущее выделение
    S = Editor1.SelText;
    -- заключаем всю строку в кавычки
    S = "'" + S + "'";
    -- записываем обновленную строку в редактор
    -- вместо прежнего выделенного фрагмента
    Editor1.SelText = S;
  end;
end;
```

Описание

Выравнивание : [Шаблон.ТипыВыравнивания](#);
Alignment : [Template.AlignmentTypes](#);

Назначение

Данное поле позволяет узнать и изменить выравнивание текста в редакторе.

Пример

```
proc OnClickRight(B1 :Buttons);  
  -- по нажатию кнопки применяем выравнивание по правому краю  
  Edit1.Alignment = Kernel.Template.RightAlign;  
end;
```

Описание

ДлинаВыделения :Целое;
SelLength :Integer;

Назначение

При обращении на чтение данное поле позволяет узнать длину выделенного фрагмента текста. Если выделения в тексте редактора нет - возвращается ноль.

При изменении данного свойства в тексте редактора выделяется фрагмент, начинающийся от позиции курсора (свойство [НачалоВыделения](#)) и указанной с помощью этого поля длиной. Если при этом окажется, что затребованная длина выделения больше, чем длина текста в редакторе (от начальной позиции выделения), то выделение осуществляется до конца текста. Последующее чтение поля **ДлинаВыделения** вернет фактическую длину выделения, а не установленное перед этим значение.

Пример

```
Editor1 :Edit;

proc Button1Press(Sender :Button)
var S :String;
  -- проверяем, есть ли выделенный текст в редакторе
  if Editor1.SelLength > 0 then
    -- получаем текущее выделение
    S = Editor1.SelText;
    -- заключаем всю строку в кавычки
    S = '"' + S + '"';
    -- записываем обновленную строку в редактор
    -- вместо прежнего выделенного фрагмента
    Editor1.SelText = S;
  end;
end;
```

Описание

НачалоВыделения :Целое;
SelStart :Integer;

Назначение

При обращении на чтение данное поле позволяет узнать позицию (смещение в тексте от его начала), с которой начинается текущий выделенный фрагмент. Если выделения в тексте редактора нет - свойство возвращает текущую позицию курсора. Первый символ в тексте имеет позицию 0.

Запись нового значения в поле устанавливает курсор в указанную позицию. Если при этом также изменяется и поле [ДлинаВыделения](#), в редакторе выделяется фрагмент, начиная с заданной с помощью **НачалоВыделения** позиции и длиной **ДлинаВыделения** символов. Если при этом окажется, что затребованная длина выделения больше, чем длина всего текста в редакторе, выделение осуществляется до конца текста.

Пример

```
Editor1 :Edit;  
  
-- процедура устанавливает курсор за последний  
-- символ в редакторе  
proc Button1Press(Sender :Button)  
var L :Integer;  
    L = Length(Editor1.Text);  
    Editor1.SelStart = L;  
end;
```

Поле Текст / Text

Описание

Текст: Строка;

Text: String;

Назначение

Данное поле позволяет узнать и изменить текст, который находится в рабочей области редактора.

Пример

```
var Settings1 : String;  
proc ПриЗакрытииБланка(Режим:Logical);  
    Settings1 = Edit1.Text;  
end;
```

Описание

Цвет : Целое;
Color :Integer;

Назначение

Данное поле позволяет узнать и изменить цвет фона редактора.

Пример

```
var R, G, B, C: Integer;  
-- составляющие цвета должны лежать в пределах 0-255  
R = Mod(Red,256); -- красный  
G = Mod(Green,256); -- зеленый  
B = Mod(Blue,256); -- синий  
C = R + G*256 + B*256*256;  
Редактор1.Цвет = C;
```

Поле Шрифт / Font

Описание

Шрифт : [Шрифт](#) ;

Font : [Font](#) ;

Назначение

Позволяет узнать параметры шрифта объекта Редактор. Поле доступно только на чтение.

Пример

```
Редактор1.Шрифт.Жирный = Истина;
```


Описание

ПриВходе: Строка;
OnEnter: String;

Назначение

Данное событие происходит, когда редактор становится текущим элементом управления на шаблоне. Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB*. Кроме того, редактор может получить фокус ввода, если пользователь выделил его щелчком мыши. Поле **ПриВходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnEnter** (E: Edit);
E - объект класса **Редактор**, с которым произошло событие.

Пример

```
proc Редактор1_ПриВходе(E1: Edit);  
  Пояснение = "В это поле вводится дата начала периода";  
  -- выводим пояснение, когда пользователь собирается заполнить поле  
end;
```

Описание

ПриВыходе: Строка;
OnExit: String;

Назначение

Данное событие происходит, когда редактор теряет фокус ввода (то есть, редактор перестает быть текущим элементом управления). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB* или в результате выделения элемента мышью. Поле **ПриВыходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnExitE**: Edit);
E - объект класса **Редактор**, с которым произошло событие.

Пример

```
proc Редактор1_ПриВыходе(E1: Edit);  
  Пояснение = "";  
  -- убираем пояснение, когда курсор уходит с поля  
end;
```

Описание

ПриИзменении: Строка;
OnChange: String;

Назначение

Данное событие происходит, когда текст в редакторе изменен. Новое содержимое редактора можно определить с помощью свойства [Текст](#).

Поле **ПриИзменении** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnChange**(E: Edit);
E - объект класса **Редактор**, с которым произошло событие.

Пример

```
proc Редактор1_ПриИзменении(E1: Edit);  
  -- по мере того, как пользователь заполняет поле,  
  -- выводим вводимую строку в качестве заголовка окна  
  self.Window.Caption = E1.Text;  
end;
```

Класс *Рисунок/Picture*, производный от класса *ОбъектШаблона*, предназначен для работы с изображениями форматов BMP, JPEG, ICO, EMF, WMF, являющимися интерфейсными элементами шаблонов Студии.

Внимание! Создать объект класса *Рисунок* из программы на ТБ.Скрипт нельзя. Объекты класса *Рисунок* создаются самой системой на основе информации о шаблоне, спроектированном в визуальном редакторе бланков Студии.

Класс *Рисунок* наследует все свойства и методы от родительских классов Объект и ОбъектШаблона.

Непосредственно в классе *Рисунок/Picture* определены следующие свойства:



Поле Изображение / Image



Поле Прозрачный / Transparent



Событие ПриВходе / OnEnter



Событие ПриВыходе / OnExit



Событие ПриНажатии / OnClick

Описание

Изображение : [Изображение](#);

Image : [Image](#);

Назначение

С помощью этого свойства можно напрямую задавать изображение, которое будет отрисовываться в объекте **Рисунок**. При этом, если рисунок связан с переменной типа [Изображение / Image](#), то данное поле обеспечивает доступ к ней (то есть позволяет изменить содержимое этой переменной). Когда рисунок не связан ни с одной переменной типа **Изображение**, то система создает внутренний объект типа **Изображение** и именно к нему обеспечивает доступ данное поле.

Поле доступно только на чтение.

Пример

```
Picture1.Image.Assign(Image23);
```

Описание

Прозрачный : Логическое;
Transparent : Logical;

Назначение

Поле позволяет установить режим, при котором фон изображения считается прозрачным. В качестве прозрачного цвета берется цвет левой верхней точки изображения.

Если данный флаг установлен, то фон делается прозрачным. В противном случае изображение выводится без всяких преобразований.

Пример

```
Picture1.Fit = TRUE;  
Picture1.Transparent = TRUE;  
Picture1.Load("C:\Windows\Logo.jpg");
```

Описание

ПриВходе: Строка;
OnEnter: String;

Назначение

Данное событие происходит, когда рисунок получает фокус ввода (то есть, становится текущим элементом управления на шаблоне). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB*. Поле **ПриВходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnEnter** (P: Picture);
P - объект класса **Рисунок**, с которым произошло событие.

Пример

```
proc Рисунок1_ПриВходе(P1: Picture);  
  -- при выделении картинки  
  -- подготавливаем всплывающую подсказку  
  -- на основе данных из записи  
  P1.Hint = ОписаниеКартинки;  
end;
```

Описание

ПриВыходе: Строка;
OnExit: String;

Назначение

Данное событие происходит, когда рисунок теряет фокус ввода (то есть, рисунок перестает быть текущим элементом управления). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши TAB. Поле **ПриВыходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnExit**(P: Picture);
P - объект класса **Рисунок**, с которым произошло событие.

Пример

```
proc Рисунок1_ПриВыходе(P1: Picture);  
  P1.Hint = " ";  
end;
```


Описание

ПриНажатии: Строка;
OnClick: String;

Назначение

Данное событие происходит по щелчку мыши на рисунке или по нажатию клавиши *Enter*, если рисунок выделен. Поле **ПриНажатии** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnClick** OnClick(F: String);
P - объект класса **Рисунок**, с которым произошло событие.

Пример








```
Рис1или2: Логическое := FALSE;  
-- массив изображений заполняется предварительно  
Image: Image[];  
  
proc Рисунок1_ПриНажатии(P1: Picture);  
  -- по щелчку в объекте рисунок сменяется изображение  
  Рис1или2 = NOT Рис1или2;  
  
  if Рис1или2 = FALSE then  
    P1.Image.Assign(Image[1]);  
  else  
    P1.Image.Assign(Image[2]);  
  end;  
end;
```

Класс *РядЗакладок/TabSet*, производный от класса *ОбъектШаблона*, позволяет создавать шаблоны с многостраничным интерфейсом. Следует отметить, что сам объект *РядЗакладок* не является контейнером для других элементов управления и не содержит "страницы" шаблона, хотя позволяет добиться похожего эффекта. Прикладной программист должен вручную запрограммировать изменение видимости тех или иных элементов шаблона в привязке к событиям, происходящим в ряду закладок.



Внимание! Создать объект класса *РядЗакладок* из программы на ТБ.Скрипт нельзя. Объекты класса *РядЗакладок* создаются самой системой на основе информации о шаблоне, спроектированном в визуальном редакторе бланков Студии.

Класс *РядЗакладок* наследует все свойства и методы от родительских классов [Объект](#) и [ОбъектШаблона](#).

Непосредственно в классе *РядЗакладок* определены следующие свойства:

-  [Поле Цвет / Color](#)
-  [Поле Шрифт / Font](#)
-  [Поле Индекс / Index](#)
-  [Поле Список / List](#)
-  [Поле РамкаЗакладка / TabsBorder](#)
-  [Поле СтилЗакладок / TabsStyle](#)
-  [Поле ПозицияЗакладок / TabsPosition](#)
-  [Событие ПриВходе / OnEnter](#)
-  [Событие ПриВыходе / OnExit](#)
-  [Событие ПриПереключении / OnSwitch](#)
-  [Событие ПриИзменении / OnChange](#)

Для указания стилей и расположения закладок предназначены перечислимые типы:

-  [СтилиЗакладок / TabsStyles](#)
-  [ПозицииЗакладок / TabsPositions.](#)

Константы способов размещения закладок

Перечислимый тип **ПозицииЗакладок** / **TabsPositions** включает константы, описывающие ориентацию закладок:

- ПозицияСнизу / AtBottom
- ПозицияСверху / AtTop
- ПозицияСлева / AtLeft
- ПозицияСправа / AtRight

Константы используются свойством [ПозицияЗакладок](#).

Константы стилей закладок (**СтилиЗакладок** / **TabsStyles**)

Перечислимый тип **СтилиЗакладок** / **TabsStyles** включает константы, описывающие внешний вид закладок:

- Трапецевидные / Trapeziums - трапецевидные закладки
- Прямоугольные / Rectangles - прямоугольные закладки
- Кнопки / Buttons - закладки в виде кнопок

Константы используются свойством [СтильЗакладок](#).

Описание

Индекс : Целое ;

Index : Integer ;

Назначение

Данное поле позволяет узнать индекс текущей выделенной закладки и изменить выделение, т.е. сделать выделенной другую закладку с указанным новым индексом. При присваивании этого свойства (только в том случае, если значение изменяется) возникает событие [Событие ПриИзменении](#).

Если с объектом **РядЗакладок** связана целочисленная переменная (см. диалог настроек свойств), то её изменение также приводит к изменению и свойства **Индекс**, но событие [Событие ПриИзменении](#) при этом не генерируется. По умолчанию интервал допустимых значений поля **Индекс** простирается от 1 до общего числа закладок. Однако программист может присвоить каждой закладке любой другой уникальный индекс, например, первой закладке – индекс 10, второй – 20, и так далее. Делается это с помощью специального именованного закладок посредством свойства [Список](#). Иными словами, индекс не обязательно совпадает с порядковым номером закладки в ряду.

Если в связанную переменную заносится значение, не укладывающееся в этот интервал, поле **Индекс** получает значение 0. При этом ни одна закладка не является выделенной.

Пример

```
TabSet1.Index = TabSet1.List.Count;
```

Описание

ПозицияЗакладок :РядЗакладок.ПозицииЗакладок;
TabsPosition :TabSet.TabsPositions;

Назначение

Поле позволяет узнать и изменить способ отображения закладок, соответствующий их [размещению](#) по отношению к управляемым элементам шаблона: закладки сверху, снизу, справа или слева.

Пример

```
TabSet1.TabsPosition = TabSet.AtBottom;
```

Описание

РамкаЗакладка :Логическое;
TabsBorder :Logical;

Назначение

Данное поле позволяет узнать и изменить настройку, определяющую, нужно ли выводить рамку вокруг объекта.

Пример

```
TabSet1.TabsBorder = true;
```

Описание

Список : [СписокСтрок](#);
List : [StringList](#);

Назначение

Данное поле позволяет узнать и изменить перечень закладок в объекте. С помощью методов объекта [СписокСтрок](#) можно удалять и добавлять закладки, а также корректировать надписи на закладках.

Каждая строка может содержать как просто заголовок закладки, так и заголовок вместе с подсказкой, которая будет выводиться при наведении курсора мыши на закладку. Для указания подсказки необходимо после заголовка добавить символ "|" (вертикальная черта) и после него текст подсказки. Например, "ОПУ|Отчет о прибылях и убытках". Следует отметить, что логика вывода подсказок может меняться в зависимости от настроек свойства [Подсказка/Hint](#). В частности, если для объекта **РядЗакладок** установлена какая-либо общая подсказка, т.е. свойство **Hint** непусто, то всегда выводится именно эта подсказка.

Кроме того, в заголовке закладки можно в явном виде указать индекс, который эта закладка должна иметь – здесь под словом индекс имеется в виду не порядковый номер закладки в ряду, а значение свойства [Индекс/Index](#). Требуемый индекс указывается в заголовке закладки после строки с подсказкой, отделенный от неё еще одной вертикальной чертой. Т.е. полный синтаксис заголовка имеет следующий вид:

<Название>|<Подсказка>|<Индекс>

Индексом закладки может быть любое целое положительное число. Если строка подсказки не используется, всё равно необходимо указать обе вертикальные черты:

<Название>||<Индекс>

Если индекс опущен у какой-либо закладки, то он по умолчанию назначается, равным значению, на 1 большему индекса предыдущей закладки. Если индекс опущен у первой закладки, он равен 1.

Пример

```
TabSet1.List.Add( "Настройки" );
```


Поле **СтильЗакладок** / **TabsStyle**

Описание

СтильЗакладок : [РядЗакладок](#).[СтилиЗакладок](#);
TabsStyle : [TabSet](#).[TabsStyles](#);

Назначение

Поле позволяет узнать и изменить [стиль закладок](#): трапециевидные, прямоугольные, в виде кнопок.

Пример

```
TabSet1.TabsStyle = TabSet.Trapeziums;
```

Поле Цвет / Color

Описание

Цвет : Целое;
Color :Integer;

Назначение

Данное поле позволяет задать цвет фона объекта. Этот цвет может отличаться от цвета поверхности закладки.

Пример

```
var Red, Green, Blue :Integer;  
Red = 128;  
Green = 128;  
Blue = 128;  
TabSet1.Color = Red + Green*256 + Blue*256*256;
```

Поле Шрифт / Font

Описание

Шрифт : [Шрифт](#) ;

Font : [Font](#) ;

Назначение

Позволяет узнать параметры шрифта текста на закладках. Поле доступно только на чтение.

Пример

```
TabSet1.Font.Bold = true;
```

Событие ПриВходе / OnEnter

Описание

ПриВходе: Строка;
OnEnter: String;

Назначение

Данное событие происходит, когда закладка получает фокус ввода (то есть, становится текущим элементом управления на шаблоне).

Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB*. Кроме того, закладка может получить фокус ввода, если пользователь выделил ее щелчком мыши.

Поле **ПриВходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnEnter**(Sender: TabSet);
Sender - объект класса **РядЗакладок/TabSet**, с которым произошло событие.

Пример

```
proc TabSet_OnEnter(Sender : TabSet);  
  TabSet1.Font.Bold = TRUE; -- описание текущей закладки делаем жирным  
end;
```

Событие ПриВыходе / OnExit

Описание

ПриВыходе: Строка;
OnExit: String;

Назначение

Данное событие происходит, когда закладка теряет фокус ввода (то есть, она перестает быть текущим элементом управления). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *Tab* или в результате выделения элемента мышью.

Поле **ПриВыходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnExit**(Sender: TabSet);
Sender - объект класса **РядЗакладок/TabSet**, с которым произошло событие.

Пример

```
proc TabSet_OnExit(Sender : TabSet);
  TabSet1.Font.Bold = FALSE;
  -- когда фокус ввода уходит с закладки,
  -- задаем отрисовку ее описания нежирным шрифтом
end;
```

Описание

ПриИзменении: Строка;
OnChange: String;

Назначение

Данное событие происходит после переключения закладок, инициированного щелчком мыши на одной из закладок или нажатием сочетания горячих клавиш *Alt*+<буква> (если в заголовке закладки с помощью символа '&' указана горячая клавиша, например, "&Отчет").

Поле **ПриИзменении** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnChange**(Sender :TabSet);
Sender - объект класса **РядЗакладок**, в котором произошло событие.

Пример

```
proc РядЗакладок_ПриИзменении(Ряд: РядЗакладок);  
  -- при выделении второй закладки  
  -- меняем особым образом видимость секций на шаблоне  
  if Ряд.Индекс = 2 then  
    Секция2.Visible = True;  
    Секция1.Visible = False;  
  end;  
end;
```

Описание

ПриПереключении : Строка;
OnSwitch : String;

Назначение

Данное событие происходит по щелчку мыши на одной из закладок, а также при нажатии сочетания горячих клавиш *Alt*+<буква>, если в заголовке закладки с помощью символа '&' указана горячая клавиша (например, "&Отчет").

Поле **ПриПереключении** позволяет узнать и изменить название прикладной функции-обработчика события.

Обработчик

func **OnSwitch**(Sender :TabSet; NewIndex: Integer) : Logical;

Sender - объект класса **РядЗакладок/TabSet**, с которым произошло событие.

NewIndex - номер закладки, на которой произошло нажатие, эта закладка станет выделенной, если это не запретить программно.

Функция возвращает значение "True", если выделение новой закладки разрешено, и "False" - в противном случае.

Если переключение произошло, то после события **ПриПереключении** будет сгенерировано событие ПриИзменении. По умолчанию переключение разрешено.

Пример

```
func РядЗакладок_ПриНПереключении(Ряд: РядЗакладок; Индекс :Целое) :Logical;  
    Result = true;  
    -- если в данный момент выделена первая закладка,  
    -- то пользователь не может сам переключиться с нее на другую  
    if Ряд.Индекс = 1 then  
        Result = False;  
    end;  
end;
```

Класс *Список/ComboBox*, производный от класса *ОбъектШаблона*, представляет собой программный интерфейс для элемента управления типа "выпадающий список", который используется в шаблонах Студии.

Внимание! Создать объект класса *Список* из программы на ТБ.Скрипт нельзя. Списки создаются самой системой на основе информации о шаблоне, спроектированном в визуальном редакторе бланков Студии.

Класс *Список* наследует все свойства и методы от родительских классов [Объект](#) и [ОбъектШаблона](#). Непосредственно в этом классе определены следующие свойства:

-  [Поле Цвет / Color](#)
-  [Поле Шрифт / Font](#)
-  [Поле Текст / Text](#)
-  [Поле Индекс / Index](#)
-  [Поле Список / List](#)

-  [Событие ПриВходе / OnEnter](#)
-  [Событие ПриВыходе / OnExit](#)
-  [Событие ПриИзменении / OnChange](#)

Описание

Индекс : Целое ;

Index : Integer ;

Назначение

Данное поле позволяет узнать номер пункта, который выбран из списка, и выбрать новый пункт. Строки в списке нумеруются с 1. Нулевой индекс означает, что ни одна из строк не выбрана.

Следует иметь в виду, что если со списком связана целочисленная переменная (без какой-либо трансляции значений), то значение данной переменной всегда равно (Index – 1), то есть в переменной строки нумеруются, начиная с 0, а значение -1 сигнализирует неопределенное состояние, когда ни одна строка не является текущей.

Пример

```
СписокВалют.Индекс = 1;
```

Описание

Список : [СписокСтрок](#);
List : [StringList](#);

Назначение

Данное свойство предназначено для программного управления перечнем строк в списке. Свойство содержит указатель на объект класса [СписокСтрок](#) с массивом строк, содержащихся в данный момент в выпадающем списке.

Для просмотра и изменения списка необходимо использовать свойства и методы класса [СписокСтрок](#).

Пример

```
ComboBox1.List.Clear;  
for i=1..КоличествоЦен do  
    ComboBox1.List.Add(Цена[i]);  
end;
```

Поле Текст / Text

Описание

Текст: Строка;

Text: String;

Назначение

Данное поле позволяет узнать, какой пункт (а именно, строка) выбран из списка, и выбрать новый пункт. Если присваиваемое полю значение совпадает со строкой одного из пунктов, он становится активным. В противном случае текущий выбор сбрасывается и выбранный до того пункт (если такой был) перестает быть активным.

Пример

```
Settings1 : String;  
proc ПриЗакрытииБланка(Режим:Logical);  
    Settings1 = ComboBox1.Text;  
end;
```

Описание

Цвет : Целое;
Color :Integer;

Назначение

Данное поле позволяет узнать и изменить цвет фона списка.

Пример

```
var R, G, B, C: Integer;  
-- составляющие цвета должны лежать в пределах 0-255  
R = Mod(Red,256); -- красный  
G = Mod(Green,256); -- зеленый  
B = Mod(Blue,256); -- синий  
C = R + G*256 + B*256*256;  
Список1.Цвет = C;
```

Поле Шрифт / Font

Описание

Шрифт : [Шрифт](#) ;

Font : [Font](#) ;

Назначение

Позволяет узнать параметры шрифта объекта Список. Поле доступно только на чтение.

Пример

```
Список1.Шрифт.Жирный = Истина;
```

Описание

ПриВходе: Строка;
OnEnter: String;

Назначение

Данное событие происходит, когда список становится текущим элементом управления на шаблоне. Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB*.

Кроме того, список может получить фокус ввода, если пользователь выделил его щелчком мыши. Поле **ПриВходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnEnter**(C: ComboBox);
C - объект класса **Список**, с которым произошло событие.

Пример

```
proc Список1_ПриВходе(C1: ComboBox);  
  Пояснение = "Из этого списка следует выбрать обозначение валюты";  
  -- выводим пояснение, когда пользователь собирается заполнить поле  
end;
```

Описание

ПриВыходе: Строка;
OnExit: String;

Назначение

Данное событие происходит, когда список теряет фокус ввода (то есть, список перестает быть текущим элементом управления). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB* или в результате выделения элемента мышью.

Поле **ПриВыходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnExit**(C: ComboBox);
C - объект класса **Список**, с которым произошло событие.

Пример

```
proc Список1_ПриВыходе(C1: ComboBox);  
    Пояснение = ""; -- убираем пояснение, когда курсор уходит с поля  
end;
```

Описание

ПриИзменении: Строка;
OnChange: String;

Назначение

Данное событие происходит, когда текст в редакторе списка изменен (путем ввода с клавиатуры или выбора из списка). Новое содержимое редактора списка можно определить с помощью свойства [Текст](#).

Поле **ПриИзменении** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnChange**(C: ComboBox);
C - объект класса **Список**, с которым произошло событие.

Пример

```
proc Список1_ПриИзменении(C1: ComboBox);  
  -- копируем новое значение в другое поле ввода  
  Edit1.Text = C1.Text;  
end;
```


Класс *Таблица/Grid*, производный от класса *ОбъектШаблона*, предназначен для работы с объектом *Таблица*, используемым в шаблонах Студии. Таблица может быть как иерархической, содержащей группы и элементы, так и обычной.

В классе *Таблица* можно использовать все свойства и методы, определенные в родительских классах [Объект](#) и [ОбъектШаблона](#).

Внимание! Создать объект класса *Таблица/Grid* из программы, написанной на языке ТБ.Скрипт, нельзя. Таблицы создаются самой системой на основе информации о шаблоне, спроектированном в визуальном редакторе шаблонов.

Непосредственно в классе *Таблица/Grid* определены следующие свойства:

-  [Поле Цвет/Color](#)
-  [Поле Шрифт / Font](#)
-  [Поле Бордюр / Bevel](#)
-  [Поле Корень / Root](#)
-  [Поле Текущий / Current](#)
-  [Поле ТекущаяСтрока / CurrentRow](#)
-  [Поле ПоказыватьЗаголовок / ShowHeader](#)
-  [Поле ПоказыватьИконки / ShowIcons](#)
-  [Поле ПоказыватьКорень / ShowRoot](#)
-  [Поле ПоказыватьСетку / ShowGrid](#)
-  [Поле КоличествоСтолбцов / ColumnsCount](#)
-  [Поле Столбец / Column](#)
-  [Функция ДобавитьСтолбец / AddColumn](#)
-  [Функция ВставитьСтолбец / InsertColumn](#)
-  [Процедура УдалитьСтолбец / DeleteColumn](#)
-  [Процедура УпорядочитьПо / SortBy](#)
-  [Функция ТекущийСтолбец / CurrentColumn](#)
-  [Поле ИндексСтолбца / CurrentColumnPos](#)
-  [Процедура НачатьМодификацию / BeginModify](#)
-  [Процедура ЗавершитьМодификацию / EndModify](#)
-  [Событие ПриВходе / OnEnter](#)
-  [Событие ПриВыходе / OnExit](#)
-  [Событие ПриНажатии / OnClick](#)
-  [Событие ПриПеремещении / OnMove](#)
-  [Событие ПриНажатииЗаголовка / OnHeaderClick](#)

Поле Бордюор / Bevel

Описание

Бордюор : [Шаблон.СтилиБордюора](#) ;
Bevel : [Template.BevelStyles](#) ;

Назначение

Свойство позволяет определить или установить вокруг таблицы один из доступных стилей бордюра, который определен в перечислимом типе [СтилиБордюора/BevelStyles](#).

При выборе значения *NoneBevel* бордюор вокруг таблицы отсутствует, в остальных случаях бордюор прорисовывается одним из следующих стилей:

- по умолчанию | DefaultBevel
- тонкий | SingleBevel
- статический | StaticBevel
- внутренний | ClientBevel
- оконный | WindowBevel

Поле доступно на чтение и запись.

Описание

`ИндексСтолбца` : Целое;
`CurrentColumnPos` : Integer;

Назначение

Данное свойство позволяет узнать индекс текущего (видимого) столбца таблицы, а также сделать текущим другой видимый столбец. *Текущим* (выделенным) считается столбец, в котором находится курсор. Свойство доступно на чтение и запись.

Внимание! Свойство **ИндексСтолбца**=CurrentColumn.Index, только в том случае, когда видимыми являются все столбцы таблицы. *В общем случае номер видимого столбца ИндексСтолбца не совпадает с порядковым номером столбца картотеки CurrentColumn.Index.*

Валидность диапазона при установке свойства не проверяется, поэтому, чтобы встать на последний столбец можно написать CurrentColumnPos = 999; или задать общее [количества столбцов](#) (см. пример ниже).

Пример

```
Grid1 : Grid;  
-- ...  
Grid1.CurrentColumnPos = 1;
```

Описание

КоличествоСтолбцов : Целое;
ColumnsCount : Integer;

Назначение

Данное свойство возвращает количество столбцов в таблице. Поле доступно только на чтение.

Для программного добавления/удаления столбцов необходимо пользоваться методами [ДобавитьСтолбец](#), [ВставитьСтолбец](#), [УдалитьСтолбец](#).

Пример

```
Grid1: Grid; -- объект шаблона
-- выводим имена полей всех столбцов
proc TraceColumns(Name: String);
var i: integer;
    for i=1..Grid1.ColumnsCount do
        trace(Grid1.Column[i].FieldName);
    end;
end;
```

Поле Корень / Root

Описание

Корень : [ЭлементТаблицы](#);

Root : [GridItem](#);

Назначение

Позволяет получить интерфейс для заполнения таблицы элементами и группами. Разыменовывает хранящуюся в данном поле ссылку на внутренние свойства класса [ЭлементТаблицы](#) / [GridItem](#).

Поле доступно только на чтение.

Пример

```
Grid1 :Grid;
-- ...
var vParent :GridItem;
-- ...
vParent = Grid1.Root;
-- ... заполняем таблицу элементами...
```

Описание

```
ПоказыватьЗаголовок : Logical;  
ShowHeader : Logical;
```

Назначение

Логическое поле позволяет узнать и изменить значение, отвечающее за отображение заголовка таблицы.

Поле доступно как на чтение, так и на запись.

Описание

`ПоказыватьИконки` : Logical;
`ShowIcons` : Logical;

Назначение

Поле позволяет узнать и изменить значение, отвечающее за отображение иконок у элементов таблицы.

Поле доступно как на чтение, так и на запись.

Описание

```
ПоказыватьКорень : Logical;  
ShowRoot : Logical;
```

Назначение

Поле позволяет узнать и изменить значение, отвечающее за отображение элемента раскрытия папки ("плюсиков") у корневых папок таблицы.

Поле доступно как на чтение, так и на запись.

Описание

```
ПоказыватьСетку : Logical;  
ShowGrid : Logical;
```

Назначение

Свойство позволяет узнать и изменить значение, отвечающее за отображение сетки в таблице. Другими словами, отвечает за отображение линий, разграничивающих элементы и столбцы таблицы, аналогично картотеке.

Поле доступно как на чтение, так и на запись.

Описание

Столбец[Индекс: Целое] :[СтолбецТаблицы](#);
Column [Index: Integer] :[TableColumn](#);

Аргументы

Индекс - задает позицию, в которую будет вставлен новый столбец. Индекс может изменяться в пределах от 1 до [количества столбцов](#).

Назначение

Данное поле позволяет получить доступ к конкретному столбцу таблицы по его номеру. Поле доступно только на чтение.

Пример

```
Grid1: Grid; -- объект шаблона
-- симулируем метод ColumnByField
func FieldByName(Name: String):TableColumn;
var i: integer;
  for i=1..Grid1.ColumnsCount do
    if Name = Grid1.Column[i].FieldName then
      return Grid1.Column[i];
    end;
  end;
  return nil;
end;
```

Описание

```
ТекущаяСтрока : Integer;  
CurrentRow : Integer;
```

Назначение

Поле позволяет узнать и изменить индекс текущей строки таблицы. Причем, если таблица иерархического вида, то поле возвращает индекс "видимой" строки, т.е. учитываются строки с элементами, находящимися только в раскрытой папке.

Поле доступно как на чтение, так и на запись.

Пример

```
Grid1 :Grid;  
-- ...  
Grid1.CurrentRow = Grid1.CurrentRow + 1;  
-- ...
```

Поле Текущий / Current

Описание

Текущий : [ЭлементТаблицы](#);

Current : [GridItem](#);

Назначение

Данное свойство позволяет узнать позицию текущего элемента таблицы, а также позиционировать на необходимый элемент таблицы. Причем, если таблица иерархического вида, то спозиционировать на соответствующий элемент можно с раскрытием до необходимой папки.

Поле доступно как на чтение, так и на запись.

Пример

```
Grid1 :Grid;  
-- ...  
var vItem :GridItem;  
-- ...  
vItem = Grid1.Current;  
-- ...
```

Описание

Цвет : Целое;
Color :Integer;

Назначение

Данное поле позволяет узнать и изменить цвет фона под объектом.

Пример

```
var R, G, B, C: Integer;  
-- составляющие цвета должны лежать в пределах 0-255  
R = Mod(Red,256); -- красный  
G = Mod(Green,256); -- зеленый  
B = Mod(Blue,256); -- синий  
C = R + G*256 + B*256*256;  
Grid1.Color = C;
```

Поле Шрифт / Font

Описание

Шрифт : [Шрифт](#) ;

Font : [Font](#) ;

Назначение

Позволяет узнать параметры шрифта объекта **Таблица**.

Поле доступно только на чтение.

Пример

```
Таблица1.Шрифт.Жирный = Истина;
```

Описание

```
ЗавершитьМодификацию;  
EndModify;
```

Назначение

Процедура разрешает обновление (перерисовку) таблицы, которое по умолчанию выполняется системой автоматически при любых программных изменениях его полей, но было запрещено предыдущим вызовом [НачатьМодификацию](#).

Следует иметь в виду, что при каждом вызове **НачатьМодификацию** система увеличивает внутренний счетчик блокировки таблицы и уменьшает его при вызове **ЗавершитьМодификацию**. Система возвращается к стандартному режиму автоматической перерисовки бланка только в тот момент, когда счетчик становится равным нулю.

Использовать процедуру имеет смысл после фрагмента кода, выполняющего массивованный пересчет переменных бланка (и записи, в случае бланка-редактора).

Пример

```
-- при нажатии кнопки начинаем длительный пересчет  
proc OnButtonClick(Button1 :Button);  
  -- запрещаем перерисовку бланка  
  BeginModify;  
  -- вызываем пользовательскую функцию  
  -- после всех изменений разрешаем перерисовку  
  EndModify;  
end;
```

Описание

НачатьМодификацию;
BeginModify;

Назначение

Процедура запрещает обновление (перерисовку) таблицы, которое по умолчанию выполняется системой автоматически при любых программных изменениях в таблице. Использовать процедуру имеет смысл перед началом фрагмента кода, выполняющего массивированный пересчет переменных бланка (и записи, в случае бланка-редактора). После выполнения вычислений следует вызвать парную процедуру [ЗавершитьМодификацию](#).

При каждом вызове **НачатьМодификацию** система увеличивает внутренний счетчик блокировки таблицы и уменьшает его при вызове **ЗавершитьМодификацию**. Когда счетчик становится равным нулю, система вновь возвращается к стандартному режиму автоматической перерисовки бланка в ответ на модификацию любого поля.

Пример

```
-- при нажатии кнопки начинаем длительный пересчет
proc OnButtonClick(Button1 :Button);
-- запрещаем перерисовку бланка
  BeginModify;
-- вызываем пользовательскую функцию
  EndModify;
end;
```


Описание

```
УдалитьСтолбец(Номер :Целое);  
DeleteColumn(Position :Integer);
```

Аргументы

Номер - задает номер столбца, который следует удалить. Номер должен лежать в пределах от 1 до [количества столбцов](#).

Назначение

Процедура удаляет указанный столбец из картотеки.

Пример

```
Grid1: Grid; -- объект шаблона  
-- ...  
-- удаляем первый столбец  
Grid1.DeleteColumn(1);
```

Описание

ПриВходе: Строка;
OnEnter: String;

Назначение

Данное событие происходит, когда таблица шаблона получает фокус ввода (то есть, становится текущим элементом управления на шаблоне). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB* или по щелчку мыши. Поле **ПриВходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnEnter** (Sender :Grid);

Sender - объект класса **Таблица**, с которым произошло событие.

Пример

```
proc Таблица_ПриВходе(Grid1 :Grid);  
    Grid1.Font.Bold = TRUE;  
end;
```

Описание

ПриВыходе: Строка;
OnExit: String;

Назначение

Данное событие происходит, когда таблица теряет фокус ввода (то есть, перестает быть текущим элементом управления). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *TAB* или по щелчку мыши. Поле **ПриВыходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnExit**(Sender :Grid);
Sender - объект класса **Таблица**, с которым произошло событие.

Пример

```
proc Таблица_ПриВыходе(Grid1 :Grid);  
    Grid1.Font.Bold = FALSE;  
end;
```

Описание

ПриНажатии: Строка;
OnClick: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при одиночном или двойном щелчке мышью в ячейке таблицы на шаблоне или при нажатии клавиши **Enter**, когда фокус ввода находится в этой таблице.

Обработчик

```
func OnClick(Sender :Grid; Action : Template.ClickTypes) :Logical;
```

Параметры:

Sender - указатель на объект класса **Таблица**, с которым произошло событие;

Action - одна из предопределенных констант, описанных в типе [Шаблон.ТипыНажатия](#), которая определяет действие пользователя:

- 0** - одинарный щелчок мышью;
- 1** - двойной щелчок мышью;
- 2** - нажатие клавиши **Enter**;

Если функция возвращает True, происходит стандартная обработка события. Если функция возвращает False, стандартная обработка не выполняется.

Пример

```
-- обработчик события "щелчок мышью в таблице"  
func GridOnClick(Grid1 :Grid; Action :Integer):Logical;  
  if Action = 0 then  
    Hint("Идет обработка данных...");  
  end;  
  return Ложь;  
end;
```

Описание

ПриНажатииЗаголовка: Строка;
OnHeaderClick: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при нажатии заголовка в колонке таблицы.

Обработчик

```
func OnHeaderClick(Sender :Grid; Action :Integer; Column :TableColumn): Logical;
```

Параметры:

Sender - указатель на объект класса **Таблица**, с которым произошло событие;

Action - не используется;

Column - столбец таблицы, с которым происходит событие.

Если функция возвращает True, происходит стандартная обработка события - сортировка столбца. Если функция возвращает False, стандартная обработка не выполняется.

Пример

```
func Grid1_OnHeaderClick(Sender :Grid; Action :Integer; Column :TableColumn): Logical;  
    Trace ("Заголовок столбца-"+Str(Column.Caption));  
    -- ...  
    Result = True;  
end;
```

Описание

ПриПеремещении: Строка;
OnMove: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при перемещении текстового курсора таблицы от одного элемента к другому.

Обработчик

proc **OnMove**(Sender :Grid);

Параметры:

Sender - указатель на объект класса **Таблица**, с которым произошло событие;

Событие позволяет управлять состоянием и значением полей таблицы в зависимости от контекста ввода.

Описание

ВставитьСтолбец(Номер :Целое) :[СтолбецТаблицы](#);
InsertColumn(Position :Integer) :[TableColumn](#);

Аргументы

Номер - задает позицию, в которую будет вставлен новый столбец. Номер должен лежать в пределах от 1 до [количества столбцов](#).

Назначение

Функция вставляет новый столбец в карточку. Позиция нового столбца задается аргументом **Номер**.

Пример

```
Grid1: Grid; -- объект шаблона
proc PlusColumn1(Fieldname :String);
var Column: TableColumn;
  -- создаем столбец, который должен располагаться первым
  Column = Grid1.InsertColumn(1);
  -- настраиваем столбец
  Column.FieldName = Fieldname;
  Column.Width = 200;
  -- ...
end;
```

Описание

ДобавитьСтолбец : [СтолбецТаблицы](#);
AddColumn : [TableColumn](#);

Назначение

Функция добавляет новый столбец в таблицу. Столбец становится последним в массиве столбцов.

Пример

```
Grid1: Grid; -- объект шаблона
proc PlusColumn(FieldName :String);
var Column: TableColumn;
  -- создаем столбец и получаем ссылку на новый объект
  Column = Grid1.AddColumn;
  -- настраиваем столбец
  Column.FieldName = FieldName;
  Column.Width = 200;
  -- ...
end;
```


Описание

ТекущийСтолбец : [СтолбецТаблицы](#);

CurrentColumn : [TableColumn](#);

Назначение

Функция позволяет получить ссылку на столбец таблицы, содержащий выделенную клетку.

Внимание! При попытке выделить невидимый столбец или столбец, отсутствующий в картотеке, текущая позиция *не* меняется.

Пример





```
Grid1 : Grid;  
...  
Grid1.CurrentColumn.Font.Bold = TRUE;
```





Флаг / CheckBox

Класс *Флаг/CheckBox* представляет собой программный интерфейс для элемента управления типа "флаг", используемого в шаблонах Студии и наследует все свойства и методы от родительских классов [Объект](#) и [ОбъектШаблона](#).

Внимание! Создать объект класса *Флаг* из программы на ТБ.Скрипт нельзя. Флаги создаются самой системой на основе информации о шаблоне, спроектированном в визуальном редакторе бланков Студии.

Непосредственно в классе *Флаг* определены следующие свойства:

-  [Поле Надпись / Caption](#)
-  [Поле Шрифт / Font](#)
-  [Поле СостояниеState](#)
-  [Поле Цвет / Color](#)

-  [Событие ПриВходе / OnEnter](#)
-  [Событие ПриВыходе / OnExit](#)
-  [Событие ПриИзменении / OnChange](#)
-  [Событие ПриНажатии / OnClick](#)

Поле Надпись / Caption

Описание

Надпись :Строка;
Caption :String;

Назначение

Данное поле позволяет узнать и изменить текст, который выводится в рабочей области флага.

Пример

```
proc ПриИзмененииФлага1(O:String);  
  if Флаг1.Состояние then  
    Флаг1.Caption = "Цена с НДС";  
  else  
    Флаг1.Caption = "Цена без НДС";  
  end;
```

Поле Состояние / State

Описание

Состояние :Логическое;
State :Logical;

Назначение

Данное поле позволяет узнать и установить состояние флага: включенное или выключенное.

Пример

```
proc ПриИзмененииФлага1(O:String);  
  if Флаг1.Состояние then  
    Флаг2.State = TRUE;  
  end;  
end;
```

Описание

Цвет : Целое;
Color :Integer;

Назначение

Данное поле позволяет узнать и изменить цвет фона под объектом.

Пример

```
var R, G, B, C: Integer;  
-- составляющие цвета должны лежать в пределах 0-255  
R = Mod(Red,256); -- красный  
G = Mod(Green,256); -- зеленый  
B = Mod(Blue,256); -- синий  
C = R + G*256 + B*256*256;  
Флаг1.Цвет = C;
```

[Класс Объект](#) : [ОбъектШаблона](#) : [Флаг](#)

Поле Шрифт / Font

Описание

Шрифт : [Шрифт](#) ;

Font : [Font](#) ;

Назначение

Позволяет узнать параметры шрифта объекта Флаг. Поле доступно только на чтение.

Пример

```
Флаг1.Шрифт.Жирный = Истина;
```

Описание

ПриВходе: Строка;
OnEnter: String;

Назначение

Данное событие происходит, когда флаг получает фокус ввода (то есть, становится текущим элементом управления на шаблоне). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *Tab*. Кроме того, флаг может получить фокус ввода, если пользователь выделил его щелчком мыши. Поле **ПриВходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnEnter**(C: CheckBox);
C - объект класса **Флаг**, с которым произошло событие.

Пример

```
proc Флаг1_ПриВходе(C1: CheckBox);  
  C1.Font.Bold = TRUE;  
  -- описание текущего флага делаем жирным  
end;
```

Описание

ПриВыходе: Строка;
OnExit: String;

Назначение

Данное событие происходит, когда флаг теряет фокус ввода (то есть, флаг перестает быть текущим элементом управления). Перемещение фокуса между элементами управления осуществляется по нажатию клавиши *Tab* или в результате выделения элемента мышью. Поле **ПриВыходе** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnExit**(C: CheckBox);
C - объект класса **Флаг**, с которым произошло событие.

Пример

```
proc Флаг1_ПриВыходе(C1: CheckBox);  
  C1.Font.Bold = FALSE;  
  -- когда фокус ввода уходит с флага,  
  -- задаем отрисовку его описания нежирным шрифтом  
end;
```


Описание

ПриИзменении: Строка;
OnChange: String;

Назначение

Данное событие происходит, когда флаг меняет свое состояние (с включенного на выключенное и наоборот), причем изменение может быть вызвано как программно, так и пользователем. Если состояние изменил пользователь, то данное событие происходит до события [ПриНажатии](#). Новое состояние флага можно определить с помощью свойства [Состояние](#).

Поле **ПриИзменении** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик

proc **OnChange** (C: CheckBox);
C - объект класса **Флаг**, с которым произошло событие.

Пример

```
proc Флаг1_ПриИзменении(C1: CheckBox);  
  -- в зависимости от состояния флага  
  if NOT C1.State then  
    Button1.Enabled = TRUE;  
    -- делаем кнопку доступной  
  else  
    Button1.Enabled = FALSE;  
    -- делаем кнопку недоступной  
  end;  
end;
```

Описание

ПриНажатии: Строка;
OnClick: String;

Назначение

Данное событие происходит при нажатии флага (щелчок мышью или клавиша *Пробел* на клавиатуре). Событие происходит после события [ПриИзменении](#).

Поле **ПриНажатии** позволяет узнать и изменить название прикладной процедуры-обработчика события.

Обработчик





proc **OnClick**(Ch: CheckBox);
Ch - объект класса **Флаг**, с которым произошло событие.

Пример

```
proc Флаги1и2_ПриНажатии(CB: CheckBox);  
  if CB = CB1 then  
    CB2.State = NOT CB2.State;  
  else  
    CB1.State = NOT CB1.State;  
  end;  
end;
```

Класс *РедакторПоля* / *FieldEdit*, производный от родительского класса [Объект](#), используется для программной вставки или замены текста (например, для вставки макросов) в текущий редактор, ссылка на который хранится в поле [ТекущийРедактор](#). Причем ссылка равна nil, если редактирование не начато.

Непосредственно в классе *РедакторПоля* определены следующие свойства:

-  [Поле Текст / Text](#)
-  [Поле ВыделенныйТекст / SelText](#)
-  [Поле НачалоВыделения / SelStart](#)
-  [Поле ДлинаВыделения / SelLength](#)

и события:

-  [Событие ПриПоиске / OnSearch](#)

Описание

ВыделенныйТекст : Строка;
SelText : String;

Назначение

Свойство задать и узнать текст, выделенный [в текущем редакторе](#). Начальная позиция выделенного текста и количество выделенных символов хранится соответственно в свойствах [НачалоВыделения](#) и [ДлинаВыделения](#).

При вставке нового текста в текущий редактор выделенный текст заменяется на новый.

Пример

```
func ПолеПараметрМаскаСчетов_ПриОбзоре(Cell :TemplateCell;  
    Value :String; var NewValue :String) :Logical;  
    var локСчет :Account;  
    var локИмяСчета :String;  
    локСчет = nil;  
  
    if ChooseAccount(локСчет, Report.AccountPlan +  
        ':' + Report.AccountFilter) = cmOK then  
        локИмяСчета = локСчет.Name;  
        if (Template.CurrentCell = Cell) and  
            (Template.CurrentEdit <> nil) then  
            with Template.CurrentEdit do  
                SelText = локИмяСчета;  
                NewValue = Text;  
            end;  
        else  
            NewValue = Value + локИмяСчета;  
        end;  
    end;  
    Result = False;  
end;
```

Описание

ДлинаВыделения : Целое;
SelLength : Integer;

Назначение

Свойство позволяет задать или считать длину (количество символов) выделенного [в текущем редакторе](#) текста. После вставки текста курсор позиционирует в конец добавленного текста.

Пример

См. [Пример работы с объектом РедакторПоля](#).

Описание

НачалоВыделения : Целое;
SelStart : Integer;

Назначение

Свойство позволяет задать или считать текущую позицию курсора, если [в текущем редакторе](#) отсутствует выделенный текст, при его наличии в данном свойстве хранится начальная позиция выделенного текста. Причем, если за курсором имеется текст, то он записывается после вставленного текста. Если имеется выделенный текст, то он заменяется новым текстом, иначе новый текст просто вставляется в заданную позицию, ничего не удаляя.

Пример

См. [Пример работы с объектом РедакторПоля](#).

Описание

Текст : Строка;

Text : String;

Аргументы

Свойство позволяет программно записать текст [в текущий редактор](#) или считать хранящийся в нем текст.

При записи текст вставляется, начиная с текущей позиции курсора. При отсутствии в поле выделенного текста, исходный текст раздвигается, если после курсора имеется текст, иначе новый текст вставляется за исходным текстом. При наличии выделенного текста, он заменяется новым текстом.

Пример

См. [Пример работы с объектом РедакторПоля](#).

```

inclass
  stored var S :String; -- текущий редактор

inobject
  Edit1 :Edit; -- объект Редактор
  Edit :FieldEdit;

  -- событие при входе в текущий редактор
  func FieldS_OnEnter(Cell :TemplateCell;
    Index :Integer; Action :Template.EnterTypes) :Logical;
    Edit = Template.CurrentEdit;
    Result = True;
  end;

  -- событие при обзоре
  func FieldS_OnLookup(Cell :TemplateCell;
    Value :String; var NewValue :String) :Logical;
    Command1_OnExecute(nil);
    NewValue = Template.CurrentEdit.Text;
    Result = False;
  end;

  -- событие ПриНажатии кнопки
  proc ButDate_OnClick(Sender :Button); -- Date (Ctrl+3)
    Command3_OnExecute(nil);
  end;

  -- добавление текста в текущий редактор
  -- путем выбора из справочника
  func Command1_OnExecute(Command :Command) :Logical;
    var Doc :Record;
    EdtPrepare;
    if Справочники.Товары.ShowFormEx(Doc, '', Window.ModalWindow) = cmOk then
      if Template.CurrentObject = Edit1 then
        Edit1.SelText = '{' + Str(Doc.Название) + '}';
      elsif Template.CurrentEdit <> nil then
        Template.CurrentEdit.SelText = '{' + Str(Doc.Название) + '}';
      end;
    end;
  end;

  -- добавление текста в текущий редактор
  func Command3_OnExecute(Command :Command) :Logical;
    EdtPrepare;
    if Template.CurrentObject = Edit1 then
      Edit1.SelText = '{Date}';
    elsif Template.CurrentEdit <> nil then
      Template.CurrentEdit.SelText = '{Date}';
    end;
  end;

  proc EdtPrepare;
    if Template.CurrentObject = Edit1 then
      -- Ok
    elsif Template.CurrentEdit = nil then
      Template.Field = "S"; -- текущий редактор
      Template.BeginEdit;
      Template.CurrentEdit.SelStart = Length(Template.CurrentEdit.Text);
    end;
  end;

```


Описание

ПриПоиске : Строка;
OnSearch : String;

Назначение

Данное свойство узнать и изменить название прикладной процедуры-обработчика события **ПриПоиске**. Событие используется для изменения запроса поиска и происходит в ячейке шаблона при нажатии клавиши **Enter** или выходе из ячейки.

Обработчик

func **OnSearch**(AEditor : [FieldEdit](#); AQuery : [Query](#); Action : [SearchActions](#) :Logical;

Параметры:

AEditor - объект класса **РедакторПоля**, с которым произошло событие.

AQuery - объект запроса поиска;

Action - действие, вызывающее это событие. Параметр **Action** может принимать одно из значений перечислимого типа *ДействияИзменения*. Если значение равно Template.ActionExit, то событие происходит при выходе из ячейки, если - Template.ActionEnter, то - при нажатии клавиши **Enter**.

Функция событию присваивается динамически, когда доступен объект [Template.CurrentEdit](#) класса FieldEdit, обычно на событии ячейки **ПриВходе**.

Событие возвращает значение логического типа. Если значение равно true, то запрос выполнится автоматически, т.е. произойдет Query.Select. При возвращении return False, запрос не будет выполняться и, если AQuery не будет открыт пользователем программно, поиск в ячейке будет отменен!

Пример

```
func Поле_ПриВходе(Cell :TemplateCell; Index :Integer) :Logical;
-- присваиваем функцию "EditorOnSearch" событию OnSearch
Template.CurrentEdit.OnSearch = "EditorOnSearch";
return True;
end;

func EditorOnSearch(AEditor: FieldEdit; AQuery: Query;
Action :Template.SearchActions) :Logical;

-- если выполняется действие вызова Template.ActionExit, значит
-- поиск будет осуществлен по окончании AEditor.Textт.е. будет
-- отображена строка, которая заканчивается текстом редактора.
if Action = Template.ActionExit then
AQuery.Filter = 'MATCH (Название, "*" +AEditor.Text+"")';
Trace("ActionExit: " + AQuery.Filter);
else
-- а если выполняется действие вызова Template.ActionExit (выход из
-- ячейки), то поиск будет осуществлен по вхождению AEditor.Text,
-- т.е. будет отображена строка, в которую входит текст редактора.
AQuery.Filter = 'MATCH (Название, "*" +AEditor.Text+"*")';
Trace("ActionEnter: " + AQuery.Filter);
end;
-- самостоятельное выполнение запроса.
AQuery.Select;
-- отмена автоматического выполнение запроса.
return false;
end;
```

Класс *СекцияШаблона* (*Секция*, *TemplateSection*, *Section*), производный от родительского класса [Объект](#), используется для работы с [секциями шаблонов](#).

Внимание! Создать объект класса *СекцияШаблона* напрямую нельзя. Секции шаблонов создаются только внутри системы и используются как свойства других объектов (класса [Шаблон/Template](#)), причем такие свойства имеют атрибут "только чтение". Вместе с тем, программа на ТБ.Скрипт может получить доступ к объекту *СекцияШаблона*, являющемуся свойством родительского объекта, и изменять свойства секции.

Непосредственно в классе *СекцияШаблона* определены следующие свойства:

-  [Поле Имя / Name](#)
-  [Поле Владелец / Owner](#)
-  [Поле РодИндекс / ParentIndex](#)
-  [Поле Видна / Visible](#)
-  [Поле Печатать / Printed](#)
-  [Поле РазрешеноПовторение / EnableRepeat](#)
-  [Поле МожноВставлятьКадр / CanInsertFrame](#)
-  [Поле МожноУдалятьКадр / CanDeleteFrame](#)
-  [Поле МожноПеремещатьКадр / CanMoveFrame](#)
-  [Поле ИмяПодтаблицы / SubtableName](#)
-  [Поле ФильтрПодтаблицы / SubtableFilter](#)
-  [Поле Подтаблица / Subtable](#)
-  [Поле КоличествоКадров / FramesCount / Количество / Count](#)
-  [Поле КоличествоСтрок / RowsCount](#)
-  [Поле КоличествоСтолбцов / ColumnsCount](#)
-  [Поле Столбец / Column](#)
-  [Поле Строка / Row](#)
-  [Поле Клетка / Cell](#)
-  [Поле КлеткаПоПолю / CellByField](#)

-  [Процедура ВставитьКадр / InsertFrame](#)
-  [Процедура УдалитьКадр / DeleteFrame](#)
-  [Процедура КадрВверх / FrameUp](#)
-  [Процедура КадрВниз / FrameDown](#)
-  [Процедура УпорядочитьПо / SortBy](#)
-  [Функция СуммаПоля / SumOfField](#)
-  [Процедура ОбъединитьКлетки / LinkCells](#)

-  [Событие ПослеВставки / AfterInsert](#)
-  [Событие ПриВставке / OnInsert](#)
-  [Событие ПослеУдаления / AfterDelete](#)
-  [Событие ПриУдалении / OnDelete](#)
-  [ПриОформлении / OnRearrange](#)
-  [Событие ПриСменеПозиции / OnChangePosition](#)

Описание

Видна : Логическое;
Visible : Logical;

Назначение

Позволяет узнать, видна ли данная секция шаблона, а также управлять ее видимостью, присваивая полю значения TRUE (секция видна) или FALSE (секция скрыта). Когда устанавливается значение данного свойства, синхронно изменяется и свойство [Печатать](#).

Пример

```
if КодОперации = кодДенежныйКредит then  
    СекцияПереченьТМЦ.Visible = FALSE;  
end;
```

Описание

Владелец : Шаблон;
Owner : Template;

Назначение

Поле позволяет получить ссылку на шаблон, в котором расположена данная секция. Поле доступно только на чтение.

Пример

```
Пример
-- на вход подается секция,
-- требующая громоздких вычислений при заполнении;
-- когда секция обрабатывается, обращаемся к владельцу,
-- чтобы временно запретить перерисовку
proc ProcessSection(aSection :TemplateSection);
  aSection.Owner.BeginModify;
  -- вычисления и заполнение полей секции
  -- ...
  aSection.Owner.EndModify;
end;
```

Поле Имя / Name

Описание

Имя : Строка;
Name : String;

Назначение

Поле позволяет узнать и изменить имя секции шаблона. Оно устанавливается в визуальном редакторе бланков.

Пример

```
-- выводим имя секции бланка, которая является текущей (то есть  
-- содержит активное поле)  
message(Template.CurrentSection.Name);
```

Описание

ИмяПодтаблицы : Строка;
SubtableName : String;

Назначение

Свойство позволяет для [повторяющейся](#) секции, размещенной на шаблоне бланка-редактора, установить имя подтаблицы редактируемого класса записи, которая (имеется в виду подтаблица) должна выводиться в этой секции.

Если имя подтаблицы не задано, секция может заполняться из какого-либо другого источника, например, программно или из массивов.

В визуальном редакторе шаблонов данное свойство вводится с помощью поля **Подтаблица** в диалоге ["Свойства секции"](#).

Пример

```
СекцияИзделие.РазрешеноПовторение = TRUE;  
СекцияИзделие.ИмяПодтаблицы = "Детали";
```

Описание

Клетка[Столбец:Целое; Строка:Целое] : [КлеткаШаблона](#);
Cell[Column:Integer; Row:Integer] : [TemplateCell](#);

Аргументы

Столбец - номер столбца;
Строка - номер строки.

Назначение

Позволяет получить доступ к [клетке секции](#), указанной с помощью пары координат. Координаты верхней левой клетки равны [1; 1]. Максимальный номер столбца равен [ColumnsCount](#). Максимальный номер строки равен [RowCount](#) * [FramesCount](#), для повторяющихся секций или **RowCount** для простых.

Пример

```
var ThisCell: TemplateCell;  
var Sec: TemplateSection;  
var k, j: integer;  
  
for k=1..Sec.ColumnsCount do  
  for j=1..Sec.RowsCount do  
    ThisCell=Sec.Cell[k,j];  
    -- ...  
  end;  
end;
```

См. также [Пример работы с секциями, столбцами, строками и клетками шаблонов](#).

Описание

`КлеткаПоПолю`[Имя:Строка] : [КлеткаШаблона](#);
`CellByField`[Name:String] : [TemplateCell](#);

Аргументы

Имя - имя поля (переменной, содержащейся в клетке).

Назначение

С помощью данного свойства можно получить ссылку на объект [TemplateCell](#), который задается не координатами клетки, а по имени поля.

Если в секции имеется несколько одноименных клеток, будет возвращена ссылка на первую из них. Если такого поля нет, возвращается пустая ссылка **nil**. Свойство доступно только на чтение.

Пример

```
try
    Секция1.КлеткаПоПолю["Итого"].Font.Bold = ИСТИНА;
except
end;
```


Описание

```
КоличествоКадров : Целое;  
Количество : Целое;  
FramesCount : Integer;  
Count : Integer;
```

Назначение

Позволяет узнать количество кадров в повторяющейся секции. Кроме того, если секция не связана с подтаблицей, то данное поле доступно на запись и позволяет изменить количество кадров. Для неповторяющихся секций значение данного поля не имеет смысла.

Пример

```
-- цикл по всем клеткам секции  
  for j=1..Sec.RowsCount*Sec.Count do  
    -- ...  
  end;
```

См. также [Пример работы с секциями, столбцами, строками и клетками шаблонов](#).

Описание

КоличествоСтолбцов : Целое;
ColumnsCount : Integer;

Назначение

Позволяет узнать и задать количество столбцов в секции.

Пример

```
-- цикл по колонкам секции
for k=1..Sec.ColumnsCount do
  -- ...
end;
```

См. также [Пример работы с секциями, столбцами, строками и клетками шаблонов](#).

Описание

КоличествоСтрок : Целое;
RowsCount : Integer;

Назначение

Позволяет узнать и изменить количество строк в каждом кадре повторяющейся секции или же во всей секции, если она неповторяющаяся. Общее количество строк в повторяющейся секции может быть рассчитано как произведение **RowsCount***[FramesCount](#).

Пример

```
-- цикл по всем строкам секции
  for j=1..Sec.RowsCount*Sec.Count do
    -- ...
  end;
```

См. также [Пример работы с секциями, столбцами, строками и клетками шаблонов](#).

Описание

МожноВставлятьКадр : Логическое;
CanInsertFrame : Logical;

Назначение

Позволяет узнать и/или изменить свойство, определяющее возможность вручную (с помощью команд пользовательского интерфейса) добавлять кадры в [повторяющейся секции](#). Если поле имеет значение TRUE, кадры в секции могут добавляться (вставляться) пользователем. В противном случае кадры могут добавляться только программно. Данное поле по функционалу эквивалентно флагу **Разрешено: добавлять строки** в диалоге ["Свойства секции"](#).

Пример

```
if АвтоматическийРасчетСебестоимости = TRUE then
  СекцияИзделие.РазрешеноПовторение = FALSE;
else
  СекцияИзделие.РазрешеноПовторение = TRUE;
  СекцияИзделие.МожноВставлятьКадр = TRUE;
end;
```

Описание

МожноПеремещатьКадр : Логическое;
CanMoveFrame : Logical;

Назначение

Позволяет узнать и/или изменить свойство, определяющее возможность вручную (с помощью команд пользовательского интерфейса) перемещать кадры в [повторяющейся секции](#). Если поле имеет значение TRUE, пользователю разрешено менять порядок следования кадров в секции. В противном случае кадры могут переупорядочиваться только программно.

Данное поле по функционалу эквивалентно флагу **Разрешено: перемещать строки** в диалоге ["Свойства секции"](#).

Пример

```
СекцияИзделие.РазрешеноПовторение = TRUE;  
СекцияИзделие.МожноВставлятьсяКадр = FALSE;  
СекцияИзделие.МожноУдалятьКадр = FALSE;  
-- добавлять и удалять строки запрещено, но  
-- менять их местами можно  
СекцияИзделие.МожноПеремещатьКадр = TRUE;
```

Описание

МожноУдалятьКадр : Логическое;
CanDeleteFrame : Logical;

Назначение

Позволяет узнать и/или изменить свойство, определяющее возможность вручную (с помощью команд пользовательского интерфейса) удалять кадры из [повторяющейся секции](#). Если поле имеет значение TRUE, кадры в секции могут удаляться пользователем. В противном случае кадры могут удаляться только программно. Данное поле по функционалу эквивалентно флагу **Разрешено: удалять строки** в диалоге ["Свойства секции"](#).

Пример

```
if АвтоматическийРасчетСебестоимости = TRUE then
  СекцияИзделие.РазрешеноПовторение = FALSE;
else
  СекцияИзделие.РазрешеноПовторение = TRUE;
  СекцияИзделие.МожноВставлятьКадр = TRUE;
  СекцияИзделие.МожноУдалятьКадр = TRUE;
end;
```

Описание

Печатать : Логическое;
Printed : Logical;

Назначение

Позволяет узнать, выводится ли секция шаблона на печать, а также управлять данным аспектом, присваивая полю значения TRUE (секция будет печататься) или FALSE (секция не будет печататься). Данное свойство автоматически изменяется при изменении свойства [Видна/Visible](#).

Пример

```
СекцияПереченьТМЦ.Печатать = FALSE;
```

См. также [Пример работы с секциями, столбцами, строками и клетками шаблонов](#).

Описание

`Подтаблица` : [Подтаблица](#);
`Subtable` : [Subtable](#);

Назначение

Поле содержит ссылку на объект подтаблицы, связанной с данной секцией. Имя этой подтаблицы можно узнать с помощью свойства [ИмяПодтаблицы](#)).

Поле доступно только на чтение.

Поле имеет смысл только в том случае, если секция является [повторяющейся](#) и находится в шаблоне бланка редактора.

Пример

```
СекцияИзделие.РазрешеноПовторение = TRUE;  
СекцияИзделие.ИмяПодтаблицы = "Детали";  
СекцияИзделие.ФильтрПодтаблицы = "Код > 100 AND Код < 200";  
-- выводим отладочную информацию о количестве записей  
-- в подтаблице (с учетом наложения фильтра)  
trace(СекцияИзделие.Подтаблица.Количество);
```


Описание

РазрешеноПовторение : Логическое;
EnableRepeat : Logical;

Назначение

Позволяет программным способом узнать и/или изменить свойство [повторяемости секции](#). Если поле имеет значение True, кадры в повторяющейся секции могут повторяться и разрешается выполнять следующие действия:

- [добавлять кадры](#);
- [перемещать кадры](#);
- [удалять кадры](#).

Указанные действия запрещены, если свойство равно False.

Данное поле по функционалу эквивалентно флагу **Разрешено повторение секции** в диалоге "[Свойства секции](#)".

Пример

```
if АвтоматическийРасчетСебестоимости = TRUE then
    СекцияИзделие.РазрешеноПовторение = FALSE;
else
    СекцияИзделие.РазрешеноПовторение = TRUE;
end;
```

Описание

РодИндекс : Целое;
ParentIndex : Integer;

Назначение

Поле предназначено для чтения порядкового номера данной секции во внутреннем списке секций, поддерживаемом шаблоном. Секции нумеруются, начиная с 1.

Поле доступно только на чтение.

Пример

```
-- по нажатию кнопки
-- добавляем секцию и выводим ее индекс
proc PressMoreButton(B:Button);
var t:TemplateSection;
  t = self.template.AddSection;
  hint(t.ParentIndex);
end;
```

Описание

Столбец[Индекс:Целое] : [СтолбецШаблона](#);
Column[Index:Integer] : [TemplateColumn](#);

Аргументы

Индекс - номер столбца секции.

Назначение

Позволяет получить доступ к указанному по порядковому номеру столбцу секции. Нумерация ведется с 1 до [ColumnsCount](#) (текущей секции). Если индекс выходит за границы секции, генерируется исключение.

Пример

```
var k: integer;  
var Col: TemplateColumn;  
var Sec: TemplateSection;  
  
for k=1..Sec.ColumnsCount do  
    -- присваиваем переменной значение текущего столбца  
    -- для оптимизации быстрогодействия  
    Col=Sec.Column[k];  
    -- ...  
end;
```

См. также [Пример работы с секциями, столбцами, строками и клетками шаблонов](#).

Описание

Строка[Индекс:Целое] : СтрокаШаблона;
Row[Index:Integer] : TemplateRow;

Назначение

Позволяет получить доступ к указанной по порядковому номеру строке секции. Если индекс выходит за границы секции, генерируется исключение.

Нумерация ведется с 1 до общего числа строк [RowsCount](#) * [FramesCount](#).

Пример

```
var k: integer;  
var Row: TemplateRow;  
var Sec: TemplateSection;  
  
for k=1..Sec.RowsCount do  
  -- присваиваем переменной значение текущей строки  
  -- для оптимизации быстродействия  
  Row=Sec.Row[k];  
  -- ...  
end;
```

См. также [Пример работы с секциями, столбцами, строками и клетками шаблонов](#).

Описание

ФильтрПодтаблицы : Строка;
SubtableFilter : String;

Назначение

Данное поле используется только в том случае, когда секция является [повторяющейся](#) и находится на шаблоне бланка-редактора, а кроме того с секцией должна быть связана подтаблица (имя подтаблицы задается с помощью свойства [ИмяПодтаблицы](#)).

Поле **ФильтрПодтаблицы** позволяет установить на отображаемые записи подтаблицы произвольный фильтр - строковое выражение, отвечающее общим правилам написания [фильтров](#). В подтаблице будут отображаться только те записи, для которых выражение фильтра принимает истинное значение.

В визуальном редакторе шаблонов данное свойство вводится с помощью поля **Фильтр** в диалоге ["Свойства секции"](#).

Пример

```
СекцияИзделие.РазрешеноПовторение = TRUE;  
СекцияИзделие.ИмяПодтаблицы = "Детали";  
СекцияИзделие.ФильтрПодтаблицы = "Код > 100 AND Код < 200";
```

Описание

```
ВставитьКадр(Индекс :Целое [; перем Структ :Структура]);  
InsertFrame(Index :Integer [; var Struct :Structure]);
```

Аргументы

Индекс - определяет, в какое место секции нужно вставить кадр.

Структ - необязательный параметр, заполняется после выполнения процедуры ссылкой на добавленный кадр. Перед вызовом процедуры **Структ** не нужно чем-либо инициализировать.

Назначение

Вставляет новый кадр в повторяющуюся секцию на указанное место, отодвигая, в случае необходимости, имеющиеся кадры вниз. Значение позиции нового кадра должно лежать в пределах от 1 до количества уже имеющихся кадров плюс 1. Например, если в секции существует 3 кадра, то новый можно вставить на позиции от 1 до 4, причем индекс 4 означает добавление кадра в самый низ секции.

Пример

```
func ПриВходеОтв(Field:String;Index:Integer):Logical;  
var ТекущФизЛицо:Пример.Справочники.ФизЛицо;  
ТекущФизЛицо:=if(ПозПериодПоля.Count>0,Значения[Index],NIL);  
if ОткрытьКартотеку( "Пример.Справочники.ФизЛица",  
ТекущФизЛицо," ",Истина)=cmOk:  
  if ПозПериодПоля.Count=0:  
    ПозПериодПоля.InsertFrame(1);  
  fi;  
  Значения[Index]=ТекущФизЛицо;  
  ЗначенияФИО[Index]=Значения[Index].ФИО;  
fi;  
return Ложь;  
end;
```

Описание

```
КадрВверх(Индекс:Целое);  
FrameUp(Index :Integer);
```

Аргументы

Индекс - порядковый номер кадра, который требуется переместить на одну позицию вверх.

Назначение

Перемещает указанный кадр на одну позицию вверх, меня его местами с предыдущим кадром.

Номер перемещаемого кадра должен лежать в пределах от 2 до числа кадров в секции. Первый кадр нельзя переместить вверх.

Пример

```
proc FrameUpSafe(S: Section; N: Integer);  
  if N > 1 AND N <= S.Count  
    S.FrameUp(N);  
  end;  
end;
```

Описание

```
КадрВниз(Индекс:Целое);  
FrameDown(Index :Integer);
```

Аргументы

Индекс - порядковый номер кадра, который требуется переместить на одну позицию вниз.

Назначение

Перемещает указанный кадр на одну позицию вниз, меня его местами с последующим кадром.

Номер перемещаемого кадра должен лежать в пределах от 1 до числа кадров в секции минус 1. Последний кадр нельзя переместить вниз.

Пример

```
proc FrameDownSafe(S: Section; N: Integer);  
  if N >= 1 AND N < S.Count  
    S.FrameDown(N);  
  end;  
end;
```


Описание

```
ОбъединитьКлетки(Колонка: Целое; Строка: Целое; [Колонок: Целое]; [Строк: Целое]);  
LinkCells(Col: Integer; Row: Integer; [Cols: Integer]; [Rows: Integer]);
```

Аргументы

Колонка и Строка - соответственно порядковые номера по горизонтали и вертикали левой верхней клетки из тех, что будут объединяться (разъединяться); индексы подсчитываются начиная с 1.

Колонок и Строк - задают соответственно ширину и высоту объединяемого фрагмента в клетках.

Назначение

Процедура объединяет или разъединяет клетки секции. Если секция повторяющаяся, то диапазон строк [Row...Row+Rows-1] должен целиком лежать в пределах первой итерации. Для разъединения клеток два последних параметра должны быть равны 1 или опущены.

Пример

```
Section1.LinkCells(3,1,1,2);  
-- "склеивает" две строки в третьей колонке
```

Описание

```
УдалитьКадр(Индекс:Целое);  
DeleteFrame(Index :Integer);
```

Аргументы

Индекс - порядковый номер кадра, который требуется удалить. Номер удаляемого кадра должен лежать в пределах от 1 до числа кадров в секции.

Назначение

Удаляет указанный кадр из секции.

Пример

```
var i,j,n: Integer;  
n = Section.Count;  
for i=1..n do j = n +1 - i; -- обратный цикл по кадрам  
  -- если кадр содержит неверную информацию...  
  if Section.Cell[j,3] = 0 then  
    Section.DeleteFrame(j); -- удаляем его  
  end;  
end;
```

Описание

```
УпорядочитьПо(Поле :Строка);  
SortBy(Field :String);
```

Аргументы

Поле - строка, содержащая записанный специальным образом, порядок сортировки повторяющейся секции. Строка должна содержать одно или несколько разделенных запятыми имен клеток секции. Под именем клетки здесь понимается либо идентификатор поля подтаблицы, вписанный в эту клетку, либо указанный в ней идентификатор массива. Напомним, что такие идентификаторы записываются в клетке после знака '#'. В простейшем случае это имя одной клетки (например, "Сумма").

Назначение

Процедура сортирует кадры повторяющейся секции по указанной клетке (или совокупности клеток одного кадра) этой секции. Если кадр секции содержит лишь одну строку, то сортировка по одной клетке задает сортировку по столбцу. Если же кадр содержит более одной строки, то, вообще говоря, один и тот же столбец может быть отсортирован несколькими способами – по любой из клеток, входящих в этот столбец в пределах кадра.

Рассмотрим пример. Пусть в дизайн-режиме спроектирована секция с двумя строками в кадре:

#Название	#Количество	#Сумма
#Цена	<пусто>	<пусто>

Если на стадии выполнения бланка секция имеет 3 кадра и отсортирована по полю "Название", то она выглядит следующим образом:

Аврора	1	50
50	<пусто>	<пусто>
Цефей	4	280
70	<пусто>	<пусто>
Ястреб	2	80
40	<пусто>	<пусто>

После сортировки по полю "Цена" (также в 1-ом столбце) секция будет выглядеть уже иначе:

Ястреб	2	80
40	<пусто>	<пусто>
Аврора	1	50
50	<пусто>	<пусто>
Цефей	4	280
70	<пусто>	<пусто>

Упорядочивание возможно по разыменованным и вычислимым полям. Например, если в клетке указано разыменование "#Товар.Название" (переменная Товар или поле записи Товар является ссылкой на объект некоторого класса, в котором определено поле Название), то для сортировки секции по названиям товаров достаточно вызвать SortBy с параметром "Товар.Название". В случае вычислимого поля для обеспечения сортировки необходимо в соответствующей клетке ввести строку вида "#Имя", где имя – произвольный идентификатор, отличный от имен других клеток секции. Поскольку содержание вычисляемых полей формируется "на лету", в них обычно не нужно (и не имеет смысла) писать строку с указанием какого-либо имени. Однако, если необходимо производить сортировку по такому полю, для его идентификации требуется указать уникальное имя. Это имя используется только для сортировки и никак не влияет на логику работы вычислимого поля.

Кроме того поддерживается упорядочивание сразу по комбинации полей, для чего их имена перечисляются в требуемом порядке приоритетов сортировки, разделенные запятыми. Например, если нужно отсортировать секцию вначале по полю Название, а потом (если встретится несколько полей с одним и тем же названием) по полю Производитель, то в процедуру необходимо передать параметр "Название,Производитель".

Для любого поля из тех, что упомянуты в строке Порядок, можно дополнительно в явном виде указать порядок сортировки: по возрастанию или по убыванию. Если сразу после имени поставлен знак '+', то

секция сортируется по возрастанию значений в этом поле, если же после имени поставлен знак '-', то сортировка выполняется по убыванию. По умолчанию, если ни тот, ни другой знак не поставлен, выполняется сортировка по возрастанию.

Примеры:

-- сортируем секцию СекТовары по цене в убывающем порядке

СекТовары.SortBy("Цена-");

-- сортируем по названиям в возрастающем порядке и по количеству в убывающем

СекТовары.SortBy("Товар.Название+,Количество-");

Пример

```
proc SortByField(S: Section; F: String);
  if S.Count > 1 then
    S.SortBy(F);
  end;
end;
```

Событие ПослеВставки / AfterInsert

Описание

ПослеВставки : Строка;
AfterInsert : String;

Назначение

Данное событие происходит после вставки нового кадра в периодическую секцию. Поле **ПослеВставки** позволяет определить или заменить название прикладной процедуры-обработчика события.

Обработчик

proc **AfterInsert** (S :Section; Index :Integer);
S - секция, в которой произошла вставка;
Index - номер вставленного кадра (нумерация ведется с 1).

Пример

```
proc Секция1_ПослеВставки(S:Section; Index:Integer);  
  S.Cell[1, Index].Value = Str(In);  
end;
```

Описание

ПослеУдаления : Строка;
AfterDelete: String;

Назначение

Данное событие происходит после удаления кадра из периодической секции. Поле **ПослеУдаления** позволяет определить или заменить название прикладной процедуры-обработчика события. Удаление кадра может быть запрещено с помощью обработчика события [ПриУдалении](#). В этом случае событие **ПослеУдаления** не генерируется.

Обработчик

proc **AfterDelete**(S :Section);
S - секция, в которой произошло удаление.

Пример

```
var BlankVariable: numeric;  
  
proc Секция1_ПослеУдаления(S1 :Section);  
var sum: numeric;  
var i:integer;  
sum = 0.0;  
for i=1..S1.Count do  
    sum = sum+S1.Cell[1, i].Value;  
end;  
BlankVariable = sum;  
end;
```

Описание

ПриВставке : Строка;
OnInsert: String;

Назначение

Данное событие происходит перед вставкой нового кадра в периодическую секцию. Поле **ПриВставке** позволяет определить или заменить название прикладной процедуры-обработчика события.

Обработчик

func **OnInsert** (S :Section; Index :Integer) :Logical;
S - секция, в которой происходит вставка;
Index -номер вставленного кадра (нумерация ведется с 1).

Если обработчик возвращает TRUE, система производит стандартные действия по вставке кадра. Если обработчик возвращает FALSE, система не делает обычной обработки (кадр не вставляет автоматически), однако программист может дополнить обработчик исходным кодом, выполняющем вставку кадра с учетом специфических условий конкретной задачи. В последнем случае некоторые действия могут быть выполнены как до, так и после вставки кадра.

Если обработчик вернул TRUE, то сразу после автоматической вставки кадра генерируется событие [ПослеВставки](#).

Пример

```
func Секция1_ПриВставке(S :TemplateSection; Index :Integer) :Logical;  
  var NewStruct :Документы.Накладная.Товары;  
  
  -- Действия до вставки  
  -- ...  
  
  -- вставляем кадр вручную,  
  -- ссылка на новую структуру будет записана  
  -- в переменную NewStruct  
  S.InsertFrame(Index, NewStruct);  
  
  NewStruct.Количество = 1;  
  -- Задание значений по умолчанию...  
  -- Если подтаблица с фильтром то здесь вы должны гарантировать,  
  -- что новая структура удовлетворяет фильтру!  
  
  -- Действия после вставки  
  -- ...  
  
  Template.Frame = Index;  
  
  return False; -- Запрещаем стандартную обработку  
  
end;
```

Описание

ПриОформлении: Строка;
OnRearrange: String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при изменении пользователем какого-либо атрибута строки или столбца секции. Системой отслеживаются события, указанные в типе **ДействияОформления**, связанные с изменением ширины столбца, высоты строки или их видимости.

Обработчик

proc **OnRearrange**(Section :TemplateSection; Action : [Template.RearrangeActions](#) SectionObject :Object);

Параметры:

Section - указатель на объект класса **СекцияШаблона**;
Action - константа, описанная в перечислимом типе [Шаблон.ДействияОформления](#) и характеризующая тип события, связанного с изменением какого-либо атрибута объекта **SectionObject**;
SectionObject - указатель на объект, в котором произошло событие.

Пример

```
proc СекцияЗаголовок_ПриОформлении(Section:TemplateSection;  
  Action :Template.RearrangeActions; SectionObject :Object);  
  var локНомерСтолбца :Integer;  
  var локШирина       :Numeric;  
  var локНомерСекции  :String;  
  
  if Action = Template.ResizeColumn then  
    if SectionObject<>nil then  
      локНомерСтолбца = (SectionObject as TemplateColumn).Number;  
      локНомерСекции  = SubStr(Section.Name, Length(Section.Name), 1);  
      локШирина       = Section.Column[локНомерСтолбца].Width;  
      УстановитьШиринуКолонки('СекцияПозиции' + локНомерСекции,  
        локНомерСтолбца, локШирина);  
    else  
      for локНомерСтолбца=1..Section.ColumnsCount do  
        локНомерСекции  = SubStr(Section.Name, Length(Section.Name), 1);  
        локШирина       = Section.Column[локНомерСтолбца].Width;  
        УстановитьШиринуКолонки('СекцияПозиции' + локНомерСекции,  
          локНомерСтолбца, локШирина);  
      end;  
    end;  
  end;  
end;  
end;
```



```

proc Card_OnRearrange(Sender :TemplateSection;
Action :Cardfile.RearrangeActions; Column :CardfileColumn);
-- в случае сортировки по какому-либо столбцу
if Action = Cardfile.Sort then
-- вызываем прикладную функцию для сортировки
-- служебных массивов
SortAuxArrays(Column);
end;
end;

```

```

proc Card_OnRearrange(Sender :TemplateCardfile; Action :Cardfile.RearrangeActions; Column :CardfileColumn); -- в
случае сортировки по какому-либо столбцу if Action = Cardfile.Sort then -- вызываем прикладную функцию для
сортировки -- служебных массивов SortAuxArrays(Column); end; end; proc Card_OnRearrange
(Action :Cardfile.RearrangeActions; Column :CardfileColumn); -- в случае сортировки по какому-либо столбцу if
Action = Cardfile.Sort then -- вызываем прикладную функцию для сортировки -- служебных массивов
SortAuxArrays(Column); end; end;

```

Описание

ПриСменеПозиции : Строка;
OnChangePosition: String;

Назначение

Данное событие позволяет управлять процессом перемещения кадров в секции.

Обработчик

func **OnChangePosition**(Section :TemplateSection; Index :Integer; MoveUp :Logical) :Logical;

Section - секция, в которой происходит перемещение кадров;

Index - номер перемещаемого кадра. Нумерация начинается с 1;

MoveUp - равно TRUE, если кадр перемещается вверх, и FALSE - в противном случае.

Возвращаемый функцией результат разрешает (TRUE) либо запрещает (FALSE) перемещение. В случае, если обработчик вернул FALSE, перемещение можно осуществить, запрограммировав его вручную (с учетом специфики выполняемой задачи).

Пример

```
func Секция1_ПриСменеПозиции(S1 :Section; N :Integer; Up :Logical): Logical;  
  if S1 = СекцияТоварныхПозиций then  
    return TRUE; -- разрешаем перемещение кадра  
  else  
    return FALSE; -- запрещаем перемещение кадра  
  end;  
end;
```

Описание

ПриУдалении : Строка;
OnDelete: String;

Назначение

Данное событие происходит до попытки удаления кадра из повторяющейся секции, причем событие не возникает, если кадр удаляется программно. Поле **ПриУдалении** позволяет определить или заменить название прикладной функции-обработчика события. Если функция возвращает значение False, то удаления кадра не происходит и событие [ПослеУдаления](#) не генерируется.

Обработчик

proc **OnDelete**(S :Section; Index :Integer): Logical;
S - секция, в которой произошло удаление;
Index - номер удаляемого кадра; нумерация начинается с 1.

Возвращаемый функцией результат разрешает либо запрещает удаление.

Пример

```
func Секция1_ПриУдалении(S1 :Section; N: Integer): Logical;  
  if N > 1 then -- если кадр не первый...  
    return TRUE; -- ...разрешаем его удалить  
  else  
    return FALSE; -- первый кадр запрещаем удалять  
  end;  
end;
```

Описание

```
СуммаПоля( (Поле:Строка);  
SumOfField(Field:String);
```

Аргументы

Поле - имя поля подтаблицы, связанной с повторяющейся секцией, или имя массива, связанного с вычисляемым столбцом повторяющейся секции.

Назначение

Функция суммирует значения в указанном столбце секции и возвращает полученный результат.

При подсчете суммы с помощью данной функции по *вычисляемому полю* вызывается обработчик [OnOutput](#) с параметром Action = Template.Calculation (по разу для каждой строки секции!)

Внимание. Использование данной возможности может привести к большим накладным расходам. Крайне не рекомендуется вызывать этот метод из обработчика **OnOutput** (например, при расчете итогов по накладной).

Пример

```
БланкСШаблоном "Счет", Редактор Документы.Счета;  
Позиции : Section;  
СумДок : Numeric=Позиции.СуммаПоля( "СУММА" );
```

Класс *СтильШаблона* (*TemplateStyle*), производный от родительского класса [Объект](#), предназначен для программного управления стилями шаблонов, то есть нюансами их визуального оформления, такими как используемый шрифт, цвет, рамки и т.д.

В классе *СтильШаблона* вводятся следующие свойства:

- [Поле Имя / Name](#)
- [Поле Владелец / Owner](#)
- [Поле РодИндекс / ParentIndex](#)
- [Поле Выравнивание / Alignment](#)
- [Поле ВертВыравнивание / VertAlignment](#)
- [Поле Свертка / Wrap](#)
- [Поле Обязательное / Required](#)
- [Поле ПечататьБордюры / PrintBevel](#)
- [Поле Многострочность / MultiLine](#)
- [Поле Макросы / UseMacro](#)
- [Поле Бордюры / Bevel](#)
- [Поле Цвет / Color](#)
- [Поле ЦветПоля / FieldColor](#)
- [Поле Шрифт / Font](#)
- [Поле Автоблокировка / AutoLock](#)
- [Поле АвтоВыделение / AutoSelect](#)
- [Поле МожноСфокусировать / CanFocus](#)
- [Поле ПоворотТекста / TextRotation](#)
- [Поле ТабСтоп / TabStop](#)

Внимание. Создать объект класса *СтильШаблона* можно с помощью функции [ДобавитьСтиль](#) класса *Шаблон*.

Кроме того, стили шаблонов создаются и самой системой на основе установок, сделанных разработчиком в визуальном редакторе бланков. Объект класса *СтильШаблона* используется в качестве свойств шаблона и клеток шаблона.

Стиль шаблона определяет поименованный набор свойств клетки шаблона, связанных с её внешним представлением, например, шрифт, цвет фона, использование макросов и т.д. Применение стиля к клетке (в дизайн-режиме или программным способом с помощью свойства *Стиль* клетки шаблона или всего шаблона) позволяет сразу изменить все её свойства. Кроме того, после применения стиля сохраняется связь между клеткой (шаблоном) и стилем, т.е. изменение какого-либо свойства в стиле автоматически приводит к изменению одноименного свойства в клетке (шаблоне).

Если впоследствии для какого-либо свойства клетки (шаблона) будет выбрано иное значение (отличное от того, которое имеется в примененном стиле), связь между клеткой и стилем по данному свойству разрывается – т.е. изменение этого свойства в стиле более не будет приводить к автоматическому изменению соответствующего свойства в клетке. Для восстановления связи можно либо заново применить стиль к клетке (шаблону), либо изменить значение требуемого свойства таким образом, чтобы оно совпадало со значением в стиле.

Описание

Автоблокировка :Логическое;
AutoLock :Logical;

Назначение

Свойство **Автоблокировка** логического типа имеет смысл только для полей бланка-редактора.

По умолчанию Autolock = True и документ при входе в поле автоматически переводится в режим редактирования. Если Autolock = False, то документ будет переведен в режим редактирования при изменении значения при завершении редактирования поля.

Если поле вычисляемое и Autolock = False, то в случае необходимости перевод в режим редактирования осуществляется только программно.

Пример

```
Шаблон.Стиль[2].AutoLock = Истина;
```

Описание

АвтоВыделение :Логическое;
AutoSelect :Logical;

Назначение

Данное свойство позволяет выделять текст (свойство равно Истина) при входе в клетку (при открытии встроенного in-place редактора) шаблона данного стиля.

Если свойство равно Ложь, то текст при входе в клетку данного стиля выделен не будет, что позволяет избежать случайного удаления выделенного текста.

Пример

Шаблон.Стиль[2].**АвтоВыделение** = Истина;

Описание

Бордюр : [Шаблон.СтилиБордюра](#) ;
Bevel : [Template.BevelStyles](#) ;

Назначение

Свойство позволяет определить или установить вокруг элемента один из доступных стилей бордюра, который определен в перечислимом типе [СтилиБордюра/BevelStyles](#).

При выборе значения *None* бордюр вокруг элемента отсутствует, в остальных случаях бордюр прорисовывается одним из следующих стилей:

- по умолчанию | DefaultBevel
- тонкий | SingleBevel
- статический | StaticBevel
- внутренний | ClientBevel
- оконный | WindowBevel

Поле доступно на чтение и запись.

Пример

```
Шаблон.Стиль[1].Бордюр = Шаблон.SingleBevel;
```


Описание

ВертВыравнивание: Шаблон.[ТипыВыравнивания](#);

VertAlignment: Template.[AlignmentTypes](#);

Назначение

Позволяет узнать и изменить режим выравнивания текста по вертикали при использовании данного стиля в клетке.

Поле может принимать одно из значений, определенных в типе [Шаблон.ТипыВыравнивания](#).

Пример

```
Шаблон.ТекущаяКлетка.Стиль.VertAlignment = Ядро.Шаблон.ВыравниватьВниз;
```

Описание

Владелец: Шаблон;
Owner: Template;

Назначение

Поле позволяет получить ссылку на шаблон, в котором определен данный стиль.

Поле доступно только на чтение.

Пример

```
-- в классе храним текущий стиль
var aCurrentStyle :TemplateStyle;
-- ...

-- когда необходимость в стиле отпала,
-- удаляем его, обращаясь к методу владельца
proc ClearStyle;
    aCurrentStyle.Owner.DeleteStyle(aCurrentStyle.ParentIndex);
end;
```

Описание

Выравнивание: Шаблон.[ТипыВыравнивания](#);

Alignment: Template.[AlignmentTypes](#);

Назначение

Позволяет узнать и изменить режим выравнивания текста в элементе, оформленном данным стилем.

Поле может принимать одно из значений, определенных в типе [Шаблон.ТипыВыравнивания](#).

Пример

Шаблон.ТекущаяКлетка.Стиль.**Выравнивание** = Ядро.Шаблон.ВыравниватьПоЦентру;

Поле Имя / Name

Описание

Имя : Строка;

Name: String;

Назначение

Поле предназначено для чтения и установки имени стиля.

Пример

```
hint(Шаблон.ТекущаяКлетка.Стиль.Имя);
```

Описание

Макросы: Логическое;

UseMacro: Logical;

Назначение

Поле определяет, разрешено ли в клетках, оформленных данным стилем, использовать [макросы](#).

Если значение свойства равно TRUE, то любая последовательность символов, отвечающая синтаксису записи макросов, будет интерпретироваться системой как макрос (т.е. изменять способ отображения сопутствующего текста).

Если значение свойства равно FALSE, весь текст выводится непосредственно в том виде, в каком он представлен в клетке.

Описание

Многострочность :Логическое;
MultiLine :Logical;

Назначение

Данное свойство позволяет представить текст в клетке в виде нескольких строк. Если свойство равно Истина, доступен перевод каретки и разрешается переносить текст на следующую строку, иначе - перевод текста на другую строчку невозможен.

Пример

```
Шаблон.Стиль[3].Многострочность = Истина;
```

Описание

МожноСфокусировать :Логическое;
CanFocus :Logical;

Назначение

В зависимости от значения этого поля на клетки данного стиля можно установить фокус или запретить его. В каждый момент времени только один объект в системе имеет фокус.

Если свойство равно Истина, то поля могут стать активными, и их можно редактировать, то есть в них разрешен ввод с клавиатуры.

Если свойство принимает значение "Ложь", то поля данного стиля не могут быть активизированы, т.е. "получить фокус", и их нельзя редактировать.

Пример

```
Шаблон.Стиль[2].МожноСфокусировать = Истина;
```

Описание

Обязательное :Логическое;

Required :Logical;

Назначение

Свойство предназначено для полей, обязательных для ввода данных и оформленных данным стилем. Если значение поля равно True и поле не заполнено, то клетка выделяется специальным цветом.

Поле доступно на чтение и запись.

Описание

```
ПечататьБордю :Логическое;  
PrintBevel :Logical;
```

Назначение

Свойство определяет, требуется ли или нет выводить на печать бордю (обрамляющую полосу по периметру рабочей области клетки) для полей, оформленных данным стилем. Если свойство равно False, бордю на печать не выводится.

Поле доступно на чтение и запись.

Пример

```
-- бордю печатается  
Template.Style[1].PrintBevel = true;
```

Описание

ПоворотТекста :Числовой;
TextRotation :Numeric;

Назначение

Данное свойство позволяет поворачивать текст внутри клетки на фиксированные углы. Если задано значение 0 (градусов), то текст поворачиваться не будет. Поворот текста учитывает способы выравнивания текста по горизонтали и вертикали. Если текст повернут на 90 или - 90, то такие клетки не участвуют в определении авто высоты строки секции.

Примечание. Допускаются следующие значения углов поворота текста -180.0, -90.0, 0.0, 90.0, 180.0. Если провести условную горизонталь через середину клетки шаблона и указать положительное направление вправо, то угол поворота против часовой стрелки считается положительным.

Пример

```
Шаблон.Стиль[1].ПоворотТекста = 90.0;
```

Описание

РодИндекс: Целое;
ParentIndex: Integer;

Назначение

Поле предназначено для чтения порядкового номера данного стиля во внутреннем списке стилей, поддерживаемом шаблоном. Стили нумеруются, начиная с 1.

Поле доступно только на чтение.

Пример

```
var i,j:integer;
-- ...
if Секция1.Клетка[i,j].Стиль.РодИндекс > 1 then
    -- изменяем любой стиль, кроме основного с номером 1
    Секция.Клетка[i,j].Стиль.Бордюр = true;
end;
```

Описание

Свертка: Логическое;
Wrap: Logical;

Назначение

Позволяет узнать и изменить режим отрисовки длинного текста в пределах элемента, оформленного данным стилем.

Если **Свертка** равна TRUE, то символы, не уместившиеся на одной строке, будут перенесены на следующую строку и так далее. (При этом высота элемента, например клетки шаблона, автоматически подстраивается с тем, чтобы уместить весь текст.)

Если **Свертка** равна FALSE, текст выводится в одну строку и обрезается правой границей элемента.

Пример

```
Шаблон.Стиль[1].Свертка = ИСТИНА;
```

Описание

ТабСтоп :Логическое;
TabStop :Logical;

Назначение

По умолчанию данное свойство равно False.

Если свойство ТабСтоп = True, то для клеток текущего стиля разрешается переход на клетку по клавише *Tab*.

Если данное свойство равно False, то нельзя перейти на эти клетки клавишей *Tab*.

Предупреждение. При считывании старых шаблонов все "поля" у которых не стоит флаг "только для чтения" и все специальные клетки (кнопки, заголовки или гиперссылки) помечаются свойством TabStop = True. В дизайн-режиме при вводе или удалении в клетке первого символа [#] "TabStop" инвертируется.

Пример

```
Шаблон.Стиль[1].ТабСтоп = ИСТИНА;
```

Описание

Цвет: Целое;
Color: Integer;

Назначение

Позволяет узнать и изменить цвет фона элемента, оформленного данным стилем.

Пример

```
-- подаем на вход функции три составляющих цвета
-- (красную, зеленую, синюю), интенсивность каждой
-- задается числом от 0 до 255
-- на выходе получаем код цвета
func Colour(Red:Integer; Green:Integer; Blue:Integer): Integer;
var R, G, B: Integer;
var C:Integer;
    -- составляющие цвета должны лежать в пределах 0-255
    R = Mod(Red,256);
    G = Mod(Green,256);
    B = Mod(Blue,256);
    C = R + G*256 + B*256*256;
    F.Color = C;
    return C;
end;

proc Button1Press(B:Button);
    Шаблон.ТекущаяКлетка.Стиль.Цвет = Colour(128,0,255);
end;
```

Описание

ЦветПоля: Целое;
FieldColor: Integer;

Назначение

Позволяет установить или изменить цвет поля клетки, оформленной данным стилем.

Пример

```
-- подаем на вход функции три составляющих цвета
-- (красную, зеленую, синюю), интенсивность каждой
-- задается числом от 0 до 255
-- на выходе получаем код цвета
func Colour(Red:Integer; Green:Integer; Blue:Integer): Integer;
var R, G, B: Integer;
var C:Integer;
    -- составляющие цвета должны лежать в пределах 0-255
    R = Mod(Red,256);
    G = Mod(Green,256);
    B = Mod(Blue,256);
    C = R + G*256 + B*256*256;
    F.Color = C;
    return C;
end;

proc Button1Press(B:Button);
    Шаблон.ТекущаяКлетка.Стиль.ЦветПоля = Colour(128,0,255);
end;
```

Описание

Шрифт: Шрифт;

Font: Font;

Назначение

Позволяет узнать параметры шрифта элемента, оформленного данным стилем. Поле доступно только на чтение.

Пример

```
proc Button1Press(B:Button);  
  Шаблон.ТекущаяКлетка.Стиль.Шрифт.Жирный = True;  
end;
```


Класс *СтолбецШаблона* (*TemplateColumn*), производный от родительского класса [Объект](#), используется для работы со столбцами шаблонов Студии. Класс *СтолбецШаблона* наследует от родительского класса свойства, описанные [в разделе Объект](#).

Непосредственно в классе *СтолбецШаблона* определены следующие свойства:

- [Поле Имя / Name](#)
- [Поле Надпись / Caption](#)
- [Поле Владелец / Owner](#)
- [Поле Номер / Number](#)
- [Поле Ширина / Width](#)
- [Поле МинШирина / MinWidth](#)
- [Поле МаксШирина / MaxWidth](#)
- [Поле Слева / Left](#)
- [Поле Виден / Visible](#)
- [Поле Печатать / Printed](#)
- [Поле МожноМенятьШирину / CanWidthModifyh](#)
- [Поле МожноСкрывать / CanVisibleModify](#)

Внимание! Создать объект класса *СтолбецШаблона* напрямую нельзя. Столбцы шаблонов создаются только внутри системы и используются как свойства других объектов (класса [СекцияШаблона](#)), причем такие свойства имеют атрибут "только чтение". Вместе с тем, программа на ТБ.Скрипт может получить доступ к объекту *СтолбецШаблона*, являющемуся свойством родительского объекта, и изменять свойства столбца.

Описание

Виден : Логическое;
Visible : Logical;

Назначение

Позволяет узнать, виден ли столбец шаблона, а также управлять его видимостью, присваивая значения TRUE (столбец виден) и FALSE (столбец скрыт). Когда устанавливается значение данного свойства, синхронно изменяется и свойство [Печатать](#).

Пример

```
-- глобально описанные интерфейсные объекты шаблона
СекцЦены : Section;
Bevel2 : Object;

-- ... внутри обработчика некоторого события
-- меняем ширину внешней рамки секции по ее размеру
var w: Numeric;
var n: Integer;
w:=0;
for n=1..СекцЦены.ColumnsCount do
  -- устанавливаем видимость столбцов
  СекцЦены.Column[n].Visible = ОбщиеНастройкиЦен.Показать[n];
  СекцЦены.Column[n].Printed = ОбщиеНастройкиЦен.Печать[n];
  -- обрабатываем только видимые столбцы
  if СекцЦены.Column[n].Visible then
    w=w+СекцЦены.Column[n].Ширина;
  end;
end;
-- подгоняем размер рамки
Bevel2.Ширина=w+5;
```

Описание

Владелец :СекцияШаблона;
Owner :TemplateSection;

Назначение

Содержит ссылку на секцию, которой принадлежит столбец. Поле доступно только на чтение.

Пример

```
-- изменяем параметры секции указанного столбца
proc CountColumns(RR: TemplateColumn);
var SS: TemplateSection;
  SS = RR.Владелец;
  -- если число столбцов не достигло лимита
  -- и задан последний столбец, тогда добавляем еще один
  if SS.ColumnsCount < 10 and RR.Number = SS.ColumnsCount then
    SS.ColumnsCount = SS.ColumnsCount + 1;
  end;
end;
```

Описание

Имя : Строка;
Name : String;

Назначение

Поле позволяет узнать и изменить имя столбца шаблона. Имя текущего столбца указывается в поле **Имя** диалога ["Свойства столбца"](#) .

Пример

```
-- процедура выделяем жирным шрифтом клетки столбца "Сумма"
proc ColumnCheck(TC: TemplateColumn);
var SS: TemplateSection;
var i, n, m: Integer;
  if TC.Name = "Сумма" then
    SS = TC.Section;
    n = TC.Number; -- индекс столбца
    m = SS.RowsCount; -- число строк в секции
    for i=1..m do -- цикл по клеткам столбца
      -- выделяем жирным шрифтом
      SS.Cell[n,i].Font.Bold = true;
    end;
  end;
end;
```

Описание

МаксШирина :Число;
MaxWidth :Numeric;

Назначение

Данное свойство устанавливает или считывает максимально разрешенную ширину столбца в миллиметрах. Совместно со свойством [МинШирина](#) оно определяет пределы изменения ширины текущего столбца секции в интерактивном режиме. Причем, менять ширину столбца можно только, если свойство [МожноМенятьШирину](#) = True.

Пример

```
-- глобально описанные интерфейсные объекты шаблона
СекцЦены : Section;

-- ... внутри обработчика некоторого события
var n: Integer;
for n=1..СекцЦены.ColumnsCount do
    -- устанавливаем пределы изменения столбцов
    СекцЦены.Column[n].МожноМенятьШирину=True;
    СекцЦены.Column[n].МинШирина = 10;
    СекцЦены.Column[n].МаксШирина = 1000;
end;
```

Описание

МинШирина :Число;
MinWidth :Numeric;

Назначение

Данное свойство совместно со свойством [МаксШирина](#) определяет пределы изменения ширины текущего столбца секции/ Причем, менять ширину столбца можно только, если свойство [МожноМенятьШирину](#) = True.

Пример

```
-- глобально описанные интерфейсные объекты шаблона
СекцЦены : Section;

-- ... внутри обработчика некоторого события
var n: Integer;
for n=1..СекцЦены.ColumnsCount do
    -- устанавливаем пределы изменения столбцов
    СекцЦены.Column[n].МожноМенятьШирину=True;
    СекцЦены.Column[n].МинШирина = 10;
    СекцЦены.Column[n].МаксШирина = 1000;
end;
```

Описание

МожноМенятьШирину :Логическое;
CanWidthModify :Logical;

Назначение

Данное свойство при условии, что оно равно Истина, позволяет пользователю изменять ширину текущего столбца. Причем, менять ширину конкретного столбца можно в пределах, заданных свойствами [МинШирина](#) и [МаксШирина](#).

Пример

```
-- глобально описанные интерфейсные объекты шаблона
СекцЦены : Section;

-- ... внутри обработчика некоторого события
var n: Integer;
for n=1..СекцЦены.ColumnsCount do
    -- устанавливаем пределы изменения столбцов
    СекцЦены.Column[n].МожноМенятьШирину= True;
    СекцЦены.Column[n].МинШирина = 10;
    СекцЦены.Column[n].МаксШирина = 1000;
end;
```

Описание

МожноСкрывать :Логическое;
CanVisibleModify :Logical;

Назначение

Позволяет скрывать и отображать столбцы текущей секции, для которых свойство [МожноСкрывать](#) = True.

Причем, если в текущей секции имеются такие столбцы, то доступна команда [Настроить видимость](#) контекстного меню, которая открывает диалог "Настройка видимости".

Пример

```
-- глобально описанные интерфейсные объекты шаблона
СекцЦены : Section;

-- ... внутри обработчика некоторого события
var n: Integer;
for n=1..СекцЦены.ColumnsCount do
  if n=1:
    СекцЦены.Column[1].МожноСкрывать = True;
  fi;
end;
```


Поле Надпись / Caption

Описание

Надпись :Строка;

Caption :String;

Назначение

Поле позволяет узнать и изменить заголовок столбца шаблона, которая указывается в поле **Надпись** диалога "[Свойства столбца](#)". Кроме этого, данное свойство используется в диалоге [Настроить видимость](#).

Если значение поля не пустое, то оно используется для задания имен столбцов, которые перечислены в указанном диалоге, иначе в качестве имени берется значение из свойства [Имя/Name](#). Если и оно пустое, то указывается порядковый номер столбца.

Пример

```
-- глобально описанные интерфейсные объекты шаблона
СекцЦены : Section;

-- ... внутри обработчика некоторого события
СекцЦены.Column[1].Надпись="Товары";
СекцЦены.Column[5].Caption="Учетная цена";
```

Описание

Номер : Целое;
Number: Integer;

Назначение

Поле содержит порядковый номер столбца в секции. Номера имеют значения от 1 до общего числа столбцов секции. Поле доступно только на чтение.

Пример

```
-- изменяем параметры секции указанного столбца
proc CountColumns(RR: TemplateColumn);
var SS: TemplateSection;
  SS = RR.Section;
  -- если число столбов не достигло лимита
  -- и задан последний столбец, тогда добавляем еще один
  if SS.ColumnsCount < 10 and RR.Number = SS.ColumnsCount then
    SS.ColumnsCount = SS.ColumnsCount + 1;
  end;
end;
```

Описание

Печатать : Логическое;
Printed : Logical;

Назначение

Позволяет узнать, будет ли выводиться столбец шаблона на печать. Если свойство **Печатать**=TRUE, столбец печатается, иначе (FALSE) - столбец не печатается. Данное свойство автоматически устанавливается, когда меняется свойство [Виден](#).

Пример

```
-- глобально описанные интерфейсные объекты шаблона
СекцЦены : Section;
Bevel2 : Object;

-- ... внутри обработчика некоторого события
-- меняем ширину внешней рамки секции по ее размеру
var w: Numeric;
var n: Integer;
w=0;
for n=1..СекцЦены.ColumnsCount do
  -- устанавливаем видимость столбцов
  СекцЦены.Column[n].Visible = ОбщиеНастройкиЦен.Показать[n];
  -- устанавливаем признак того, что столбец должен выводиться на печать
  СекцЦены.Column[n].Printed = ОбщиеНастройкиЦен.Печать[n];
  if СекцЦены.Column[n].Visible then
    w=w+СекцЦены.Column[n].Ширина;
  end;
end;
-- подгоняем размер рамки
Bevel2.Ширина=w+5;
```

Описание

Слева :Число;
Left :Numeric;

Назначение

Свойство возвращает расстояние в миллиметрах от левого края шаблона без учета отступов шаблона от края окна до левой границы столбца. Свойство может динамически менять свое значение, если слева от текущего столбца появятся столбцы, изменившие свою [ширину](#) или [видимость](#).

Пример

```
-- глобально описанные интерфейсные объекты шаблона
СекцЦены : Section;
var s: Numeric;

-- ...
s=СекцЦены.Column[3].Left;
```

Описание

Ширина : Число;
Width : Numeric;

Назначение

Позволяет узнать ширину столбца шаблона и изменить ее. Ширина измеряется в миллиметрах.

Пример

```
-- глобально описанные интерфейсные объекты шаблона
СекцЦены : Section;
Bevel2 : Object;

-- ... внутри обработчика некоторого события
-- меняем ширину внешней рамки секции по ее размеру
var w: Numeric;
var n: Integer;
w=0;
for n=1..СекцЦены.ColumnsCount do
  -- устанавливаем видимость столбцов
  СекцЦены.Column[n].Visible = ОбщиеНастройкиЦен.Показать[n];
  СекцЦены.Column[n].Printed = ОбщиеНастройкиЦен.Печать[n];
  if СекцЦены.Column[n].Visible then
    w=w+СекцЦены.Column[n].Ширина; -- суммируем ширину столбцов
  end;
end;
-- подгоняем размер рамки
Bevel2.Ширина=w+5;
```

Класс *СтрокаШаблона* (*TemplateRow*), производный от родительского класса [Объект](#), используется для работы со строками шаблонов Студии. [в разделе Объект](#).

Непосредственно в классе *СтрокаШаблона* определены следующие свойства:

-  [Поле Владелец / Owner](#)
-  [Поле Имя / Name](#)
-  [Поле Надпись / Caption](#)
-  [Поле Номер / Number](#)
-  [Поле Виден / Visible](#)
-  [Поле Печатать / Printed](#)
-  [Поле АвтоВысота / AutoHeight](#)
-  [Поле Высота / Height](#)
-  [Поле МаксВысота / MaxHeight](#)
-  [Поле МинВысота / MinHeight](#)
-  [Поле МожноМенятьВысоту / CanHeightModify](#)
-  [Поле МожноСкрывать / CanVisibleModify](#)
-  [Поле НеМожетБытьПервойНаСтранице / NotFirstOnPage](#)
-  [Поле НоваяСтраница / NewPage](#)
-  [Поле Сверху / Top](#)

Внимание! Создать объект класса *СтрокаШаблона* напрямую нельзя. Строки шаблонов создаются только внутри системы и используются как свойства других объектов (класса [СекцияШаблона](#)), причем такие свойства имеют атрибут "только чтение" (им нельзя присвоить новое значение). Вместе с тем, программа на ТБ.Скрипт может получить доступ к объекту *СтрокаШаблона*, являющемуся свойством родительского объекта, и изменять свойства строки.

Описание

МожноМенятьВысоту :Логическое;
CanHeightModify :Logical;

Назначение

Данное свойство при условии, что оно равно Истина, позволяет пользователю изменять высоту текущей строки. Причем, менять высоту конкретной строки можно в пределах, заданных свойствами [МинВысота](#) и [МаксВысота](#).

Описание

АвтоВысота :Логическое;
AutoHeight :Logical;

Назначение

Данное свойство позволяет определять высоту строки секции программой автоматически в зависимости от параметров используемых в ней шрифтов.

Значение свойства **АвтоВысота** = Истина соответствует установке флага **Автоматическое определение высоты** в диалоге "[Свойства строки](#)".

Описание

Виден :Логическое;
Visible :Logical;

Назначение

Позволяет узнать, видна ли строка шаблона, а также управлять ее видимостью, присваивая значения TRUE (строка видна) и FALSE (строка скрыта). Когда устанавливается значение данного свойства, синхронно изменяется и свойство [Печатать](#).

Пример

```
-- глобально описанные интерфейсные объекты шаблона
СекцЦены : Section;
Bevel2 : Object;

-- ... внутри обработчика некоторого события
-- меняем высоту внешней рамки секции по ее размеру
var h: Numeric;
var n: Integer;
h=0;
for n=1..СекцЦены.RowsCount do
  -- устанавливаем видимость строк
  СекцЦены.Row[n].Visible = ОбщиеНастройкиЦен.Показать[n];
  СекцЦены.Row[n].Printed = ОбщиеНастройкиЦен.Печать[n];
  -- проверяем видима ли строка
  if СекцЦены.Row[n].Visible then
    h=h+СекцЦены.Row[n].Высота;
  end;
end;
-- подгоняем размер рамки
Bevel2.Высота=h*6+6.7;
```

Описание

Владелец :СекцияШаблона;
Owner :TemplateSection;

Назначение

Содержит ссылку на секцию, которой принадлежит строка. Поле доступно только на чтение.

Предупреждение. Старое название Секция/Section текущего свойства Владелец/Owner сохранено в качестве синонима, для совместимости данных.

Пример

```
-- изменяем параметры секции указанной строки
proc CountRows(RR: TemplateRow);
var SS: TemplateSection;
  SS = RR.Owner;
  -- если число строк не достигло лимита
  -- и задана первая строка, тогда добавляем еще одну строку
  if SS.RowsCount < 10 and RR.Number = 1 then
    SS.RowsCount = SS.RowsCount + 1;
  end;
end;
```

Описание

Высота : Число;
Height : Numeric;

Назначение

Позволяет узнать высоту строки шаблона и изменить ее. Высота измеряется в миллиметрах.

Пример

```
-- глобально описанные интерфейсные объекты шаблона
СекцЦены : Section;
Bevel2 : Object;

-- ... внутри обработчика некоторого события
-- меняем высоту внешней рамки секции по ее размеру
var h: Numeric;
var n: Integer;
h=0;
for n=1..СекцЦены.RowsCount do
    -- устанавливаем видимость строк
    СекцЦены.Row[n].Visible = ОбщиеНастройкиЦен.Показать[n];
    СекцЦены.Row[n].Printed = ОбщиеНастройкиЦен.Печать[n];
    if СекцЦены.Row[n].Visible then
        h=h+СекцЦены.Row[n].Высота; -- суммируем высоты всех строк
    end;
end;
-- подгоняем размер рамки
Bevel2.Высота=h*6+6.7;
```

Описание

Имя:Строка;
Name :String;

Назначение

Поле позволяет узнать и изменить имя строки шаблона. Имя может быть не задано и не обязано быть уникальным, однако в этих случаях теряется смысл данного поля, обеспечивающего идентификацию строки по имени.

Пример

```
proc TotalRowCheck(SS: TemplateSection);
var i, n: Integer;
    n = SS.RowsCount;
    -- ищем в цикле строки с именем "Итого"
    for i = 1..n do
        if SS.Row[i].Name = "Итого" then
            -- найденные строки выделяем жирным с помощью
            -- пользовательской функции MakeRowInBold
            MakeRowInBold(SS.Row[i]);
        end;
    end;
end;
```

Описание

МаксВысота :Число;
MaxHeight :Numeric;

Назначение

Данное свойство совместно со свойством [МинВысота/MinHeight](#) определяет пределы изменения высоты текущей строки секции в интерактивном режиме. Причем, изменять высоту строки можно только при условии, что свойство [МожноМенятьВысоту](#) имеет значение True.

Описание

МинВысота :Число;
MinHeight :Numeric;

Назначение

Данное свойство совместно со свойством [МаксВысота](#) определяет пределы изменения высоты текущей строки секции в интерактивном режиме. Причем, изменять высоту строки можно только при условии, что свойство [МожноМенятьВысоту](#) = True.

Описание

`МожноСкрывать` :Логическое;
`CanVisibleModify` :Logical;

Назначение

Позволяет скрывать и отображать строки текущей секции, для которых свойство [МожноСкрывать](#) = True.

Причем, если в текущей секции имеются такие строки или столбцы, то доступна команда [Настроить видимость](#) контекстного меню, которая открывает диалог "Настройка видимости".

Описание

Надпись :Строка;
Caption :String;

Назначение

Поле позволяет узнать и изменить надпись к строке шаблона, которая указывается в поле **Надпись** диалога "[Свойства строки](#)". Кроме этого, данное свойство используется в диалоге [Настроить видимость](#).

Если значение поля не пустое, то оно используется для задания имен строк, которые перечислены в указанном диалоге, иначе в качестве имени берется значение из свойства [Имя/Name](#). Если и оно пустое, то указывается порядковый номер строки.

Описание

НеМожетБытьПервойНаСтранице :Логическое;
NotFirstOnPage :Logical;

Назначение

Данное свойство влияет на разбивку страниц при печати. Если строка, у которой свойство **NotFirstOnPage** равно *Истина*, не помещается целиком на странице, то она будет перенесена на следующую страницу только совместно со строками, которые не имеют этого свойства (Ложь) и лежат выше данной строки. Поэтому, если при формировании страницы у всех строк секции это свойство будет равно *Истина*, то перенос строк с учетом этого свойства будет проигнорирован.

Аналогом данного свойства является флаг **Не первая на странице** в диалоге "[Свойства строки](#)". Причем в диалоге одновременно может быть установлен только один из флагов, либо флаг **Не первая на странице**, либо флаг **Печатать с новой страницы**.

Описание

НоваяСтраница :Логическое;

NewPage :Logical;

Назначение

Данное свойство позволяет при печати бланка переводить страницу перед данной строкой, т.е. печатать эту строку с новой страницы. Аналогичные действия выполняются при установке флага **Печатать с новой страницы** в диалоге ["Свойства строки"](#).

Описание

Номер :Число;
Number :Numeric;

Назначение

Поле содержит порядковый номер строки в секции. Номера имеют значения от 1 до общего числа строк в секции (в кадре повторяющейся секции). Поле доступно только на чтение.

Пример

```
-- изменяем параметры секции указанной строки
proc CountRows(RR: TemplateRow);
var SS: TemplateSection;
  SS = RR.Section;
  -- если число строк не достигло лимита
  -- и задана первая строка, тогда добавляем еще одну строку
  if SS.RowCount < 10 and RR.Number = 1 then
    SS.RowCount = SS.RowCount + 1;
  end;
end;
```

Описание

Печатать :Логическое;
Printed :Logical;

Назначение

Позволяет узнать, будет ли выводиться строка шаблона на печать, а также управлять этим аспектом, присваивая значения TRUE (строка печатается) и FALSE (строка не печатается). Данное свойство автоматически устанавливается, когда меняется свойство [Виден](#).

Пример

```
-- глобально описанные интерфейсные объекты шаблона
СекцЦены : Section;
Bevel2 : Object;

-- ... внутри обработчика некоторого события
-- меняем высоту внешней рамки секции по ее размеру
var h: Numeric;
var n: Integer;
h=0;
for n=1..СекцЦены.RowsCount do
    СекцЦены.Row[n].Visible = ОбщиеНастройкиЦен.Показать[n];
-- устанавливаем признак вывода на печать текущей строки
    СекцЦены.Row[n].Printed = ОбщиеНастройкиЦен.Печать[n];
    if СекцЦены.Row[n].Visible then
        h=h+СекцЦены.Row[n].Высота;
    end;
end;
-- подгоняем размер рамки
Bevel2.Высота=h*6+6.7;
```

Описание

Сверху :Число;

Тор :Numeric;

Назначение


Свойство возвращает расстояние в миллиметрах от верхнего края шаблона без учета отступов шаблона от края окна до верхней границы строки. Свойство может динамически менять свое значение, если выше данной строки есть строки, изменившие свою высоту или видимость.










Класс *ФреймШаблона / TemplateFrame*, производный от родительского класса [Объект](#), наследует все его свойства и используется для работы с фреймами шаблонов.



Фрейм - это область шаблона, имеющая уникальное имя и набор атрибутов, которые задаются при создании нового фрейма. На шаблоне всегда присутствует корневой фрейм с предопределенным именем *RootFrame*. Фреймы на шаблоне могут быть представлены в иерархическом виде. Под *субфреймом* понимается фрейм, вложенный в другой фрейм.

Внимание! Для создания объектов данного класса используются функции *ВставитьФрейм* и *ДобавитьФрейм* этого класса.

Непосредственно в классе *ФреймШаблона* определены следующие свойства:

-  [Поле Имя / Name](#)
-  [Поле Надпись / Caption](#)
-  [Поле Подсказка / Hint](#)
-  [Поле КонтекстПомощи / HelpContext](#)
-  [Поле Владелец / Owner](#)
-  [Поле РодФрейм / ParentFrame](#)
-  [Поле РодИндекс / ParentIndex](#)
-  [Поле Цвет / Color](#)
-  [Поле Виден / Visible](#)
-  [Поле Печатать / Printed](#)
-  [Поле ИспользоватьРазделитель / UseSplitter](#)
-  [Поле РазделительСправа / SplitterAtRight](#)
-  [Поле РазделительВыпуклый / SplitterRaised](#)
-  [Поле МожноМенятьРазмеры / CanResizeBySplit](#)
-  [Поле МожноСкрывать / CanHideBySplit](#)
-  [Поле МожноВыделятьБлок / CanSelectBlock](#)
-  [Поле СтилСкроллера / ScrollerStyle](#)
-  [Поле Ориентация / Orientation](#)
-  [Поле АвтоРазмер / AutoSize](#)
-  [Поле Размер / Size](#)
-  [Поле МинРазмер / MinSize](#)
-  [Поле МаксРазмер / MaxSize](#)
-  [Поле Отступ / Margins](#)
-  [Поле ОтступДоп / MarginsEx](#)
-  [Поле Бордю / Bevel](#)
-  [Поле ПоказыватьЗакладки / ShowTabs](#)
-  [Поле АвтоРазмерЗакладок / TabAutoSize](#)
-  [Поле КлеткаПоПолю / CellByField](#)
-  [Поле ТекущийФрейм / CurrentFrame](#)
-  [Поле КоличествоФреймов / FramesCount](#)
-  [Поле Фрейм / Frame](#)
-  [Поле КоличествоОбъектов / ObjectsCount](#)
-  [Поле Объект / Object](#)
-  [Поле КоличествоСекций / SectionsCount](#)
-  [Поле Секция / Section](#)
-  [Поле Заморожен / Freezed](#)
-  [Поле ВстроеннаяФорма / InplaceForm](#)
-  [Функция ДобавитьФрейм / AddFrame](#)
-  [Функция ВставитьФрейм / InsertFrame](#)
-  [Процедура УдалитьФрейм / DeleteFrame](#)
-  [Функция ДобавитьСекцию / AddSection](#)
-  [Функция ВставитьСекцию / InsertSection](#)

-  [Процедура УдалитьСекцию / DeleteSection](#)
-  [Функция ДобавитьОбъект / AddObject](#)
-  [Функция ВставитьОбъект / InsertObject](#)
-  [Процедура УдалитьОбъект / DeleteObject](#)
-  [Процедура ЗаморозитьОбласть / FreezeArea](#)
-  [Процедура ВзятьОбластьЗаморозки / GetFreezeArea](#)
-  [Процедура РазморозитьОбласть / UnFreezeArea](#)
-  [Процедура Обновить / Update](#)
-  [Функция Очистить / Clear](#)

-  [Функция ПриПереключении / OnSwitch](#)
-  [Функция ПриСкрытии / OnHide](#)

Описание

АвтоРазмер : Логическое;
AutoSize : Logical;

Назначение

Разрешает или запрещает автоматическое вычисление размеров фрейма в зависимости от его содержимого.

Пример

```
var FrTempl :TemplateFrame;  
....  
FrTempl=Template.CurrentFrame;  
FrTempl.AutoSize = true;
```


Описание

АвтоРазмерЗакладок : Логическое;
TabAutoSize : Logical;

Назначение

Свойство позволяет включать (True) и отключать (False) масштабирование закладок фреймов. По умолчанию свойство равно False, т. е. режим масштабирования закладок выключен.

Свойство выполняется, если задан режим показа закладок, т.е. значение свойства [ShowTabs](#), задано равным True.

Пример

```
var FrTempl :TemplateFrame;  
....  
FrTempl=Template.CurrentFrame;  
if FrTempl.ShowTabs then  
    FrTempl.TabAutoSize=true;  
end;
```

Описание

Бордюор : [Шаблон.СтилиБордюора](#);
Bevel : [Template.BevelStyles](#);

Назначение

Свойство позволяет определить или установить вокруг фрейма один из доступных стилей бордюра, который определен в перечислимом типе [СтилиБордюора/BevelStyles](#).

При выборе значения *NoneBevel* бордюор вокруг фрейма отсутствует, в остальных случаях бордюор прорисовывается одним из следующих стилей:

- по умолчанию | DefaultBevel
- тонкий | SingleBevel
- статический | StaticBevel
- внутренний | ClientBevel
- оконный | WindowBevel

Поле доступно на чтение и запись.

Пример

```
var FrTempl :TemplateFrame;  
....  
FrTempl=Template.CurrentFrame;  
FrTempl.Bevel = Template.SingleBevel;
```

Описание

Виден :Логическое;
Visible : Logical;

Назначение

Свойство скрывает (False) или показывает фрейм (True) на экране. Поле доступно на чтение и запись.

Пример

```
var FrTempl :TemplateFrame;  
....  
FrTempl=Template.CurrentFrame;  
FrTempl.Visible = true;
```

Описание

Владелец : [Шаблон](#);
Owner : [Template](#);

Назначение

Возвращает указатель на объект класса **Шаблон/Template**, т.е. на шаблон, которому принадлежит данный фрейм.

Поле доступно только на чтение.

Пример

```
proc КоличествоОбъектов(Fr :TemplateFrame);  
  var N, j : Integer;  
  -- определяем количество объектов,  
  -- размещенных на текущем фрейме  
  ...  
  N=Fr.Owner.ObjectsCount;  
  Message("Количество = " + N);  
end;
```

Описание

```
ВстроеннаяФорма : Форма;  
InplaceForm : Form;
```

Назначение

Возвращает ссылку на форму, если она была подгружена во фрейм, иначе - nil. Поле доступно только на чтение.

Пример

```
var aForm :Form;  
....  
aForm=Template.CurrentFrame.InplaceForm;
```

Описание

Заморожен : Логическое;
Freezed : Logical;

Назначение

Свойство позволяет получить информацию о наличии на фрейме в данный момент замороженной области. Если значение поле равно true, то на фрейме шаблона имеется [замороженная область](#) и можно [узнать ее размеры](#). После выполнения процедуры [РазморозитьОбласть](#) значение поля становится равным false.

Поле доступно только на чтение.

Пример

```
var FrTempl :TemplateFrame;  
....  
FrTempl=Template.CurrentFrame;  
FrTempl.Freezed = true;
```

Описание

Имя : Строка;
Name : String;

Назначение

Позволяет определить или задать уникальное имя фрейма, по которому он будет идентифицироваться программой.

Пример

```
var N, j : Integer;
var аИмя : String;
var FrRoot :TemplateFrame;
....
FrRoot=Template.RootFrame;
N=FrRoot.FramesCount;
-- выводим имена фреймов, вложенных в корневой фрейм
for j= 1.. N do
    аИмя=FrRoot.Frame[j].Name;
    Message("Номер фрейма= "+ Стр(j)+ " Имя = " + аИмя);
end;
```

Описание

ИспользоватьРазделитель : Логическое;
UseSplitter : Logical;

Назначение

Данное свойство позволяет отображать и скрывать разделитель между фреймами. Местоположение разделителя определяется свойством [РазделительСправа](#). Разделитель отображается на шаблоне, если значение поля [ИспользоватьРазделитель](#) равно TRUE.

Пример

```
Template.RootFrame.Orientation=Template.WithTabs;
...
proc Pl(TF :TemplateFrame; aName:String);
  var TFNew :TemplateFrame;
  -- добавить фрейм
  TFNew=TF.AddFrame;
  TFNew.Имя=aName;
  TFNew.UseSplitter=true;
end;
```


Описание

```
КлеткаПоПолю[Поле :Строка] :КлеткаШаблона;  
CellByField[Field :String] :TemplateCell;
```

Аргументы

Поле - имя поля шаблона.

Назначение

Свойство позволяет получить ссылку на объект [КлеткаШаблона](#) по заданному имени поля (переменной), содержащейся в клетке. Поиск производится по всем секциям заданного фрейма шаблона. Имя задается в качестве индекса. Если в секциях фрейма шаблона есть несколько клеток для одноименных полей, будет возвращена первая (с учетом следования секций в шаблоне и нумерации клеток в каждой из них). Если такого поля нет, возвращается **nil**.

Свойство доступно только на чтение.

Пример

```
-- фрагмент кода бланка  
var TF :TemplateFrame;  
....  
TF=Template.CurrentFrame;  
TF.КлеткаПоПолю["Итого"].Font.Bold = Истина;
```

Описание

КоличествоОбъектов : Целое;
ObjectsCount : Integer;

Назначение

Поле хранит количество интерфейсных объектов, то есть объектов классов, производных от класса [ОбъектШаблона](#), расположенных на фрейме шаблона любого уровня вложенности.

Поле доступно только на чтение.

Пример

```
var N, j :Integer;
var i, k :Integer;
var Имя   : String;
var FrRoot :TemplateFrame;

FrRoot=Template.RootFrame;
N=FrRoot.FramesCount;
-- получаем количество объектов для каждого
-- фрейма первого и второго уровней вложенности
for j= 1.. N do -- фреймы первого уровня
  k=FrRoot.Frame[j].ObjectsCount;
  Message("Номер="+Стр(j)+ " . Количество объектов = "+Стр(k));
  for i= 1.. k do -- фреймы второго уровня
    Message("Номер="+Стр(i)+ " . Количество объектов = "+
      Стр(FrRoot.Frame[j].Frame[i].ObjectsCount));
  end;
end;
```

Описание

КоличествоСекций : Целое;
SectionsCount : Integer;

Назначение

Данное поле позволяет определить количество секций на фрейме шаблона. Поле доступно только на чтение.

Пример

```
var N :Integer;  
...  
N = Template.CurrentFrame.SectionsCount;
```

Описание

КоличествоФреймов : Целое;
FramesCount : Integer;

Назначение

Поле хранит количество фреймов, вложенных в заданный фрейм. Поле доступно только на чтение.

Пример

```
var N,j:Integer;
var i,k,Колич :Integer;
var Имя : String;
var FrRoot :TemplateFrame;

FrRoot=Template.RootFrame;
N=FrRoot.FramesCount;
for j= 1.. N do
  k=FrRoot.Frame[j].FramesCount;
  Message("Номер="+Стр(j)+ " Количество фреймов = " + Стр(k));
  for i= 1.. k do
    Message("Номер="+Стр(i)+ " Количество фреймов = " +
      Стр(FrRoot.Frame[j].Frame[i].FramesCount));
  end;
end;
```

Описание

```
КонтекстПомощи :Строка;  
HelpContext : String;
```

Назначение

Свойство позволяет определить и задать путь к файлу с текстом помощи, поясняющим назначение данного фрейма.

Этот текст будет появляться в специальном всплывающем окне, если на экране открыт данный фрейм, нажата клавиша *F1*, а фокус установлен на каком-либо объекте заданного фрейма.

Если поле пустое, то справка к данному фрейму берется для [всего шаблона](#).

Пример

```
var RF :TemplateFrame;  
....  
RF=Template.RootFrame;  
RF.Orientation=Template.WithTabs;  
RF.ShowTabs=true;  
RF.HelpContext=" ";
```

Описание

МаксРазмер : Число;
MaxSize : Numeric;

Назначение

Свойство ограничивает [размер фрейма](#) сверху. Поле доступно на чтение и запись.

Пример

```
var FrTempl :TemplateFrame;  
var FrRoot :TemplateFrame;  
  
FrRoot=Template.RootFrame;  
Template.RootFrame.Orientation=Template.WithTabs;  
-- добавить фрейм в корень  
FrTempl=FrRoot.AddFrame;  
FrTempl.Имя="ФреймКнопки";  
FrTempl.Надпись="Кнопки";  
FrTempl.MaxSize=150;
```

Описание

МинРазмер : Число;
MinSize : Numeric;

Назначение

Свойство ограничивает [размер фрейма](#) снизу. Поле доступно на чтение и запись.

Пример

```
var FrTempl :TemplateFrame;  
var FrRoot :TemplateFrame;  
  
FrRoot=Template.RootFrame;  
Template.RootFrame.Orientation=Template.WithTabs;  
-- добавить фрейм в корень  
FrTempl=FrRoot.AddFrame;  
FrTempl.Имя="ФреймКнопки";  
FrTempl.Надпись="Кнопки";  
FrTempl.MinSize=30;
```

Описание

МожноВыделятьБлок : Логическое;
CanSelectBlock : Logical;

Назначение

Свойство позволяет выделять на фрейме группу клеток (блок), принадлежащих одной секции. По умолчанию свойство равно True.

Пример

```
var FrTempl :TemplateFrame;  
....  
FrTempl=Template.CurrentFrame;  
FrTempl.CanSelectBlock = true;
```


Описание

МожноМенятьРазмеры : Логическое;
CanResizeBySplit : Logical;

Назначение

Если значение поле установлено равным TRUE, то в режиме исполнения разрешается изменять размеры фрейма, перемещая разделитель мышью, иначе - не разрешается. Свойство выполняется, если значение поля [ИспользоватьРазделитель](#) равно True.

Поле доступно на чтение и запись.

Пример

```
var FrTempl :TemplateFrame;  
....  
FrTempl=Template.CurrentFrame;  
if FrTempl.UseSplitter then  
    FrTempl.CanResizeBySplit=true;  
end;
```

Описание

МожноСкрывать :Логическое;
CanHideBySplit :Logical;

Назначение

Если значение поля установлено равным True, то в режиме исполнения разрешается сделать фрейм невидимым, щелкнув мышью на разделителе в том месте, где находится так называемая область закрытия, которая окрашивается в желтый цвет (см. рис.1).



Рис 1. Горизонтальная область закрытия

Свойство доступно на чтение и запись, оно выполняется в том случае, когда значение поля [ИспользоватьРазделитель](#) равно True.

Пример

```
var FrTempl :TemplateFrame;  
....  
FrTempl=Template.CurrentFrame;  
if FrTempl.UseSplitter then  
    FrTempl.CanHideBySplit=true;  
end;
```

Описание

Надпись :Строка;
Caption : String;

Назначение

Данное поле позволяет узнать и изменить текст надписи, поясняющий назначение фрейма. Поле доступно на чтение и запись.

Пример

```
var N, j : Integer;  
var aCaption : String;  
var FrRoot :TemplateFrame;  
....  
FrRoot=Template.RootFrame;  
-- выводим содержание полей Надпись для фреймов,  
-- которые вложены в корневой фрейм  
N=FrRoot.FramesCount;  
for j= 1.. N do  
    aCaption=FrRoot.Frame[j].Caption;  
    Message("Номер фрейма = "+Стр(j)+ " Надпись = " +aCaption);  
end;
```

Описание

Объект [Индекс :Целое] : [ОбъектШаблона](#);
Object[Index :Integer] : [TemplateObject](#);

Аргументы

Индекс - порядковый номер объекта. Объекты нумеруются от 1 до числа объектов расположенных на фрейме шаблона, которое возвращается свойством [КоличествоОбъектов](#).

Назначение

Поле хранит ссылки на объекты классов, производных от класса [ОбъектШаблона](#). Поле доступно только на чтение.

Для работы с объектами используются функции [ДобавитьОбъект](#) и [ВставитьОбъект](#) и процедура [УдалитьОбъект](#).

Пример

```
var N, j : Integer;  
varOb : TemplateObject;  
...  
N=Template.CurrentFrame.ObjectsCount;  
-- получаем ссылку на объект  
if N > 1 then  
  Ob = Template.CurrentFrame.Object[1];  
end;
```

Описание

Ориентация : [Шаблон.ТипыОриентации](#);
Orientation : [Template.OrientationTypes](#);

Назначение

Свойство позволяет определить или установить один из возможных способов размещения фреймов (горизонтально, вертикально, с закладками) на шаблоне, который определен в перечислимом типе [ТипыОриентации](#).

Пример

```
-- ориентация с закладками  
Template.RootFrame.Orientation=Template.WithTabs;
```

Описание

Отступ : Целое;
Margins : Integer;

Назначение

Свойство позволяет определить или установить величину отступа содержимого от края фрейма. Величина отступа измеряется в **пикселях**.

Поле доступно на чтение и запись.

Пример

```
var FrTempl :TemplateFrame;  
var FrRoot :TemplateFrame;  
  
FrRoot=Template.RootFrame;  
Template.RootFrame.Orientation=Template.WithTabs;  
-- добавить фрейм в корень  
FrTempl=FrRoot.AddFrame;  
FrTempl.Имя="ФреймКнопки";  
FrTempl.Надпись="Кнопки";  
FrTempl.Margins=5;
```

Описание

ОтступДоп : Целое[];
MarginsEx : Integer[];

Назначение

Свойство позволяет определить или установить отступы для каждой из сторон фрейма по отдельности. Величины отступа измеряются в **пикселях**. В поле хранится массив из четырех элементов, задающих величину отступа слева, сверху, справа, снизу.

Пример

```
var FrTempl :TemplateFrame;  
var FrRoot :TemplateFrame;  
....  
FrRoot=Template.RootFrame;  
Template.RootFrame.Orientation=Template.WithTabs;  
-- добавить фрейм в корень  
FrTempl=FrRoot.AddFrame;  
FrTempl.Имя="Фрейм1";  
FrTempl.MarginsEx = [50,100,50,5];
```

Описание

Печатать : Логическое;
Printed : Logical;

Назначение

Свойство разрешает или запрещает печатать текущий фрейм. Поле доступно на чтение и запись.

Пример

```
Template.CurrentFrame.Printed=true;
```


Описание

Hint :Строка;
Подсказка : String;

Назначение

Позволяет узнать и изменить строку с текстом подсказки, которая появляется при выполнении следующих условий:

- выбран способ ориентации фреймов - с закладками, т.е. в поле [Ориентация](#) задано значение Шаблон.СЗакладками,
- разрешено [показывать закладки](#),
- и курсор установлен на закладке фрейма.

Пример

```
var RF :TemplateFrame;  
....  
RF=Template.RootFrame;  
RF.Orientation=Template.WithTabs;  
RF.ShowTabs=true;  
RF.Hint="Текст подсказки";
```

Описание

ПоказыватьЗакладки : Логическое;
ShowTabs : Logical;

Назначение

Свойство позволяет определить или установить режим видимости закладок на шаблоне. Свойство выполняется, если выбрана ориентация размещения фреймов ["С закладками"](#). Если значение свойство равно True, то закладки отображаются на шаблоне, в этом случае можно задать свойство [АвтоРазмерЗакладок](#). Закладки не видны, если свойство равно False.

Поле доступно на чтение и запись.

Пример

```
Template.RootFrame.Orientation=Template.WithTabs;  
Template.RootFrame.ShowTabs=true;
```

Описание

РазделительВыпуклый : Логическое;
SplitterRaised : Logical;

Назначение

Свойство определяет внешний вид разделителя. Если значение поле установлено равным TRUE, то разделитель является выпуклым, иначе - плоским. Данное свойство работает, при условии, что значение поля [ИспользоватьРазделитель](#) задано, равным TRUE.

Пример

```
var FrTempl :TemplateFrame;  
....  
FrTempl=Template.CurrentFrame;  
if FrTempl.UseSplitter then  
    FrTempl.SplitterRaised=true;  
    FrTempl.SplitterAtRight=true;  
end;
```

Описание

РазделительСправа : Логическое;
SplitterAtRight : Logical;

Назначение

Свойство определяет местоположение разделителя. Если значение поле установлено равным TRUE, то разделитель отображается справа (при вертикальной ориентации) или снизу (при горизонтальной ориентации), иначе - слева/вверху.

Свойство доступно на чтение и запись и работает, если значение поля [ИспользоватьРазделитель](#) равно TRUE.

Пример

```
var FrTempl :TemplateFrame;  
....  
if FrTempl.UseSplitter then  
    FrTempl.SplitterRaised=true;  
    FrTempl.SplitterAtRight=true;  
end;
```

Описание

Размер : Число;
Size : Numeric;

Назначение

Позволяет узнать размер фрейма в мм и при необходимости изменить его.

Пример

```
var FrTempl :TemplateFrame;  
var FrRoot :TemplateFrame;  
  
FrRoot=Template.RootFrame;  
Template.RootFrame.Orientation=Template.WithTabs;  
-- добавить фрейм в корень  
FrTempl=FrRoot.AddFrame;  
FrTempl.Имя="ФреймКнопки";  
FrTempl.Надпись="Кнопки";  
FrTempl.Size=200;
```

Описание

РодИндекс : Целое;
ParentIndex : Integer;

Назначение

Поле предназначено для чтения порядкового номера данного фрейма во внутреннем списке родительского фрейма. Список нумеруются, начиная с 1.

Поле доступно только на чтение.

Пример

```
var FrTempl :TemplateFrame;  
var FrRoot :TemplateFrame;  
...  
FrRoot=Template.RootFrame;  
-- добавляем фрейм и выводим его индекс  
FrTempl=FrRoot.AddFrame;  
Hint(FrTempl.ParentIndex);
```

Описание

РодФрейм : [ФреймШаблона](#);
Parent : [TemplateFrame](#);

Назначение

Возвращает указатель на родительский фрейм. Если родительского фрейма не существует (например, RootFrame), возвращает nil. Поле доступно только на чтение.

Пример

```
proc Pl(FrTempl : TemplateFrame);  
var TF :TemplateFrame;  
  TF=FrTempl.ParentFrame;  
  if TF <> nil then  
    Message(Количество фреймов = " + Стр(TF.FramesCount));  
  fi;  
end;
```

Описание

Секция[Индекс :Целое] : [СекцияШаблона](#);
Section[Index :Integer] : [TemplateSection](#);

Назначение

Данное поле позволяет получить доступ к существующей секции по ее номеру. Поле доступно только на чтение. Для работы с секциями используются функции [ДобавитьСекцию](#) и [ВставитьСекцию](#) и процедура [УдалитьСекцию](#).

Пример

```
var N : Integer;  
var TS :TemplateSection;  
...  
N = Template.CurrentFrame.SectionsCount;  
-- получаем ссылку на секцию  
if N > 1 then  
    TS = Template.CurrentFrame.Section[1];  
end;
```


Описание

СтильСкроллера : [Шаблон.СтилиБлоков](#);
ScrollerStyle : [Template.BlockStyles](#);

Назначение

Свойство позволяет определить или установить один из возможных способов отображения/скрытия на фрейме полос прокрутки, можно отображать как обе полосы вместе, так и отдельно одну из них или вообще скрыть их. Видимость полос прокрутки зависит от выбранной константы перечислимого типа [СтилиБлоков](#).

Пример

```
var FrTempl :TemplateFrame;  
....  
FrTempl=Template.CurrentFrame;  
FrTempl.ScrollerStyle=Template.ScrollerBoth;
```

Описание

ТекущийФрейм : [ФреймШаблона](#);
CurrentFrame : [TemplateFrame](#);

Назначение

Свойство позволяет узнать текущий субфрейм у фрейма, а также изменить его. Под субфреймом понимается фрейм, вложенный в другой фрейм. Если текущий фрейм не будет являться дочерним к родительскому, возникнет исключение.

Поле доступно на чтение и запись.

Пример

```
var TF :TemplateFrame;  
....  
TF=Template.CurrentFrame;
```

Описание

Фрейм[Индекс :Целое] : [ФреймШаблона](#);
Frame[Index :Integer] : [TemplateFrame](#);

Аргументы

Индекс - [порядковый номер](#) фрейма. Он может изменяться в пределах от 1 до количества фреймов, которое возвращается свойством [КоличествоФреймов](#).

Назначение

Возвращает ссылку на фрейм, заданный по его номеру.

Пример

```
var N, j : Integer;
var аИмя : String;
var FrRoot :TemplateFrame;
....
FrRoot=Template.RootFrame;
N=FrRoot.FramesCount;
-- выводим имена фреймов, вложенных в корневой фрейм
for j= 1.. N do
    аИмя=FrRoot.Frame[j].Name;
    Message("Номер фрейма= "+ Стр(j)+ " Имя = " + аИмя);
end;
....
```

Описание

Цвет : Целое;
Color : Integer;

Назначение

Позволяет узнать цвет фона заданного фрейма шаблона и/или изменить его.

Пример

```
var TF :TemplateFrame;  
....  
TF=Template.CurrentFrame.Color=clSilver;;
```

Описание

```
ВзятьОбластьЗаморозки(var Левый :Число; var Верхний :Число; var Ширина :Число; var  
Высота :Число);  
GetFreezeArea(var Left :Numeric; var Top :Numeric; var Width :Numeric; var Height :Numeric);
```

Аргументы

Левый, Верхний - координаты в мм. левого верхнего угла области замораживания (см. ниже).
Ширина, Высота - ширина и высота области замораживания в мм.

Назначение

Процедура позволяет определить размеры замороженной области, т.е. фиксированной прямоугольной области в окне с фреймом шаблоном, которая будет неизменна при навигации по фрейму шаблону и скроллинге.

Пример

```
var aLeft,aWidth : Numeric;  
var aTop,aHeight : Numeric;  
var TF :TemplateFrame;  
...  
TF=Template.CurrentFrame;  
TF.GetFreezeArea(aLeft, aTop, aWidth, aHeight);
```

Описание

```
ЗаморозитьОбласть({Левый :Число}; {Верхний :Число}; Ширина :Число; Высота :Число);  
FreezeArea({Left :Numeric}; {Top :Numeric}; Width :Numeric; Height :Numeric);
```

Аргументы

Левый, Верхний - необязательные параметры, задающие координаты в мм. левого верхнего угла области замораживания (см. ниже). Если они опущены, то смещение области заморозки считается равным отрицательному отступу фрейма шаблона.

Ширина, Высота - ширина и высота области замораживания в мм., которые совместно с параметрами Левый, Верхний определяют размер области замораживания.

Назначение

Процедура замораживает область фрейма шаблона, размеры которой задаются параметрами **Левый, Верхний, Ширина, Высота**.

Под областью замораживания понимается фиксированная прямоугольная область на фрейме шаблона, которая будет неизменна при навигации по фрейму шаблона и скроллинге. Область замораживания работает только в режиме исполнения шаблона. Не замороженная область в окне выделяется двойной зеленой пунктирной линией от остальной части окна. Если подвести к этой линии курсор мыши, то область замораживания можно изменять.

Предупреждение. Программа не проверяет заданные размеры области, поэтому может возникнуть ситуация, когда весь экран заморожен даже при максимизированном окне, или область заморозки выходит за границы экрана.

Пример

```
var TF :TemplateFrame;  
...  
TF=Template.CurrentFrame;  
TF.FreezeArea(0,0, 250,70);
```

Описание

Обновить ;
Update ;

Назначение

Процедура предназначена для принудительной перерисовки фрейма шаблона, что может быть полезно в процессе выполнения продолжительных операций внутри обработчиков событий, так как шаблон штатно перерисовывается только по завершении работы обработчика событий.

Пример

```
proc P1(Ob:String);  
var i:integer;  
  for i=1..Documents do  
    -- ... обработка документов,  
    -- влияющая на значения полей шаблона  
    Template.CurrentFrame.Update;  
  end;  
end;
```

Описание

```
РазморозитьОбласть;  
FreezeArea;
```

Назначение

Процедура размораживает область фрейма шаблона, которая была зафиксирована процедурой [ЗаморозитьОбласть / FreezeArea](#). В этом случае значение поля [Заморожен / Freezed](#) становится равным False.

Пример

```
var TF :TemplateFrame;  
...  
TF=Template.CurrentFrame;  
TF.UnFreezeArea;
```


Описание

```
УдалитьОбъект(Индекс :Целое);  
DeleteObject(Index :Integer);
```

Аргументы

Индекс - порядковый номер нового объекта, который может изменяться в пределах от 1 до [количества объектов](#) на заданном фрейме.

Назначение

Функция предназначена для программного удаления заданного объекта, размещенного на фрейме шаблона. В результате уменьшаются на 1 порядковые номера объектов, у которых до выполнения функции порядковые номера были больше или равны заданному порядковому номеру.

Пример

```
var N : Integer;  
var TF :TemplateFrame;  
...  
TF=Template.CurrentFrame;  
N=TF.ObjectsCount;  
-- удаляем последний по номеру объект фрейма  
if N > 0 then  
    TF.DeleteObject[N];  
end;
```

Описание

УдалитьСекцию(Индекс :Целое) : [СекцияШаблона](#);
DeleteSection(Index :Integer): [TemplateSection](#);

Аргументы

Индекс - порядковый номер секции, которую требуется удалить. Значение должно лежать в пределах от 1 до [количества секций](#) на заданном фрейме шаблона.

Назначение

Функция предназначена для программного удаления секции шаблона. После выполнения операции все нижележащие секции сдвигаются вверх, а их индексы уменьшаются на 1.

Назначение

Пример

```
var N : Integer;  
var TF :TemplateFrame;  
...  
TF=Template.CurrentFrame;  
N=TF.SectionsCount;  
if N > 0 then  
    TF.DeleteSection[N];  
end;
```

Описание

```
УдалитьФрейм(Индекс :Целое);  
DeleteFrame(Index :Integer);
```

Аргументы

Индекс - порядковый номер секции, который изменяется в пределах от 1 до [количества фреймов](#), вложенных в заданный фрейм.

Назначение

Функция предназначена для программного удаления заданного фрейма шаблона. В результате уменьшаются на 1 порядковые номера фреймов, у которых до выполнения функции порядковые номера были больше или равны заданному порядковому номеру. Причем, речь идет только о фреймах, вложенных в один и тот же фрейм.

Пример

```
var N : Integer;  
var FrRoot :TemplateFrame;  
...  
FrRoot=Template.RootFrame;  
N=FrRoot.FramesCount;  
if N>0 then  
    FrRoot.DeleteFrame[N];  
end;
```

Описание

ВставитьОбъект(Индекс :Целое; Класс :Класс ОбъектШаблона) : [ОбъектШаблона](#);
InsertObject(Index :Integer; Class :Class TemplateObject) : [TemplateObject](#);

Аргументы

Индекс - порядковый номер нового объекта, который может изменяться в пределах от 1 до [количества объектов](#) на заданном фрейме.

Класс - название класса, производного от класса [ОбъектШаблона](#).

Назначение

Функция предназначена для программного создания и добавления на фрейм шаблона нового объекта, который вставляется перед объектом, индекс которого передан в качестве параметра **Индекс**. После выполнения функции новый объект получит заданный индекс, а все объекты, расположенные за ним в списке, переместятся на одну позицию вниз, т.е. их индекс увеличится на 1.

Пример

```
-- вставляем поле ввода в начало списка объектов фрейма
proc P1(B:Button);
var e1:Edit;
    e1 = Template.CurrentFrame.InsertObject(1,Edit);
end;
```

Описание

```
ВставитьСекцию(Индекс :Целое) : СекцияШаблона;  
InsertSection(Index :Integer) : TemplateSection;
```

Аргументы

Индекс - позиция, в которую требуется вставить новую секцию. Значение должно лежать в пределах от 1 до [количества секций](#) на фрейме шаблоне.

Назначение

Функция предназначена для программного создания и добавления на фрейм шаблона новой секции, которая добавляется перед секцией, индекс которой передан в качестве параметра. После выполнения функции новая секция получит тот же индекс, а индексы всех секций, номера которых до выполнения операции были меньше заданного индекса останутся неизменными, а номера остальных секций увеличатся на 1.

Пример

```
-- по нажатию кнопки  
-- вставляем секцию в начало шаблона  
proc PressMoreButton(B:Button);  
var t:TemplateSection;  
    t = Template.CurrentFrame.InsertSection[1];  
end;
```

Описание

ВставитьФрейм(Индекс :Целое) : [ФреймШаблона](#);
InsertFrame (Index :Integer) : [TemplateFrame](#);

Аргументы

Индекс - позиция, в которую требуется вставить новый фрейм. Значение должно лежать в пределах от 1 до [количества секций](#) на фрейме шаблоне.

Назначение

Функция предназначена для программного создания и добавления нового фрейма в дерево фреймов на шаблоне в ту позицию, которая указана в индексе. Номера индексов всех остальных фреймов, начиная с заданного индекса увеличатся на 1.

Пример

```
-- вставляем новый фрейм в начало дерева фреймов
proc PressMoreButton(B:Button);
  var FrTempl :TemplateFrame;
  FrTempl = Template.CurrentFrame.InsertFrame[1];
end;
```

Описание

```
ДобавитьОбъект(Класс : Класс ОбъектШаблона) : ОбъектШаблона;  
AddObject(Class : Class TemplateObject) : TemplateObject;
```

Аргументы

Класс - имя класса добавляемого объекта. Причем, класс должен быть производным от класса [ОбъектШаблона](#).

Назначение

Функция предназначена для программного создания и добавления на заданный фрейм шаблона нового объекта. Объект добавляется в конец списка объектов фрейма. По умолчанию объект появляется в верхнем левом углу фрейма.

Пример

```
-- по нажатию кнопки  
-- добавляем еще одну кнопку  
proc P1(B:Button);  
var b2:Button;  
    b2 = Template.CurrentFrame.AddObject(Button);  
    b2.Caption = "Перерасчет";  
    b2.Left = 10;  
    b2.Top = 5;  
end;
```

Описание

ДобавитьСекцию : [СекцияШаблона](#);

AddSection : [TemplateSection](#);

Назначение

Функция предназначена для программного создания и добавления на заданный фрейм шаблона новой секции. Секция добавляется после последней из существующих секций.

Пример

```
-- по нажатию кнопки
-- добавляем секцию вниз текущего фрейма
proc P1(B:Button);
var TS:TemplateSection;
  TS = Template.CurrentFrame.AddSection;
end;
```


Описание

ДобавитьФрейм : [ФреймШаблона](#);
AddFrame : [TemplateFrame](#);

Назначение

Функция предназначена для программного создания и добавления новой секции на заданный фрейм шаблон, при этом порядковый номер секции на единицу больше, чем количество секций, которые размещались на фрейме до создания новой секции.

Пример

```
var FrTempl :TemplateFrame;  
var FrRoot :TemplateFrame;  
  
FrRoot=Template.RootFrame;  
Template.RootFrame.Orientation=Template.WithTabs;  
-- добавить фрейм на верхний уровень дерева фреймов  
FrTempl=FrRoot.AddFrame;  
FrTempl.Имя="ФреймКнопки";  
FrTempl.Надпись="Кнопки";
```

Описание

Очистить : Логическое;
Clear : Logical;

Назначение

Функция возвращает значение True, если фрейм был очищен от содержимого без внутренних ошибок, а также, если загруженная во фрейм форма была удалена без проблем, иначе - False.

Пример

```
var TF :TemplateFrame;  
....  
TF=Template.CurrentFrame;  
Clear;
```

Описание

```
ПриПереключении : Строка;  
OnSwitch : String;
```

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при перемещении между фреймами шаблона.

Обработчик

```
func OnSwitch(Frame :TemplateFrame; NewFrame :TemplateFrame) : Logical;
```

Параметры:

Frame - указатель на объект **ФреймШаблона**, в котором меняется текущий дочерний фрейм.

NewFrame - указатель на фрейм, который должен стать текущим.

Если обработчик возвращает True, то переключение между фреймами разрешено, иначе - запрещено (False).

Пример

```
func Frame_OnHide(aFrame :TemplateFrame;  
    aNewFrame :TemplateFrame) :Logical;  
    ....  
    return True;  
end;
```

Описание

ПриСкрытии : Строка;
OnHide : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при скрытии текущего фрейма шаблона.

Обработчик

```
func OnHide(Frame :TemplateFrame) : Logical;
```

Параметры:

Frame - указатель на объект **ФреймШаблона**, в котором произошло событие.

Если обработчик возвращает TRUE, то фрейм не будет отображаться на экране. При входе в обработчик Result равен FALSE (фрейм виден).














































Пример










```
func Frame_OnHide(aFrame :TemplateFrame) :Logical;  
    ....  
    return TRUE;  
end;
```

Класс *Шаблон (Template)*, производный от класса [Объект](#), используется для работы с бланками программы и наследует все свойства и методы родительского класса. Фактически, шаблон – это визуальная форма бланка.

Внимание! Создать объект класса *Шаблон* напрямую нельзя. Шаблоны создаются только самой системой на основе визуальных форм, подготовленных программистом с помощью [редактора шаблонов бланков](#).

















Получить доступ к объекту *Шаблон* и управлять им можно с помощью перечисленных ниже свойств:

-  [Поле Владелец / Owner](#)
-  [Поле Цвет / Color](#)
-  [Поле Строка / Row](#)
-  [Поле Столбец / Column](#)
-  [Поле Кадр / Frame](#)
-  [Поле Поле / Field](#)
-  [Поле Выделение / Selection](#)
-  [Поле КонтекстПомощи / HelpContext](#)
-  [Поле НастройкиПечати / PrintSettings](#)
-  [Поле ТекущийФрейм / CurrentFrame](#)
-  [Поле ТекущийОбъект / CurrentObject](#)
-  [Поле ТекущаяСекция / CurrentSection](#)
-  [Поле ТекущаяКлетка / CurrentCell](#)
-  [Поле ТекущийРедактор / CurrentEdit](#)
-  [Поле КорневойФрейм / RootFrame](#)
-  [Поле ФреймПоИмени / FrameByName](#)
-  [Поле КлеткаПоПолю / CellByField](#)
-  [Поле КоличествоОбъектов / ObjectsCount](#)
-  [Поле Объект / Object](#)
-  [Поле КоличествоСекций / SectionsCount](#)
-  [Поле Секция / Section](#)
-  [Поле КоличествоСтилей / StylesCount](#)
-  [Поле Стил / Style](#)
-  [Поле Заморожен / Freezed](#)
-  [Поле ЦветнаяПечать / ColourPrinting](#)
-  [Функция ДобавитьСекцию / AddSection](#)
-  [Функция ВставитьСекцию / InsertSection](#)
-  [Процедура УдалитьСекцию / DeleteSection](#)
-  [Функция ДобавитьОбъект / AddObject](#)
-  [Функция ВставитьОбъект / InsertObject](#)
-  [Процедура УдалитьОбъект / DeleteObject](#)
-  [Процедура ЗаморозитьОбласть / FreezeArea](#)
-  [Процедура ВзятьОбластьЗаморозки / GetFreezeArea](#)
-  [Процедура РазморозитьОбласть / UnFreezeArea](#)
-  [Функция ДобавитьСтил / AddStyle](#)
-  [УдалитьСтил / DeleteStyle](#)
-  [Процедура НачатьРедактирование / BeginEdit](#)
-  [Процедура ЗавершитьРедактирование / EndEdit](#)
-  [Процедура НачатьМодификацию / BeginModify](#)
-  [Процедура ЗавершитьМодификацию / EndModify](#)
-  [Процедура Обновить / Update](#)
-  [Событие ПриСоздании / OnCreate](#)
-  [Событие ПриОткрытии / OnOpen](#)
-  [Событие ПередЗакрытием / BeforeClose](#)
-  [Событие ПриЗакрытии / OnClose](#)

-  [Событие ПриСчитывании / OnRead](#)
-  [Событие ПриПроверке / OnVerify](#)
-  [Событие ПриЗаписи / OnPost](#)
-  [Событие ПриОтмене / OnCancel](#)
-  [Событие ПриПеремещении / OnMove](#)
-  [Событие ПередИзменением / BeforeModify](#)
-  [Событие ПриКоманде / OnCommand](#)
-  [Событие ПриПодготовкеОтчета / OnPrepareReport](#)
-  [ПриОбновленииОтчета / OnUpdateReport](#)

См. также раздел [Последовательность событий в объектах Шаблон](#).

В классе *Шаблон* определены перечислимые типы, которые используются в других связанных классах (*КлеткаШаблона*, *СтолбецКартотеки*, *СтолбецПодтаблицы*, *ФреймШаблона* и др.) для указания типов полей, их выравнивания, сути событий и пр. К ним относятся:

-  [Константы типов полей \(ТипыПоля / FieldTypes\)](#)
-  [Константы типов клетки \(ТипыКлетки/ CellTypes\)](#)
-  [Константы стилей статического текста \(СтилиСтатическойКлетки / StaticStyles\)](#)
-  [Константы стилей логического поля \(СтилиЛогическогоПоля / LogicalStyles\)](#)
-  [Константы типов выравнивания \(ТипыВыравнивания / AlignmentTypes\)](#)
-  [Константы типов расположения \(ТипыРасположения / AlignTypes\)](#)
-  [Константы типов ориентации \(ТипыОриентации / OrientationTypes\).](#)
-  [Константы стилей бордюра \(СтилиБордюра / BevelStyles\)](#)
-  [Константы типов отступов \(ТипыОтступов / MarginTypes\)](#)
-  [Константы типов нажатий \(ТипыНажатия / ClickTypes\)](#)
-  [Константы типов входа \(ТипыВхода / EnterTypes\)](#)
-  [Константы типов вывода \(ТипыВывода / OutputTypes\)](#)
-  [Константы типов оформления \(ДействияОформления / RearrangeActions\)](#)
-  [Константы форматов поля даты \(ФорматыПоляДаты / DateFormats\)](#)
-  [Константы стилей блоков СтилиБлоков / BlockStyles](#)
-  [Константы стилей скроллера \(СтилиСкроллера /: ScrollerStyles\)](#)
-  [Константы действий изменения \(ДействияИзменения / SearchActions\)](#)

Описание

ПриСоздании : Строка;
OnCreate : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при создании объекта [формы](#) (бланка, картотеки, отчета).

Обработчик

proc **OnCreate**(Context :Variant);

Событие вызывается изнутри конструктора Create и является "логическим продолжением" конструктора. В нем уже доступно программное изменение шаблона. Любое исключение в обработчике **OnCreate** отменяет создание объекта формы.

Внимание. Шаблон (tpl и/или bro файл) считывается не при визуализации формы (бланков, картотек, отчетов), а сразу во время создания объекта. Т.е. после того как конструктор вернул объект формы в нем уже проинициализированы и доступны для модификации свойства Template/Cardfile/Report, а также автоинициализируемые поля, одноименные объектам шаблона. Аналогично, эти поля не обнуляются при закрытии окна, и остаются доступны на все время существования объекта.

Пример

```
Section1 :TemplateSection;  
...  
proc Blank_OnCreate(Context :Variant);  
  var vCell1 :TemplateCell;  
  var I, J :Integer;  
  
  Section1.RowsCount = 100;  
  Section1.ColumnsCount = 100;  
  
  vCell1 = Section1.Cell[1, 1];  
  for I = 1..Section1.ColumnsCount do  
    for J = 1..Section1.RowsCount do  
      if I <> 1 or J <> 1 then  
        Section1.Cell[I, J].Assign(vCell1);  
      end;  
    end;  
  end;  
end;
```

Константы действий изменения (ДействияИзменения / SearchActions)

Перечислимый тип *ДействияИзменения* / *SearchActions*, определенный в классе *Шаблон*, используется для изменения запроса поиска в ячейке шаблона. Перечислимый тип *ДействияИзменения* позволяет разработчику определить, каким способом было вызвано событие [ПриПоиске](#) / [OnSearch](#), и содержит следующие константы:

- **ActionExit** - событие было вызвано при выходе из ячейки;
- **ActionEnter** - событие было вызвано при нажатии клавиши **Enter**).

Константы, определенные в данном типе, используются в событии *ПриПоиске* класса *РедакторПоля* для определения типа действия, вызвавшего это событие.

Перечислимый тип **ДействияОформления / RearrangeActions**, определенный в классе **Шаблон / Template** содержит следующие предопределенные константы, которые описывают возможные действия пользователя, связанные с изменением ширины столбца, высоты строки или их видимости:

- **ШиринаСтолбца / ResizeColumn** - изменена ширина столбца;
- **ВидимостьСтолбца / VisibilityColumn** - столбец сделан видимым или скрыт;
- **ВысотаСтроки / ResizeRow** - изменена высота строки;
- **ВидимостьСтроки / VisibilityRow** - строка видима или скрыта.

Константы данного типа используются в обработчике [ПриОформлении](#) класса **СекцияШаблона**.

Перечислимый тип **СтилиБлоков / BlockStyles**, определенный в классе **Шаблон / Template** предназначен для выбора стиля выделенного блока.

Перечислимый тип **СтилиБлоков** содержит следующие константы:

- ПустойБлок / NoneBlock = 0 - выделение отсутствует;
- СтроковыйБлок / LineBlock = 1 - выделено несколько строк;
- СтолбцовыйБлок / ColumnBlock = 2 - выделено несколько столбцов;
- КвадратныйБлок / BoxBlock = 3 - выделена группа клеток, прямоугольная область, заданная номерами начального столбца и строки, а также номерами конечного столбца и строки.

Константы перечислимого типа **СтилиБлоков** используются в свойстве [Стиль/Style](#) класса **ОбластьШаблона**.

Константы стилей бордюра (СтилиБордюра / BevelStyles))

В классе *Шаблон* определен перечислимый тип *СтилиБордюра / BevelStyles*, содержащий predefined константы, описывающие различные стили бордюра, используемые при работе с шаблонами.

Тип *СтилиБордюра* содержит следующие константы:

- БезБордюра / NoneBevel;
- Бордюроумолчанию / DefaultBevel;
- ТонкийБордюро / SingleBevel;
- СтатическийБордюро / StaticBevel;
- ВнутреннийБордюро / ClientBevel;
- ОконныйБордюро / WindowBevel.

Константы перечислимого типа *СтилиБордюра* применяются при определении свойства *Бордюро/Bevel* в классах:

- [КлеткаШаблона](#)
- [СтильШаблона](#)
- [ФреймШаблона](#)
- [Таблица](#)

Класс Объект : [Шаблон](#)

Константы стилей логического поля (СтилиЛогическогоПоля / LogicalStyles)

Перечислимый тип **СтилиЛогическогоПоля / LogicalStyles**, определенный в классе **Шаблон** содержит предопределенные константы, описывающие способ использования клетки с логическим полем.

В типе **LogicalStyles** предопределены следующие константы:

- ЛогическийФлаг / LogicalCheckbox;
- ЛогическаяКнопка / LogicalButton.

Значения типа **LogicalStyles** применяются при определении свойства [СтильЛогическогоПоля](#) класса **КлеткаШаблона**.

Константы стилей статической клетки (СтилиСтатическойКлетки / StaticStyles)

Перечислимый тип *СтилиСтатическойКлетки* / *StaticStyles*, определенный в классе *Шаблон* содержит предопределенные константы, описывающие способ использования клетки со статическим текстом.

В типе *StaticStyles* предопределены следующие константы, определяющие содержание клетки шаблона:

- ОбычныйТекст / NormalText - отображается обычный текст;
- СтатическаяКнопка / StaticButton - отображается кнопка;
- СтатическийЗаголовок / StaticHeader - отображается заголовок;
- СтатическаяГиперссылка / StaticHyperlink - отображается гиперссылка.

Константы стилей типа *StaticStyles* применяются при определении свойства [СтильСтатическойКлетки](#) класса *КлеткаШаблона*.

Перечислимый тип *СтилиСкроллера / ScrollerStyles*, определенный в классе *Шаблон / Template* предназначен для скрытия/отображения на фрейме вертикальной и/или горизонтальной полос прокрутки.

Перечислимый тип *СтилиСкроллер* включает следующие константы:

- **ОбаСкроллера / ScrollerBoth** - отображаются одновременно обе полосы;
- **НетСкроллера / ScrollerNone** - не видна ни одна из полос прокрутки;
- **ВертСкроллер / ScrollerVert** - отображается вертикальная полоса прокрутки;
- **ГорСкроллер / ScrollerHorz** - отображается горизонтальная полоса прокрутки.

Константы, определенные в данном типе, используются в свойстве [СтильСкроллера](#) класса *ФреймШаблона*.

Константы типов входа (ТипыВхода / EnterTypes)

В классе **Шаблон** определен тип-перечисление **ТипыВхода / EnterTypes**, содержащий константы для обозначения способа изменения данных в клетке шаблона. Значения типа **EnterTypes** используются в обработчиках событий [ПриВходе/OnEnter](#).

В типе **EnterTypes** предопределены следующие константы:

- **Вход / Enter** - ввод данных стандартным способом;
- **Вставка / Paste** - вставка данных из буфера обмена;
- **Очистка /** - удаление данных.

Константы типов вывода (ТипыВывода / OutputTypes)

В классе **Шаблон** определен перечислимый тип **ТипыВывода / OutputTypes**, содержащий константы для обозначения причины, по которой система запрашивает значение клетки шаблона для вывода.

Значения типа **OutputTypes** используются в обработчиках событий [ПриВыводе](#).

В типе **OutputTypes** предопределены следующие константы:

- **Вывод / Output** - вывод содержимого на экран или при печати;
- **Вычисление / Calculation** - получение значения вычисляемого поля;
- **Копирование / Copy** - копирование в буфер обмена;
- **Экспорт / Export** - вывод данных с помощью команды экспорта.

Константы типов выравнивания (ТипыВыравнивания / AlignmentTypes)

Перечислимый тип **ТипыВыравнивания / AlignmentTypes** из класса **Шаблон / Template** содержит следующие predefined константы, описывающие типы выравнивания полей в шаблонах и столбцах картотек:

- **ВыравниватьВлево / LeftAlign** - выравнивание по левому краю;
- **ВыравниватьВправо / RightAlign** - выравнивание по правому краю;
- **ВыравниватьПоЦентру / CenterAlign** - центрирование.

Значения перечислимого типа **AlignmentTypes** используются при определении свойства **Выравнивание** в классах [СтильШаблона](#), [КлеткаШаблона](#), [СтолбецТаблицы](#), [СтолбецКартотеки](#), [СтолбецПодтаблицы](#).

Константы типов клетки (ТипыКлетки/ CellTypes)

Перечислимый тип **ТипыКлетки/ CellTypes**, определенный в классе **Шаблон / Template** содержит набор констант, описывающих типы полей в шаблонах и картотеках.

В данном типе предопределены следующие константы:


- **СтатическаяКлетка / StaticCell** - статический текст (обычный текст, кнопка, заголовок, гиперссылка);
- **КлеткаПоле / FieldCell** - клетка для ввода и вывода данных;
- **КлеткаВычисляемоеПоле / CalcFieldCell** - клетка, ввод и вывод в которую осуществляется с помощью обработчиков событий.

Константы, определенные в данном типе, используются при определении свойства [ТипКлетки/CellType](#) класса **КлеткаШаблона**.

Константы типов нажатий (ТипыНажатия / ClickTypes)

В классе **Шаблон** определен перечислимый тип **ТипыНажатия / ClickTypes**, содержащий predefined константы для обозначения действий пользователя в шаблонах.

В типе **ClickTypes** predefined следующие константы:

- **ОдиночноеНажатие / SingleClick** - выполнен щелчок мышью;
- **ДвойноеНажатие / DoubleClick** - выполнен двойной щелчок мышью;
- **НажатиеВвода / EnterPressed** - нажата клавиша **Enter**. Для клеток шаблона, работающих как кнопка или флаг, дополнительно может использоваться и клавиша **Пробел**;
- **НажатиеКнопки / ButtonPressed** - нажата кнопка обзора ().

Значения типа **ClickTypes** используются в обработчиках событий ПриНажатии/OnClick в объектах классов:

- [КлеткаШаблона](#)
- [КартотекаШаблона](#)
- [ДеревоКартотеки](#)
- [ПодтаблицаШаблона](#)
- [Картотека](#)
- [ПодтаблицаКартотеки](#)
- [Таблица](#)

Константы типов ориентации (ТипыОриентации / OrientationTypes)

Перечислимый тип **ТипыОриентации / OrientationTypes**, определенный в классе **Шаблон / Template** позволяет задать способ размещения фреймов на шаблоне.

В данном типе определены следующие константы:

- **Горизонтально / Horizontal** - фреймы на шаблоне располагаются друг под другом;
- **Вертикально / Vertical** - фреймы расположены вертикально;
- **СЗакладками / WithTabs** - используется смешанная ориентация, когда фреймы накладываются друг на друга. Инициализация фреймов осуществляется с помощью закладок.

Константы, определенные в данном типе, используются в свойстве [Ориентация / Orientation](#) класса **ФреймШаблона**.

Перечислимый тип **ТипыОтступов / MarginTypes**, определенный в классе **Шаблон / Template** содержит набор констант, описывающих типы отступов.

В данном типе определены следующие константы:

- ОтступСлева / MarginLeft;
- ОтступСверху / MarginTop;
- ОтступСправа / MarginRight;
- ОтступСнизу / MarginBottom.

Константы типов полей (ТипыПоля / FieldTypes)

Перечислимый тип **ТипыПоля / FieldTypes** из класса **Шаблон / Template** содержит следующие predefined константы, описывающие типы полей в шаблонах и картотеках:

- **СтатическийТекст / StaticText** - статический текст
- **ОбщееПоле / CommonField** - поле ввода/вывода
- **СтроковоеПоле / StringField** - поле строки
- **ЧисловоеПоле / NumericField** - поле числа
- **ПолеДаты / DateField** - поле даты
- **ЛогическоеПоле / LogicalField** - логическое поле
- **ПеречислимоеПоле / EnumField** - перечислимое поле
- **СсылочноеПоле / ReferenceField** - ссылочное поле
- **ВычисляемоеПоле / CalcField** - вычисляемое поле

Значения типа **ТипыПоля / FieldTypes** используются при определении свойства **ТипПоля** в классах [КлеткаШаблона](#), [СтолбецТаблицы](#), [СтолбецКартотеки](#), [СтолбецПодтаблицы](#).

Перечислимый тип *ТипыРасположения / AlignTypes*, определенный в классе *Шаблон*, содержит набор констант для задания местоположения:

- картинки относительно текста, который выводится на кнопке (свойство [ПоложениеИзображения](#) класса *Кнопка*);
- встраиваемого окна поверх главного окна (свойство [DockAlign](#) класса *Окно*).

В данном типе определены следующие константы:

- **НетРасположения / AlignNone** - рисунок и текст совмещены, накладываются друг на друга;
- **РасположитьСверху / AlignTop** - картинка расположена сверху, над текстом или встраиваемое окно прижато к верхнему краю главного окна;
- **РасположитьСнизу / AlignBottom** - картинка расположена под текстом или встраиваемое окно прижато к нижнему краю главного окна;
- **РасположитьСлева / AlignLeft** - картинка расположена слева относительно текста или встраиваемое окно прижато к левому краю главного окна;
- **РасположитьСправа / AlignRight** - рисунок на кнопке размещен справа относительно текста или встраиваемое окно прижато к правому краю главного окна;
- **РасположитьПоОкну / AlignClient** - рисунок на кнопке занимает всю площадь, отводимую под кнопку, т.е. его размеры совпадают с размерами кнопки.

Перечислимый тип **ФорматыКлетки / CellFormats**, определенный в классе **Шаблон / Template** содержит набор констант, описывающих форматы полей в шаблонах и картотеках.

В данном типе определены следующие константы:

- ФорматОбщий / CommonFormat;
- ФорматСтроки / StringFormat;
- ФорматЧисла / NumericFormat;
- ФорматДаты / DateFormat;
- ФорматЛогический / LogicalFormat;
- ФорматПеречислимый / EnumFormat;
- ФорматСсылочный / ReferenceFormat.

Константы, определенные в данном типе, используются в свойствах [ФорматКлетки/CellFormat](#) класса **КлеткаШаблона**.

Константы форматов поля даты (ФорматыПоляДаты / DateFormats)

Перечислимый тип **ФорматыПоляДаты / DateFormats**, определенный в классе **Шаблон / Template** содержит набор констант, описывающих форматы полей типа Дата в шаблонах и картотеках.

В данном типе определены следующие константы:

- ТолькоДата / DateOnly;
- ТолькоВремя / TimeOnly;
- ДатаИВремя / DateAndTime.

Константы, определенные в данном типе, используются в свойствах [ФорматПоляДаты/DateFormat](#) класса **КлеткаШаблона**.

Описание

Владелец : [Форма](#);

Owner : [Form](#);

Назначение

Возвращает ссылку на форму, к которой принадлежит шаблон. Поле доступно только на чтение.

Пример

```
var локФорма :Form;
...
proc Pl(Sender :Button);
  локФорма = Template.Owner;
  if (локФорма = nil) then
    локФорма.Close;
  elsif (локФорма <> nil) then
    ...
  end;
end;
```

Описание

Выделение : [ОбластьШаблона](#);

Selection : [TemplateRange](#)

Назначение

Возвращает ссылку (<>nil) на объект из класса [ОбластьШаблона](#), если выделена группа клеток и фокус установлен на фрейме шаблона, а не объекте шаблона, иначе пустую ссылку (nil).

Пример

```
func ВыделенаКлеткаСФормулой(var аКлетка :TemplateCell) :Logical;
var локВыделение :TemplateRange;
var локКлетка :TemplateCell;
локВыделение = Template.Selection;
Result = (локВыделение.BeginColumn = локВыделение.EndColumn) and
(локВыделение.BeginRow = локВыделение.EndRow);
if Result then
    локКлетка = Template.CurrentCell;
    if локКлетка <> nil then
        Result = (локКлетка.CellFormat = Template.StringFormat) and
        (локКлетка.OnOutput = 'ПолеФормула_ПриВыводе');
        if Result then
            аКлетка = локКлетка;
        end;
    else
        Result = False;
    end;
end;
end;
```

Описание

Заморожен : Логическое;
Freezed : Logical;

Назначение

Свойство позволяет получить информацию о наличии на шаблоне в данный момент замороженной области. Если значение поле равно true, то на шаблона имеется [замороженная область](#) и можно [узнать ее размеры](#). После выполнения процедуры [РазморозитьОбласть](#) значение поля становится равным false.

Поле доступно только на чтение.

Пример

```
var FrTempl :TemplateFrame;  
....  
FrTempl=Template.CurrentFrame;  
FrTempl.Freezed = true;
```

Описание

Кадр :Целое;
Frame :Integer;

Назначение

Позволяет узнать и изменить номер текущего кадра текущей секции шаблона. Если полю присваивается значение большее, чем количество кадров в текущей секции, курсор устанавливается на последнюю строку последнего кадра. Важно отметить, что в неповторяющейся секции всегда только один кадр размером во всю секцию.

Пример

```
proc ПриОткрытии(Режим:Logical);  
    -- ставим курсор на нужную позицию  
    self.Template.Frame = 3;  
end;
```

Описание

`КлеткаПоПолю[Имя:Строка] :КлеткаШаблона;`
`CellByField[Name:String] :TemplateCell;`

Аргументы

Имя - имя поля (переменной), содержащейся в клетке.

Назначение

С помощью данного свойства можно получить объект [TemplateCell](#) по имени поля (переменной), содержащейся в клетке. Поиск производится по всем секциям шаблона. Имя задается в качестве индекса. Если в секциях шаблона есть несколько клеток для одноименных полей, будет возвращена первая (с учетом следования секций в шаблоне и нумерации клеток в каждой из них). Если такого поля нет, возвращается **nil**.

Свойство доступно только на чтение.

Пример

```
-- фрагмент кода бланка
try
  КлеткаПоПолю["Итого"].Font.Bold = ИСТИНА;
except
end;
```

Описание

`КоличествоОбъектов` : Целое;
`ObjectsCount` : Integer;

Назначение

Поле хранит количество интерфейсных объектов (то есть объектов классов, производных от класса **ОбъектШаблона**), расположенных на шаблоне.

Поле доступно только на чтение.

Пример

```
-- делаем недоступными все кнопки на шаблоне
for i = 1..Template.ObjectsCount do
  if Template.Object[i].ClassName = "Кнопка" then
    Template.Object[i].Enabled = FALSE;
  end;
end;
```

Описание

КоличествоСекций : Целое;
SectionsCount : Integer;

Назначение

Данное поле позволяет определить количество секций в шаблоне. Поле доступно только на чтение.

Пример

```
-- по нажатию кнопки
proc Press(O:String);
var i:integer;
    -- цикл по всем секциям
    for i=1..Template.SectionsCount do
        Template.Section[i].SortBy("Сумма");
    end;
end;
```


Описание

КоличествоСтилей : Целое;
StylesCount : Integer;

Назначение

Поле содержит количество стилей, определенных для данного шаблона. Поле доступно только на чтение.

Для добавления стиля необходимо воспользоваться функцией [ДобавитьСтиль / AddStyle](#).

Пример

```
-- по нажатию кнопки пытаемся найти
-- и удалить стиль "Специальный"
proc Press1Button(B:Button);
var i:integer;
var n:integer;
  n = self.template.StylesCount;
  for i=1..n do
    if self.template.Style[i].Name = "Специальный" then
      self.template.DeleteStyle(i);
    end;
  end;
end;
```

Описание

```
КонтекстПомощи :Строка;  
HelpContext : String;
```

Назначение

Свойство позволяет определить и задать путь к файлу с текстом помощи, поясняющим назначение текущего шаблона.

Этот текст будет появляться в окне справке по прикладным системам, если в текущем шаблоне нажата клавиша *F1*. Контекст помощи может быть задан также и для [конкретного фрейма](#).

Если поле пустое, то справка к данному шаблону берется из свойства [HelpContext](#) класса **ОбщийДоступ**.

Описание

КорневойФрейм : [ФреймШаблона](#);
RootFrame : [TemplateFrame](#);

Назначение

Свойство возвращает ссылку на корневой фрейм, который всегда существует на шаблоне и имеет предопределенное имя **RootFrame**.

Поле доступно на чтение и запись.

Пример

```
var N :Integer;  
...  
-- определяет количество объектов,  
-- размещенных в корневом фрейме  
N=Template.FrRoot.FramesCount;
```

Описание

НастройкиПечати : Строка;
PrintSettings : String;

Назначение

Свойство позволяет узнать или изменить имя файла настроек печати, имеющего расширение *.cfg. Файлы настроек печати размещаются в папке \Settings\Priters*.cfg.

Пример

```
func АбсолютноеИмяНастроекПечати :String;  
    if ИмеетСобственныеНастройкиПечати then  
        Result = GetCommonFileNameByRecord(Self.НастройкаПечати);  
    end;  
end;  
...  
func ПолеНастройкаПечати_ПриВводе(Cell :TemplateCell; Value :Variant) :Logical;  
    НастройкаПечати = Value;  
    Template.PrintSettings = АбсолютноеИмяНастроекПечати;  
end;
```

Описание

Объект[Индекс:Целое] : [ОбъектШаблона](#);
Object[Index:Integer] : [TemplateObject](#);

Аргументы

Индекс - порядковый номер объекта. Объекты нумеруются от 1 до числа объектов, которое возвращается свойством [КоличествоОбъектов](#).

Назначение

Поле представляет собой массив интерфейсных объектов (то есть объектов классов, производных от класса :[ОбъектШаблона](#)), расположенных на шаблоне.

Поле доступно только на чтение. Объекты добавляются на шаблон и удаляются с него с помощью визуального редактора бланков (на стадии проектирования) либо программно (во время выполнения проекта).

Пример

```
-- делаем недоступными все кнопки на шаблоне
for i = 1..self.Template.ObjectsCount do
  if self.Template.Object[i].ClassName = "Кнопка" then
    self.Template.Object[i].Enabled = FALSE;
  end;
end;
```

Описание

Поле : Строка;
Field : String;

Назначение

Позволяет узнать имя текущего поля ввода (на котором находится курсор), а также установить курсор на поле ввода с указанным именем.

Пример

```
proc ПриОткрытии(Режим:Logical);  
  -- ставим курсор на нужное поле  
  self.Template.Field = "Контрагент";  
end;
```

Описание

Секция[Индекс:Целое] : [СекцияШаблона](#);
Section[Index:Integer] : [TemplateSection](#);

Аргументы

Индекс - номер секции шаблона, который должен изменяться в пределах от 1 до [количества секций](#) на шаблоне.

Назначение

Данное поле позволяет получить доступ к конкретной секции шаблона по ее номеру. Поле доступно только на чтение. Секции добавляются и удаляются с шаблона с помощью визуального редактора на стадии проектирования, либо программно во время выполнения проекта. Секции нумеруются начиная с 1.

Пример

```
-- по нажатию кнопки
proc Press(O:String);
var i:integer;
    -- цикл по всем секциям
    for i=1..Template.SectionsCount do
        Template.Section[i].SortBy( "Сумма" );
    end;
end;
```

Описание

Стиль[Индекс:Целое] : [СтильШаблона\[\]](#);
Style[Index:Integer] : [TemplateStyle\[\]](#);

Аргументы

Индекс - номер стиля, определенный для данного шаблона, который должен изменяться в пределах от 1 до [КоличествоСтилей](#).

Назначение

Поле содержит массив значений типа [СтильШаблона / TemplateStyle](#), позволяя обращаться к стилям, определенным для данного шаблона. Размер массива можно определить с помощью свойства [КоличествоСтилей](#).

Поле доступно только на чтение (элементам массива нельзя присваивать новые значения), однако параметры самих стилей можно менять.

Пример

```
-- по нажатию кнопки пытаемся найти
-- и удалить стиль "Специальный"
proc Press1Button(B:Button);
var i:integer;
var n:integer;
  n = self.template.StylesCount;
  for i=1..n do
    if self.template.Style[i].Name = "Специальный" then
      self.template.DeleteStyle(i);
    end;
  end;
end;
```


Описание

Столбец : Целое;
Column : Integer;

Назначение

Позволяет узнать и изменить номер текущего столбца текущей секции шаблона.

Пример

```
if Шаблон1.Столбец = 3 then  
    Подсказка( "Сальдо" );  
end;
```

Описание

Строка :Целое;
Row :Integer;

Назначение

Позволяет узнать и изменить номер текущей строки шаблона.

Пример

```
if Шаблон1.Строка = 11 then  
    Подсказка("Обобщенные показатели");  
end;
```

Описание

ТекущаяКлетка : [КлеткаШаблона](#);

CurrentCell : [TemplateCell](#);

Назначение

Поле обеспечивает доступ к текущей клетке шаблона, то есть клетке, в которой в данный момент находится курсор.

Поле доступно на чтение и запись, что позволяет программно сделать текущей другую клетку.

Пример

```
proc ButOpenBlank_OnClick(Sender :Button);
var S :String;
if Template.CurrentCell <> nil then
  S = Template.CurrentCell.Text;
  if S <> '' then
    OpenBlank(S);
  end;
end;
end;
```

Описание

ТекущаяСекция : [СекцияШаблона](#);
CurrentSection : [TemplateSection](#);

Назначение

Поле обеспечивает доступ к текущей [секции шаблона](#), то есть секции, в которой в данный момент находится курсор. Кроме того, в данное поле можно записать ссылку на другую секцию, сделав ее текущей (курсор перемещается в первую клетку секции).

Пример

```
if self.Template.CurrentSection = Позиции then
    Button1.Visible = FALSE;
else
    Button1.Visible = TRUE;
end;
```

Описание

ТекущийОбъект : [ОбъектШаблона](#);
CurrentObject : [TemplateObject](#);

Назначение

Поле предназначено для определения текущего объекта шаблона (элемента управления). Текущий объект шаблона обладает фокусом ввода (то есть в него осуществляется ввод с клавиатуры), что отображается с помощью пунктирной рамки по периметру объекта.

Пример

Класс "БланкРедактор", Редактор Пример.Накладная;

```
СуммаПлатежа: Edit;  
proc ПриОткрытии(Режим:Logical);  
    -- ставим курсор на объект "редактор"  
    self.Template.CurrentObject = СуммаПлатежа;  
end;
```

Описание

ТекущийРедактор : [РедакторПоля](#);
CurrentEdit : [FieldEdit](#);

Назначение

Возвращает ссылку на текущий редактор, т.е. на текущее поле бланка, в котором установлен курсор. Причем, если редактирование в поле не начато, то свойство возвращает пустую ссылку (nil).

Поле доступно только на чтение.

Внимание. Для получения ссылки необходимо, чтобы в диалоге ["Свойства кнопки"](#) на странице "Общие" был снят флаг **Закрывает редактор**.

Пример

```
Edit :FieldEdit;  
  
-- событие при входе в текущий редактор  
func FieldS_OnEnter(Cell :TemplateCell;  
  Index :Integer; Action :Template.EnterTypes) :Logical;  
  Edit = Template.CurrentEdit;  
  Result = True;  
end;
```

Описание

ТекущийФрейм : [ФреймШаблона](#);
CurrentFrame : [TemplateFrame](#);

Назначение

Свойство возвращает ссылку на текущий фрейм, т.е. на тот, на котором в данный момент установлен фокус.

Поле доступно на чтение и запись.

Пример

```
var TF :TemplateFrame;  
....  
TF=Template.CurrentFrame;
```

Описание

ФреймПоИмени[Имя :Строка] : [ФреймШаблона](#);
FrameByName[Name :String] : [TemplateFrame](#);

Аргументы

Имя - имя фрейма, по которому определяется ссылка на фрейм.

Назначение

По заданному имени фрейма определяется ссылка на него. Если фрейм не существует, то свойство возвращает nil.

Пример

```
Фрейм1 :TemplateFrame;  
....  
Фрейм1 = Template.FrameByName['Фрейм1'];  
if Фрейм1 <> nil then  
  with Фрейм1.InsertSection(1) do  
    Column[1].Width = 160;  
    ....  
  end;  
end;
```


Описание

Цвет : Целое;
Color : Integer;

Назначение

Позволяет узнать цвет фона шаблона и/или изменить его.

Пример

```
proc OnButtonRed(O:String);  
    template.Color = clRed;  
end;
```

Описание

ЦветнаяПечать :Логическое;
ColourPrinting :Logical;

Назначение

Свойство определяет, какая печать цветная или черно-белая используется при распечатке шаблона. По умолчанию значение свойства равно False, что приводит к печати только черного текста на белом фоне, за исключением кнопок и т.п.

Если значение поле установлено равным True, то шаблон выводится на печать (предварительный просмотр) в цветном изображении в соответствии с цветом его текста, фона и полей.

Поле доступно на чтение и запись.

Пример

```
template.ColourPrinting = False;
```

Описание

```
ВзятьОбластьЗаморозки(var Левый :Число; var Верхний :Число; var Ширина :Число; var  
Высота :Число);  
GetFreezeArea(var Left :Numeric; var Top :Numeric; var Width :Numeric; var Height :Numeric);
```

Аргументы

Левый, Верхний - координаты в мм. левого верхнего угла области замораживания (см. ниже).
Ширина, Высота - ширина и высота области замораживания в мм.

Назначение

Процедура позволяет определить размеры замороженной области, т.е. фиксированной прямоугольной области в окне с шаблоном, которая будет неизменна при навигации по фрейму шаблону и скроллинге.

Пример

```
var aLeft,aWidth : Numeric;  
var aTop,aHeight : Numeric;  
...  
Template.GetFreezeArea(aLeft, aTop, aWidth, aHeight);
```

Описание

```
ЗавершитьМодификацию;  
EndModify;
```

Назначение

Процедура разрешает обновление (перерисовку) шаблона, которое по умолчанию выполняется системой автоматически при любых программных изменениях его полей, но было запрещено предыдущим вызовом [НачатьМодификацию](#).

Следует иметь в виду, что при каждом вызове **НачатьМодификацию** система увеличивает внутренний счетчик блокировки шаблона и уменьшает его при вызове **ЗавершитьМодификацию**. Система возвращается к стандартному режиму автоматической перерисовки бланка только в тот момент, когда счетчик становится равным нулю.

Использовать процедуру имеет смысл после фрагмента кода, выполняющего массивованный пересчет переменных бланка (и записи, в случае бланка-редактора).

Пример

```
-- при нажатии кнопки начинаем длительный пересчет  
proc OnButtonClick(Button1 :Button);  
  -- запрещаем перерисовку бланка  
  BeginModify;  
  -- вызываем пользовательскую функцию  
  CalculateAllFields;  
  -- после всех изменений разрешаем перерисовку  
  EndModify;  
end;
```

Описание

```
ЗавершитьРедактирование(Сохранить:Логическое);  
EndEdit(Save:Logical);
```

Аргументы

Сохранить - логическое значение или выражение, содержащее признак того, следует ли принять сделанные изменения или нет.

Назначение

Закрывает встроенный редактор в клетке, открытый ранее вручную (пользователем) или программно (с помощью [НачатьРедактирование](#)).

В зависимости от значения аргумента, введенное значение сохраняется (TRUE) или нет (FALSE), что эквивалентно, соответственно, нажатию клавиш *Enter* и *Esc*.

Пример

```
func ТМЦ_ПриНаборе(C:TemplateCell; Key:String; Value:String;  
    var NewValue:Пример.Справочник):Logical;  
  
    if Key<>CHR(13) then -- ввод строки еще не завершен  
        return Истина;  
    fi;  
    -- ...  
    Template.EndEdit(Истина);  
    return Ложь;  
end;
```

См. также [Пример обработчика события ПриНаборе в клетке шаблона](#)

Описание

```
ЗаморозитьОбласть({Левый :Число}; {Верхний :Число}; Ширина :Число; Высота :Число);  
FreezeArea({Left :Numeric}; {Top :Numeric}; Width :Numeric; Height :Numeric);
```

Аргументы

Левый, Верхний - необязательные параметры, задающие координаты в мм. левого верхнего угла области замораживания (см. ниже). Если они опущены, то смещение области заморозки считается равным отрицательному отступу шаблона.

Ширина, Высота - ширина и высота области замораживания в мм., которые совместно с параметрами Левый, Верхний определяют размер области замораживания.

Назначение

Процедура замораживает область шаблона, размеры которой задаются параметрами **Левый, Верхний, Ширина, Высота**.

Под *областью замораживания* понимается фиксированная прямоугольная область на шаблоне, которая будет неизменна при навигации по шаблону и скролировании. Область замораживания работает только в режиме исполнения шаблона. Не замороженная область в окне выделяется двойной зеленой пунктирной линией от остальной части окна. Если подвести к этой линии курсор мыши, то область замораживания можно изменять.

Предупреждение. Программа не проверяет заданные размеры области, поэтому может возникнуть ситуация, когда весь экран заморожен даже при максимизированном окне, или область заморозки выходит за границы экрана.

Пример

```
Template.FreezeArea(0,0, 250,70);
```

Описание

```
НачатьМодификацию;  
BeginModify;
```

Назначение

Процедура запрещает обновление (перерисовку) шаблона, которое по умолчанию выполняется системой автоматически при любых программных изменениях его полей. Использовать процедуру имеет смысл перед началом фрагмента кода, выполняющего массивированный пересчет переменных бланка (и записи, в случае бланка-редактора). После выполнения вычислений следует вызвать парную процедуру [ЗавершитьМодификацию](#).

При каждом вызове **НачатьМодификацию** система увеличивает внутренний счетчик блокировки шаблона и уменьшает его при вызове **ЗавершитьМодификацию**. Когда счетчик становится равным нулю, система вновь возвращается к стандартному режиму автоматической перерисовки бланка в ответ на модификацию любого поля.

Пример

```
-- при нажатии кнопки начинаем длительный пересчет  
proc OnButtonClick(Button1 :Button);  
  -- запрещаем перерисовку бланка  
  BeginModify;  
  -- вызываем пользовательскую функцию  
  CalculateAllFields;  
  EndModify;  
end;
```

Описание

НачатьРедактирование ;
BeginEdit ;

Назначение

Если текущая клетка (выделенная текстовым курсором) имеет тип "поле ввода", то вызов данной процедуры переводит ее в режим редактирования.

Пример

```
-- при смене текущей клетки в шаблоне
proc OnMove;
  -- проверяем, связана ли клетка с переменной "Сумма"
  if Template.CurrentCell.Contents = "Сумма" then
    -- и переводим ее в состояние редактирования
    Template.BeginEdit;
  end;
end;
```


Описание

Обновить;
Update;

Назначение

Процедура предназначена для принудительной перерисовки шаблона, что может быть полезно в процессе выполнения продолжительных операций внутри обработчиков событий, так как шаблон штатно перерисовывается только по завершении обработчика.

Процедура устарела и оставлена лишь для совместимости.

Пример

```
proc Press(O:String);
var i:integer;
  for i=1..Documents do
    -- ... обработка документов, влияющая на значения полей шаблона
    Template.Update;
  end;
end;
```

Описание

```
РазморозитьОбласть ;  
FreezeArea ;
```

Назначение

Процедура размораживает область шаблона, которая была зафиксирована процедурой [ЗаморозитьОбласть / FreezeArea](#). В этом случае значение поля [Заморожен / Freezed](#) становится равным False.

Пример

```
Template.UnFreezeArea ;
```

Описание

```
УдалитьОбъект(Индекс:Целое);  
DeleteObject(Index:Integer);
```

Аргументы

Индекс - порядковый номер объекта, который требуется удалить. Значение должно лежать в пределах от 1 до числа объектов, которое возвращается свойством [КоличествоОбъектов](#).

Назначение

Функция предназначена для программного удаления объекта шаблона. В результате выполнения функции все объекты, бывшие ранее в списке после удаленного, сдвигаются к началу списка, то есть их индексы уменьшаются на 1.

Пример

```
-- по нажатию кнопки  
-- удаляем последний объект из списка объектов шаблона  
proc PressDelButton(B:Button);  
    template.DeleteObject[template.ObjectsCount];  
end;
```

Описание

```
УдалитьСекцию(Индекс:Целое);  
DeleteSection(Index:Integer);
```

Аргументы

Индекс - позиция, куда требуется удалить. Значение должно лежать в пределах от 1 до [количества секций](#) на шаблоне.

Назначение

Функция предназначена для программного удаления секции шаблона. После выполнения функции все нижележащие секции сдвигаются вверх, а их индексы уменьшаются на 1.

Пример

```
-- по нажатию кнопки  
-- удаляем нижнюю секцию  
proc PressDelButton(B:Button);  
    template.DeleteSection[template.SectionsCount];  
end;
```

Описание

```
УдалитьСтиль(Номер: Целое);  
DeleteStyle (Index: Integer);
```

Аргументы

Номер - задает индекс удаляемого стиля. Значение должно лежать в пределах от 1 до [количества имеющихся стилей](#).

Назначение

Процедура предназначена для удаления указанного по индексу стиля. Все элементы шаблона, оформленные удаленным стилем, теряют свое оформление.

Пример

```
-- по нажатию кнопки пытаемся найти  
-- и удалить стиль "Специальный"  
proc Press1Button(B:Button);  
var i:integer;  
var n:integer;  
  n = self.template.StylesCount;  
  for i=1..n do  
    if self.template.Style[i].Name = "Специальный" then  
      self.template.DeleteStyle(i);  
    end;  
  end;  
end;
```

Описание

ПередЗакрытием : Строка;
BeforeClose : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит перед закрытием шаблона.

Обработчик

func **BeforeClose**(ModalResult:Integer): Logical;

Параметр **ModalResult** содержит код возврата в случае, если шаблон был открыт в модальном режиме. В противном случае **ModalResult** всегда равно коду cmCancel. Если функция возвращает TRUE – шаблон закрывается, если функция возвращает FALSE (или возникает ошибка) – шаблон не закрывается.

Внимание! Будьте осторожны при реализации этой функции. В случае ошибки в логике работы данного обработчика, единственный способ закрыть сессию – снять программу по Ctrl+Alt+Del!

Пример

```
func ПередЗакрытием(КодВозврата:Целое): Логическое;  
  if КодВозврата = cmOk:  
    if Query.Current.IsGroup then  
      message("Эта операция не может производиться с группой!");  
      return Ложь;  
    else  
      return Истина;  
    fi;  
  else  
    return Истина;  
  fi;  
end;
```

Описание

ПередИзменением : Строка;
BeforeModify : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит перед началом изменения записи, редактируемой бланком-редактором.

Обработчик

```
func BeforeModify(Action : Cardfile.ModifyActions; TheRecord :Record; var AskConfirm :Logical) :Logical;
```

Параметр **Action** детализирует, какое именно действие производится над записью (удаление, или восстановление). Ссылка на запись передается через параметр **TheRecord**. Параметр **AskConfirm** позволяет указать системе, следует ли запрашивать подтверждения у пользователя. Если данный параметр при выходе из обработчика равен TRUE, то пользователю выдается запрос, а в случае FALSE – нет. По умолчанию (при входе в обработчик) **AskConfirm** равно TRUE.

Если обработчик возвращает TRUE, то тем самым разрешается выполнение действия и будут сгенерированы все последующие события. Если обработчик **ПередИзменением** вернет FALSE, выполнение действия над записями картотеки прекращается.

При входе в обработчик Result равен FALSE (действия запрещены).

Пример

```
func Card_BeforeModify(Action :Cardfile.ModifyActions; TheRecord :Record; var AskConfirm :Logical) :Logical;  
-- все действия выполнять без подтверждений  
AskConfirm = FALSE;  
return TRUE;
```

Описание

```
ПриЗакрытии : Строка;  
OnClose : String;
```

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при закрытии шаблона.

Событие происходит до того, как хранимые [переменные](#) бланка сохраняются в файл.

Обработчик

```
proc OnClose(BlankOnly:Logical);
```

Параметр **BlankOnly** содержит признак того, закрывается ли конкретный бланк или все приложение (вместе с бланком). Если закрывается только бланк, параметр

BlankOnly равен TRUE. Если закрывается весь проект (осуществляется выход из сессии), то значение равно FALSE.

Пример

```
proc ПриЗакрытии(ТолькоБланк:Logical);  
  -- если пользователь закрывает данный бланк...  
  if ТолькоБланк then  
    CloseApp; -- он закрывает и всю программу  
  fi;  
end;
```


Описание

ПриЗаписи : Строка;
OnPost : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при попытке сохранить документ, а именно: после события [OnVerify](#) (если таковое было).

Обработчик

func **OnPost** : Logical;

Событие позволяет контролировать запись новых значений переменных бланка-редактора в документ. Если обработчик возвращает TRUE, документ автоматически сохраняется, иначе – нет (однако разработчик может сохранить документ, вызвав соответствующие методы вручную).

Внимание! При вызове метода [Post](#) объекта **Запись** событие **ПриЗаписи** не происходит!

Пример

```
-- процедура-обработчик события ПриЗаписи
func шаблон_ПриЗаписи :Logical;
  Result = ОбработкаДокументаПриЗаписи;
  if Result then
    Result = ПроверитьКодПриЗаписиДокумента;
    if Result then
      Result = Inherited шаблон_ПриЗаписи;
    end;
  end;
end;

func ПриЗаписи:Логическое;
  Hint ("Документ сохранен");
  return TRUE;
end;
```

Описание

ПриОбновленииОтчета : Строка;
OnUpdateReport : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое позволяет скорректировать свойства клетки, переданной в качестве параметра, или свойства ее колонки.

Обработчик

прос **OnUpdateReport**

(Action : [ReportForm.UpdateActions](#);
Cell : [TemplateCell](#);
IndicatorIndex :Integer;
SumKind : [Report.SumKinds](#);
DebCre : [Report.SumDebCre](#);
Part : [Report.IndicatorParts](#));

Параметры:

Action - действие обновления, задается одной из констант [перечислимого типа UpdateActions](#);

Cell - ссылка на текущую [клетку шаблона](#);

IndicatorIndex - порядковый номер показателя в списке показателей, перечисленных в поле **Имя** на [странице "Показатели"](#) диалога "Внутренние отчеты";

SumKind - [константа](#), задающая тип показателя (начальный остаток, оборот или конечный остаток);

DebCre - [константа](#), определяющая формат вывода заданного показателя (выводится дебет, кредит или свернутое значение показателя);

Part - [константа](#), показывающая, какая часть показателя выводится: его значение или единица измерения.

Событие вызывается для клеток в таблицах формируемого шаблона, а при завершении формирования таблицы и для колонок. Событие позволяет прикладному программисту при формировании шаблона отчета скорректировать свойства клетки, переданной в качестве параметра, или свойства ее колонки.

В момент вызова события внутренняя структура отчета синхронизирована на нужную таблицу, строку, столбец. Вызов любых процедур, приводящих к изменению этого положения, запрещен (приведет к возбуждению исключения)!!!

Событие срабатывает:

- при первоначальном формировании шаблона;
- при интерактивном раскрытии иерархии;
- при интерактивном уточнении.

Событие вызывается на тех клетках с данными отчета, которые соответствуют тем показателям, для которых включена соответствующая опция, которую можно задать только программно, используя свойство [Опция/Option](#).

Предупреждение. Событие не вызывается на заголовках столбцов и строк, а также на строках, к которым нет программного доступа.

Описание

ПриОткрытии : Строка;
OnOpen : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит каждый раз при открытии шаблона.

Обработчик

proc **OnOpen**(OpenMode:Logical);

Параметр **OpenMode** содержит логическое значение TRUE, если бланк открывается пользователем или программно в процессе работы с проектом, и FALSE, если бланк загружается вместе с проектом в момент открытия сессии. То есть, бланк был открыт пользователем в прошлой сессии, затем сессия была закрыта, и программа сохранила состояние бланка в специальном долговременном кэше. Теперь, в момент запуска новой сессии состояние бланка восстанавливается из кэша (иными словами, открытие бланка инициировано самой системой, а не пользователем или кодом прикладного проекта).

Если шаблон принадлежит форме картотеки, то в поведении события есть некоторая особенность. Форма картотеки имеет свойство Запрос, определяющее множество записей, которые должны отображаться в картотеке. В момент возникновения события запрос еще не открыт. Он откроется только после завершения работы обработчика, но до появления картотеки на экране. Это дает возможность настраивать параметры запроса (например, [LoadingFields](#)) и картотеки (например, [Filter](#)) которые вступят в силу сразу после открытия запроса. Альтернативный вариант – открыть запрос самостоятельно ([Query.Select](#)) прямо из обработчика – тогда по его завершении ничего делаться не будет.

Пример

```
proc ПриОткрытии(Режим:Logical);  
  -- устанавливаем значения по умолчанию  
  Валюта = "USD";  
end;
```

См. также пример работы с [измененной записью](#).

Описание

ПриОтмене : Строка;
OnCancel : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит при попытке отменить сделанные в документе изменения и вернуть его в первоначальное состояние.

Обработчик

func **OnCancel** : Logical;

Событие позволяет контролировать "откат" к старому состоянию документа. Если обработчик возвращает TRUE, документ возвращается в исходное состояние; если функция возвращает FALSE – ничего не происходит.

При вызове метода [Cancel](#) объекта **Document** событие **ПриОтмене** не происходит. Чтобы его сгенерировать, пользуйтесь методом [EditorCancel](#) объекта **ФормаБланка**.

Методы [EditorPost](#) и [EditorCancel](#) нельзя вызывать из обработчиков событий [ПриПроверке](#), [ПриЗаписи](#) или [ПриОтмене](#) во избежание зацикливания.

Пример

```
НовыйДокумент:Logical;  
-- ...  
func ПриОтмене : Логическое;  
-- если в новых документах какие-то поля проставляются  
-- программой по умолчанию, то при "сбросе" такого документа  
-- нужно повторить инициализацию  
if НовыйДокумент then  
    УстановитьЗначенияПоУмолчанию; -- прикладная процедура  
end;  
return TRUE;  
end;
```

Описание

ПриПеремещении : Строка;
OnMove : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при перемещении текстового курсора в шаблоне (смене текущей клетки или объекта шаблона).

Обработчик

проц **OnMove**;

Событие позволяет управлять состоянием и значением полей бланка в зависимости от контекста ввода.

Пример

```
-- процедура выводит остатки ТМЦ, выделенного в позициях накладной,  
-- в клетку, связанную с переменной ОстаткиТекущегоТМЦ  
proc OnMove;  
  if Template.CurrentSection = Позиции then  
    ОстаткиТекущегоТМЦ = РасчетОстатковПоПозиции(Template.Frame);  
  end;  
end;
```

Описание

ПриПодготовкеОтчета : Строка;
OnPrepareReport : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит при интерактивном раскрытии иерархии и уточнении.

Обработчик

func **OnPrepareReport**(Action : [ReportForm.ExpandActions](#); Report : [Report](#)) :Logical;

Параметры:

Action - перечислимый параметр, указывающий, в какой ситуации вызвано событие: при интерактивном раскрытии иерархии (в том числе и полном раскрытии, см. ниже) или уточнении;
Report - ссылка на отчет, который модифицируется.

Событие используется для программной модификации отчета и вызывается при интерактивном раскрытии иерархии и уточнении. Внутри обработчика считается, что внутренняя структура отчета синхронизирована на текущей строке секции шаблона.

Внимание! В процедуре, которая будет вызываться на это событие, нельзя выходить за пределы данной группы или уточнения. Эту же процедуру можно использовать и для первоначального заполнения отчета.

Если обработчик возвращает False, то в обработчике события были произведены все необходимые действия.

Установка Result = True разрешает стандартную обработку, которая зависит от текущего состояния группы|уточнения. Например, если группа закрыта, то, достраивает интерактивную группу|уточнение (если надо), а затем форматирует. Если группа открыта - закрывает и форматирует.

Особенности работы обработчика при полном раскрытии иерархии

Полное раскрытие иерархии используется в параметрических отчетах (для заполнения пользовательских показателей). Для возникновения этого события следует задать **Action** = FullExpandHierarchy. При этом возможны два варианта:

- Report.CurRow = 0 (итоговая строка), то раскрывается вся таблица (как бы от корня);
- Report.CurRow > 0, то раскрывается группа.

Для отображения построенной иерархии по всей таблице используется свойство [ГруппаОткрыта](#), как и для группы, но при этом нужно, чтобы Report.CurRow = 0. Отображение перестроенной иерархии можно не делать в обработчике, вернув в качестве результата True, тогда это произойдет по умолчанию.

Пример

См. в теме [Пример работы с уточняющими отчетами](#).

Описание

ПриПроверке : Строка;
OnVerify : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит каждый раз, когда пользователь закончил ввод в поля шаблона и дал команду на его сохранение.

В момент события измененный документ еще не сохранен. Программа может проверить введенные значения на допустимость и, в случае необходимости, откорректировать их.

Обработчик

func **OnVerify** : Logical;

Обработчик может выполнить одно из трех действий:

1. отменить сохранение документа; в этом случае обработчик должен вернуть значение FALSE;
2. разрешить сохранение документа в существующем виде; в этом случае обработчик должен вернуть значение TRUE;
3. откорректировать введенные пользователем значения полей, сделав их допустимыми, и разрешить сохранение документа; в этом случае обработчик также вернуть TRUE.

Данный обработчик вызывается до обработчика события [ПриЗаписи/OnPost](#).

Внимание! При вызове метода [Post](#) объекта **Document** событие **ПриПроверке** не происходит! Если необходимо программно записать документ с проверкой – вызывайте метод [EditorPost](#) объекта **ФормаБланка**.

Методы [EditorPost](#) и [EditorCancel](#) нельзя вызывать из обработчиков событий [ПриПроверке](#), [ПриЗаписи](#) или [ПриОтмене](#) во избежание зацикливания.

Пример

```
func ПриПроверке:Logical;  
  if КорИмя<>' ' then  
    return Истина;  
  else  
    message('Укажите краткое название юр.лица!');  
    return Ложь;  
  fi;  
end;
```

Описание

ПриСчитывании : Строка;
OnRead : String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной процедуры-обработчика события, которое происходит каждый раз при считывании полей шаблона из полей записи (при этом шаблон должен использоваться в бланке-редакторе).

Обработчик

проц **OnRead**;

Данный обработчик вызывается после обработчика [ПриОткрытии](#).

Пример

```
проц ПриСчитывании;  
  Заголовок=if(isGroup,'Группа сделок','Сделка');  
end;
```


Описание

ВставитьОбъект(Индекс:Целое; Класс:Класс ОбъектШаблона) :[ОбъектШаблона](#);
InsertObject(Index:Integer; Class:Class TemplateObject) :[TemplateObject](#);

Аргументы

Индекс - позиция, куда требуется вставить новый объект. Значение должно лежать в пределах от 1 до числа объектов на шаблоне.

Класс - название класса, производного от класса [ОбъектШаблона](#).

Назначение

Функция предназначена для программного создания и добавления на шаблон нового объекта. Объект вставляется перед объектом, индекс которого передан в качестве параметра **Индекс**. После выполнения функции новый объект получит тот же индекс, вытесняя старые объекты к концу служебного списка объектов, поддерживаемого шаблоном.

Пример

```
-- по нажатию кнопки
-- вставляем поле ввода в начало списка объектов шаблона
proc PressMoreButton(B:Button);
var e1:Edit;
    e1 = template.InsertObject(1,Edit);
end;
```

Описание

ВставитьСекцию(Индекс:Целое) : [СекцияШаблона](#);
InsertSection(Index:Integer) : [TemplateSection](#);

Аргументы

Индекс - позиция, куда требуется вставить новую секцию. Значение должно лежать в пределах от 1 до [количества секций](#) на шаблоне.

Назначение

Функция предназначена для программного создания и добавления на шаблон новой секции. Секция добавляется перед секцией, индекс которой передан в качестве параметра. После выполнения функции новая секция получит тот же индекс, а индексы всех нижележащих секций увеличатся на 1.

Пример

```
-- по нажатию кнопки
-- вставляем секцию в начало шаблона
proc PressMoreButton(B:Button);
var t:TemplateSection;
    t = self.template.InsertSection[1];
end;
```

Описание

ДобавитьОбъект(Класс:Класс ОбъектШаблона) :[ОбъектШаблона](#);
AddObject(Class:Class TemplateObject) :[TemplateObject](#);

Аргументы

Класс - имя класса добавляемого объекта. Это должен быть один из классов, производных от класса [ОбъектШаблона](#).

Назначение

Функция предназначена для программного создания и добавления на шаблон нового объекта. Объект добавляется в конец списка объектов шаблона, доступного через свойство Объект. По умолчанию объект появляется в верхнем левом углу шаблона.

Пример

```
-- по нажатию кнопки
-- добавляем еще одну кнопку
proc PressMoreButton(B:Button);
var b2:Button;
    b2 = template.AddObject(Button);
    b2.Left = 100;
    b2.Top = 50;
end;
```

Описание

ДобавитьСекцию : [СекцияШаблона](#);
AddSection : [TemplateSection](#);

Назначение

Функция предназначена для программного создания и добавления на шаблон новой секции. Секция добавляется под последней из существующих секций.

Пример

```
НазваниеГруппыТМЦ:Строка;  
  
-- по нажатию кнопки  
-- добавляем секцию и устанавливаем ее параметры  
proc PressMoreButton(B:Button);  
var t:TemplateSection;  
  t = self.template.AddSection;  
  t.ColumnsCount = 3;  
  ЗаполнитьСекциюГруппойТМЦ(НазваниеГруппыТМЦ,t);  
end;  
  
proc ЗаполнитьСекциюГруппойТМЦ(Название:Строка; T:СекцияШаблона)  
  -- ...  
end;
```

Описание

ДобавитьСтиль : [СтильШаблона](#);

AddStyle : [TemplateStyle](#);

Назначение

Функция предназначена для создания нового [стиля](#) и добавления его в число стилей, определенных для текущего шаблона.

Пример

```
-- по нажатию кнопки
-- создаем новый стиль и устанавливаем его параметры
proc Press2Button(B:Button);
var T: TemplateStyle;
  T = self.template.AddStyle;
  T.Name = "Специальный";
  T.Bevel = TRUE;
end;
```

Описание

ПриКоманде :Строка;
OnCommand :String;

Назначение

Данное свойство содержит и позволяет изменить название прикладной функции-обработчика события, которое происходит перед стандартной обработкой очередной команды.

Обработчик

func **OnCommand**(Command :Command) :Logical;

Параметры:

Command - указатель на объект класса [Команда](#).

Пользовательский обработчик вызывается перед стандартной обработкой заданной команды из активного бланка. Если обработчик возвращает TRUE, то команда была обработана стандартным образом, если - False, то команда будет обработана, но стандартный механизм вызван не будет.

Пример

```
func шаблон_ПриКоманде(aCommand :Command) :Logical;  
  if Command.FullName = 'Kernel.Help.Help' then  
    Result = inherited шаблон_ПриКоманде(Command);  
  else  
    if (ИнтерфейсПредставления <> nil) then  
      -- Result = ...  
    else  
      Result = inherited шаблон_ПриКоманде(Command);  
    end;  
  end;  
end;  
end;
```

Обслуживающие (сервисные) классы

Класс Объект

Все классы, наследующие свойства и методы от класса [Объект](#), в Справочной системе сгруппированы по своему назначению на группы.

К группе сервисных классов, входящих в группу "Прочие", относятся следующие:









- [Администрирование / Administration](#)
- [Хранилище / Storage](#)
- [ХранилищеПараметров / ParamStorage](#)
- [ДвоичныйОбъект / BinaryObject](#)
- [АвтоОбъект / AutoObject](#)
- [ТекстовыйФайл / TextFile](#)
- [DBFФайл / DBFFile](#)
- [Окно / Window](#)
- [Принтер / Printer](#)
- [List](#)
- [СписокСтрок / StringList](#)
- [Шрифт / Font](#)
- [Таймер / Timer](#)
- [Изображение / Image](#)
- [OLEДокумент / OLEDocument](#)
- [ЭлементТаблицы / GridItem](#)

Класс *List* является производным от класса [Объект](#) и создается с помощью конструктора `Create`. Класс *List* хранит список элементов, значения которых могут принадлежать разным типам. Его можно использовать как аналог массива, в некоторых случаях он более эффективен.

Класс *List* практически во всем аналогичен классу [StringList](#), который является его наследником, за исключением двух моментов:

- элементами класса *List* являются варианты;
- класса *List* можно создавать с помощью конструктора `Create`.

Непосредственно в классе *List* вводятся следующие свойства и методы:

-  [Функция Создать / Create](#)
-  [Поле Количество / Count](#)
-  [Поле Пункты / Items](#)
-  [Процедура Добавить / Add](#)
-  [Процедура Вставить / Insert](#)
-  [Процедура Удалить / Delete](#)
-  [Процедура Очистить / Clear](#)
-  [Процедура Упорядочить / Sort](#)

Описание

Количество :Целое;
Count :Integer;

Назначение

Данное поле позволяет узнать количество позиций в списке.

Поле доступно только на чтение.

Пример

```
var List1 :List;  
  
Proc P1(Sender :Button);  
  var I :Integer;  
  List1:=List.Create;  
  
  for I = 1..10 do  
    List1.Add("Товар"+Стр(I));  
  end;  
  I = List1.Count;  
  Message("Количество позиций в списке = " + Стр(I));  
end;
```

Описание

Пункты[Индекс :Целое]:Вариант;
Items[Index :Integer]:Variant;

Назначение

По заданному индексу позволяет узнать или изменить значение произвольного типа в списке. Поле доступно на чтение и запись.

Если указанный индекс выходит за пределы списка, генерируется ошибка.

Пример

```
var List1 :List;  
  
Proc P1(Sender :Button);  
  var i :Integer;  
  List1=List.Create;  
  -- ....  
  for i=1.. List1.Count do  
    Message(List1.Items[i]);  
  end;  
end;
```

Описание

Вставить(Индекс :Целое; Значение :Вариант);
Insert(Index :Integer; Value :Variant);

Аргументы

Индекс - индекс, на основании которого определяется позиция вставляемого значение;
Значение - вставляемое значение.

Назначение

Вставляет указанное значение в позицию, индекс которой на единицу больше, заданного в первом параметре. Значение индекса должно лежать в пределах от 1 до количества, уже имеющихся индексов, плюс один.

Пример

```
var List1 :List;  
  
Proc P1(Sender :Button);  
    List1=List.Create;  
    List1.Insert(1,"Товар1");  
    Message("Количество позиций в списке = " +  
        Ctr(List1.Count));  
    -- ...  
end;
```

Описание

```
Добавить(Значение :Вариант);  
AddInsert(Value :Variant);
```

Аргументы

Значение - вставляемое значение.

Назначение

Добавляет указанное значение в конец списка.

Пример

```
var List1 :List;  
  
Proc P1(Sender :Button);  
  var I :Integer;  
  List1:=List.Create;  
  
  for I = 1..10 do  
    List1.Add("Товар"+Стр(I));  
    Message("Значение = " + List1.Items[I] +  
      " (номер позиции "+ Стр(I)+"");  
  end;  
  -- ....  
end;
```

Описание

Очистить;
Clear;

Назначение

Очищает список, т.е. удаляет все позиции списка.

Пример

```
var List1 :List;  
  
Proc P1(Sender :Button);  
  var I :Integer;  
  List1=List.Create;  
  -- ....  
  List1.Clear;  
end;
```

Описание

```
Удалить(Индекс :Целое);  
Delete(Index :Integer);
```

Аргументы

Индекс - номер удаляемого элемента списка.

Назначение

Удаляет указанный элемент из списка.

Пример

```
var List1 :List;  
var i,n :Integer;  
-- ....  
n = List1.Count;  
if i<=n then  
    List1.Delete(i);  
end;
```

Описание

Упорядочить;
Sort;

Назначение

Производит упорядочивание значений списка.

Если значения списка сравнить нельзя, то выдается сообщение об ошибке.

Пример

```
var List1 :List;  
--...  
List1.Sort;
```

Описание

Создать : [List](#);

Create : [List](#);

Назначение

Создает новый объект класса [List](#).

Пример

```
var List1 :List;

Proc P1(Sender :Button);
  List1=List.Create;
  List1.Insert(1,"Товар1");
  Message("Количество позиций в списке = " +
    Стр(List1.Count));
  -- ...
end;
```


Класс *OLEДокумент* / *OLEDocument* является производным от класса *ДвоичныйОбъект* и позволяет работать с документами, отвечающими спецификации OLE (например, документ Microsoft Word, книга Microsoft Excel, видеоклип и т.д.). Возможно хранение таких документов в БД. Для этого в MTL-описании нужно добавить поле типа OleDocument.

Класс *OLEДокумент* наследует все свойства родительских классов [Объект/Object](#) и [ДвоичныйОбъект](#).

В классе *OLEДокумент* вводится свойство

 [Создать / Create](#)

Класс Объект : [ДвоичныйОбъект](#) : [OLEДокумент](#)

Функция Создать / Create

Описание

Создать : [OLEДокумент](#) ;

Create : [OLEDocument](#) ;

Назначение

Создает новый объект класса [OLEДокумент](#). Как правило, переменная типа **OLEДокумент** должна быть связана с объектом шаблона [OLEКонтейнер](#) и обрабатываться через него.

Класс *DBFФайл* (*DBFFile*) является производным от класса *Объект*. С помощью объектов *DBFФайл* реализуется работа с файлами формата DBF. *DBFФайл* наследует от родительского класса следующие свойства, описанные в [разделе Объект](#).

Новые свойства класса, которые вводятся в классе *DBFФайл*, а также наследуемые переопределенные свойства, описываются в следующих темах:

-  [Функция Создать / Create](#)
-  [Функция ИмяФайла / FileName](#)
-  [Функция КоличествоПолей / FieldsCount](#)
-  [Поле ИмяПоля / FieldName](#)
-  [Поле / Field](#)
-  [Поле ВКонец / EOF](#)
-  [Поле ВНачале / BOF](#)
-  [Процедура Добавить / Add](#)
-  [Процедура Удалить / Delete](#)
-  [Процедура Первый / First](#)
-  [Процедура Последний / Last](#)
-  [Процедура Следующий / Next](#)
-  [Процедура Предыдущий / Previous](#)
-  [Поле Количество / Count](#)

Описание

ВКонец: Логическое;
EOF: Logical;

Назначение

Возвращает значение TRUE после попытки переместить указатель DBF-файла за конец файла (указатель при этом остается на последней записи), и FALSE в противном случае. Поле доступно только на чтение.

Если файл пустой, свойства **BOF** и **EOF** равны TRUE.

Пример

См. также [Пример работы с DBF-файлами](#).

Описание

ВНачале: Логическое;
BOF: Logical;

Назначение

Возвращает значение TRUE после попытки переместить указатель DBF-файла за начало файла (указатель при этом остается на первой записи), и FALSE в противном случае. Поле доступно только на чтение.

Если файл пустой, свойства **BOF** и **EOF** равны TRUE.

Пример

```
var dbf : DBFFile;  
-- создаем пустой файл  
dbf = DBFFile.Create("Text.dbf", "dbfs.txt", fmCreate, TRUE);  
-- записей нет, поэтому O.BOF равно TRUE  
if dbf.BOF then  
    Message("Это сообщение всегда выводится");  
fi;
```

Описание

```
ИмяПоля[Номер :Целое] :Строка;  
fieldName[Number :Integer] :String;
```

Аргументы

Номер - номер поля, для которого требуется узнать имя.

Назначение

Возвращает имя заданного по номеру поля. Индекс должен лежать в пределах от 1 до общего числа полей. Поле доступно только на чтение.

Пример

См. также [Пример работы с DBF-файлами.](#)

Описание

Количество: Целое;

Count: Integer;

Назначение

Возвращает общее количество записей в файле, исключая те, что помечены к удалению. Поле доступно только на чтение.

Пример

```
var dbf : DBFFile;  
dbf = DBFFile.Create("Text.dbf","",0,TRUE);  
Message("В файле " + Str(dbf.Count) + " записей" );
```

Описание

Поле [Имя :Строка] :Вариант;
Field [Name:String] :Variant;

Аргументы

Имя - имя поля, к которому требуется обратиться.

Назначение

Свойство позволяет прочитать и изменить значение указанного по имени поля текущей записи DBF-файла.

Пример

```
var dbf : DBFFile;  
-- создаем новый файл  
dbf = DBFFile.Create("Text.dbf", "dbfs.txt", fmCreate, TRUE);  
dbf.Add; -- добавляем 1 запись  
dbf.Field["STR"]="текст"; -- записываем в нее в поле "STR" строку "текст"  
dbf.Add; -- добавляем еще 1 запись  
dbf.Field["NUM2"]=784.43; -- записываем в нее в поле "NUM2" число 784.43  
dbf = nil; -- закрываем файл
```


Описание

Добавить;
Add;

Назначение

Добавляет новую запись в конец DBF-файла. Если непосредственно после вызова этой процедуры в поля новой записи не записываются какие-либо значения, запись удаляется.

Пример

```
var dbf : DBFFile;  
dbf = DBFFile.Create("Text.dbf", "dbfs.txt", fmCreate, TRUE);  
dbf.Add; -- добавляем одну запись  
-- записываем в её поля требуемые значения  
dbf.Field["STR"] = "текст";  
dbf.Field["NUM2"] = 784.43;  
dbf = nil; -- закрываем файл
```

Описание

Первый;
First;

Назначение

Перемещает указатель DBF-файла на первую запись.

Пример

```
var t : Integer;
var dbf : DBFFile;
var D[] : String;
D = [ "раз", "два", "три" ];

-- записываем файл
dbf = DBFFile.Create("Text.dbf", "dbfs.txt", fmCreate, TRUE);
for t=1..3 do
  dbf.Add;
  dbf.Field["STR1"]=D[t];
end;
dbf.First; -- возвращаемся в начало
for t=1..3 do
  trace(dbf.Field["STR1"]); -- проверяем, что записали
  dbf.Next; -- переходим на следующую запись
end;
```

Описание

Последний;
Last;

Назначение

Перемещает указатель DBF-файла на последнюю запись.

Пример

```
var dbf : DBFFile;  
dbf = DBFFile.Create("Text.dbf","",0,TRUE);  
dbf.Last;  
Message(dbf.Field["STR1"]);  
-- выводим поле из последней записи
```

Описание

Предыдущий;
Previous;

Назначение

Перемещает указатель DBF-файла на предыдущую запись (если она не первая). Удаленные записи (которые физически еще находятся в файле) минуются.

Пример

```
var dbf : DBFFile;  
dbf = DBFFile.Create("Text.dbf", "", 0, TRUE);  
dbf.Last;  
dbf.Previous;  
-- выводим поле из предпоследней записи  
Message(dbf.Field["STR1"]);
```

Описание

Следующий ;
Next ;

Назначение

Перемещает указатель DBF-файла на следующую запись (если она не последняя). Удаленные записи (которые физически еще находятся в файле) минуются.

Пример

См. также [Пример работы с DBF-файлами.](#)

Описание

Удалить;
Delete;

Назначение

Помечает текущую запись удаленной. Физически запись остается в DBF-файле.

Пример

```
var dbf : DBFFile;  
-- открываем существующий файл  
dbf = DBFFile.Create("Text.dbf", "", 0, TRUE);  
--текущей становится первая запись  
dbf.Delete; -- удаляем первую запись
```

Описание

ИмяФайла :Строка;
FileName :String;

Назначение

Возвращает название DBF-файла.

Пример

```
proc ProcessDbf(const input :DBFFile)

  Hint("Обрабатываем файл:"+ input.FileName);
  -- ...
  -- непосредственно, обработка
  Hint("");
end;
```

Класс Объект : [DBFФайл](#)

Функция КоличествоПолей / FieldsCount

Описание

```
КоличествоПолей :Целое;  
FieldsCount :Integer;
```

Назначение

Возвращает количество полей в DBF-файле.

Пример

См. также [Пример работы с DBF-файлами.](#)

Описание

Создать(Имя:Строка; Описание:Строка; Режим:РежимыФайлов; WinКодировка:Логическое):DBFФайл;
Create(Name:String; Desc:String; Mode:FileModes; WinCharCode:Logical):DBFFile;

Аргументы

Имя - имя файла (может включать полный путь);

Описание - имя файла с описанием полей DBF-файла; требуется только при создании файла, а при открытии уже существующего файла должно быть пустой строкой;

Режим - одна из предопределенных [констант](#), задающая режим работы с файлом; если Режим равен fmCreate - создается новый файл;

ДосКодировка - логическое значение задает кодировку текста: Windows, если FALSE, и кодировка DOS, если TRUE.

Назначение

Создает или открывает DBF-файл с указанным именем в каталоге текущего проекта или в другом каталоге, если через аргумент **Имя** передается полный путь.

Файл с описанием структуры определяет, какие поля должен содержать создаваемый DBF-файл. Формат файла со структурой следующий: на каждой строке описывается одно поле DBF-файла в виде

ИмяПоля:ИмяТипа(Размер[.Точность])

где ИмяПоля ограничено 10 символами и может содержать только символы латинского алфавита;

ИмяТипа может быть STRING, REAL, INTEGER, BOOLEAN, DATE; **Размер** задает ширину поля в байтах или количество знакомест, отведенных под число; **Точность** определяет количество знаков после запятой для действительных чисел (REAL).

Режим работы с файлом определяется третьим аргументом, который может быть равен rfСоздать/fmCreate (см. [константы режимов работы с файлами](#)), для того чтобы создать новый файл, или нулю во всех остальных случаях. Если файл с указанным именем уже существует и при этом используется режим fmCreate, старый файл обнуляется.

Параметр **ДосКодировка** задает кодовую таблицу текста. Кодировка Windows (CP-1251) используется, когда данный параметр равен FALSE; кодировка ДОС (CP-866) используется, когда данный параметр равен TRUE.

Для того чтобы закрыть открытый файл необходимо присвоить **nil** переменной файла.

Пример






```
proc ReadPrice(s:String);
var dbffil:DBFFile;

-- открыть dbf с Dos-кодировкой
dbffil=DBFFile.Create(s,"",0,true);
try
-- обработка файла
finally
--
end;
end;
```

См. также [Пример работы с DBF-файлами](#).

Класс *AutoObject* (АвтоОбъект) является производным от класса *Объект*. С помощью объектов *AutoObject* можно управлять серверами автоматизации (OLE Automation) и интегрировать прикладные проекты Студии с другим современным ПО, поддерживающим основанные на COM технологии межпрограммного взаимодействия. *AutoObject* наследует от родительского класса следующие свойства, описанные в [разделе Объект](#):

Непосредственно в классе *AutoObject* вводятся следующие свойства и методы: а также наследуемые переопределенные свойства:

-  [Создать / Create](#)
-  [Открыть / Open](#)
-  [СоздатьУдаленно / CreateRemote](#)
-  [ИмяОбъекта / ObjectName](#)
-  [ИмяСервера / ServerName](#)

Поскольку *АвтоОбъект* может быть создан для сервера автоматизации с произвольным набором свойств, то вероятно ситуация, когда одно из свойств сервера перекрывается одноименным свойством объекта *АвтоОбъект*. В таком случае обращение к свойству сервера напрямую через разыменовывание (например, "Авто.Открыть") будет приводить к вызову свойства *АвтоОбъект*. Чтобы разрешить данную проблему, в класс *АвтоОбъект* введен специальный метод **Invoke (Вызвать)** :

```
Вызвать (ИмяСвойства: Строка [, ...]) [:Вариант];  
Invoke (PropertyName: String [, ...]) [:Variant];
```

Данный метод принимает в качестве первого параметра название свойства (метода) сервера, а далее передается переменное число параметров, определяемых прототипом самого вызываемого свойства. Если вызывается метод-процедура, то возвращаемого значения нет. В случае же функции, метод **Invoke** возвращает результат работы функции.

С помощью **Invoke** можно вызывать абсолютно все свойства сервера (в том числе и одноименные **Invoke**).

Описание

ИмяОбъекта:Строка;
ObjectName:String;

Назначение

Возвращает имя класса OLE-объекта, зарегистрированного в системе Windows.

Пример

```
proc P1(O:AutoObject);  
  try  
    message(O.ObjectName);  
  except  
    message('Не удастся опознать объект');  
  end;  
end;
```

Описание

ИмяСервера:Строка;
ServerName:String;

Назначение

Возвращает имя сервера, на котором выполняется OLE-объект. Если объект существует локально, возвращается пустая строка.

Пример

```
proc P1(O:AutoObject);  
  try  
    if length(O.ServerName) = 0 then  
      message( "Локальный сервер" );  
    else  
      message("Сервер " + O.ServerName);  
    fi;  
  except  
    message("Не удастся опознать сервер");  
  end;  
end;
```

Описание

```
Открыть(Имя:Строка; Новый:Логическое):АвтоОбъект;  
Open(Name:String;New:Logical):AutoObject;
```

Аргументы

Имя - текстовая строка с именем OLE-объекта;

Новый - логическое значение: TRUE предписывает создать новый экземпляр объекта, если такого еще нет, FALSE означает, что нужно лишь проверить, существует ли такой OLE-объект и если да, вернуть ссылку на него.

Назначение

По имени класса OLE-объекта, зарегистрированного в системе Windows, проверяет, существует ли экземпляр указанного класса. Если объект уже существует, функция возвращает ссылку на него, и далее с объектом можно работать точно также, как и с объектом, созданным с помощью [Create](#).

В противном случае, если объектов указанного класса пока нет, то функция может создать новый объект при условии, что второй параметр равен TRUE.

Пример

```
ExcelObj:AutoObject;  
proc Fl;  
  try  
    -- проверить, запущен ли Excel и если да, то обратиться к нему  
    -- в противном случае - запустить Excel  
    if ExcelObj = nil:  
      ExcelObj = AutoObject.Open("Excel.Application", ИСТИНА);  
    fi;  
    ExcelObj.Visible=Истина;  
  except  
    message('Не удастся запустить MS Excel');  
  end;  
end;
```

Описание

Создать(Имя:Строка):АвтоОбъект;
Create(Name:String):AutoObject;

Аргументы

Имя - зарегистрированное в системе Windows имя OLE-объекта.

Назначение

По имени класса OLE-объекта, зарегистрированного в системе Windows, создает его экземпляр и возвращает указатель на вновь созданный объект. Если объект невозможно создать, генерируется исключительная ситуация (ошибка).

Пример

```
ExcelObj:AutoObject;  
proc F1;  
  try  
    ExcelObj = AutoObject.Create("Excel.Sheet");  
    ExcelObj.Application.Visible=Истина;  
  except  
    message('Не удастся запустить MS Excel');  
  end;  
end;
```

Описание

СоздатьУдаленно(Имя:Строка; Комп:Строка):АвтоОбъект;
CreateRemote(Name:String; Comp:String):AutoObject;

Аргументы

Имя - текстовая строка с именем OLE-объекта;
Комп - сетевое имя сервера.

Назначение

По имени класса OLE-объекта, зарегистрированного в системе Windows, создает на удаленном компьютере экземпляр указанного класса. Если объект невозможно создать, возвращает nil.








Пример

```
proc Pl(O:AutoObject);
  var ExcelObj:AutoObject;
  ...
  try
    ExcelObj = AutoObject.CreateRemote("Excel.Sheet",O.ServerName);
    if ExcelObj <> nil:
      message('Объект создан');
    else
      message('Не удастся создать объект');
    fi;
  except
    message('....');
  end;
  ....
end;
```

Класс *Администрирование* / *Administration* является наследником класса *Объект* и сохраняет все его свойства и методы, описанные [в разделе Объект](#).

Класс *Администрирование* предназначен для работы с [информационными базами](#). В частности, объекты этого класса позволяют создать [резервную копию базы](#), выполнить [сборку мусора](#) и [репликацию](#) информационной базы с одного компьютера на другой. Для создания объектов класса *Администрирование* используется функция **Создать** данного класса.

Непосредственно в данном классе определены следующие методы и перечислимые типы:

-  [Функция Создать / Create](#)
-  [Процедура РезервноеКопирование / InfobaseBackup](#)
-  [Процедура СборкаМусора / GarbageCollection](#)
-  [Процедура ЭкспортРепликаций / ReplicationExport](#)
-  [Процедура ИмпортРепликаций / ReplicationImport](#)
-  [Функция ПередачаРепликаций / ReplicationTransfer](#)
-  [Перечислимый тип РежимУдаления / DeleteMode](#)

Перечислимый тип *РежимУдаления / DeleteMode*, определенный в классе *Администрирование*, предназначен для задания режимов удаления переданных с сервера файлов репликации.

В данном типе определены следующие константы:

- **pyОставлять / dmLeave** - оставлять переданные файлы;
- **pyУдалять / dmDelete** - удалять переданные файлы без использования корзины;
- **pyУдалятьВКорзину / dmDeleteToRecycle** - удалять переданные в корзину файлы.

Константы, определенные в данном типе, используются в методе [ПередачаРепликаций](#) класса *Администрирование*.

Описание

```
ИмпортРепликаций(ИнфБаза :Строка; Схема :Строка; Файл :Строка; {Удалить :Логическое};  
{РазрешитьИмпортСтарыхПакетов :Логическое}; {ПоказатьМастер :Логическое});
```

```
ReplicationImport (InfoBase :String; Schema :String; File :String; {Delete :Logical};  
{AllowImportOldPackets :Logical}; {ShowWizard :Logical});
```

Аргументы

ИнфБаза - имя информационной базы;

Схема - название схемы репликации;

Файл - имя файла пакета репликаций. Если полный путь не указан, по умолчанию он берется относительно каталога для входящих пакетов репликаций (обычно

Удалить - необязательный логический параметр, по умолчанию значение равно True. В этом случае после успешной загрузки пакет удаляется;

ПоказатьМастер - необязательный логический параметр, если он равен True, будет запущен мастер импорта пакетов репликации. По умолчанию значение аргумента ПоказатьМастер = True.

Назначение

Процедура выполняет импорт репликаций для одного или всех участников с использованием мастера импорта пакетов репликации или без него.

Пример

```
inclass public  
  stored var Infobase :String;  
  stored var Schema :String := 'Cxemal';  
  stored var File :String;  
  stored var ShowWizard1 :Logical;  
inobject private  
  var Admin :Administration;  
  
  proc ButImport_OnClick(Sender :Button);  
    Admin.ReplicationImport(Infobase, Schema,  
      File, True, ShowWizard1);  
  end;
```

Здесь Admin - объект, созданный функцией [Создать](#).

Описание

РезервноеКопирование (ИнфБаза :Строка; Файл :Строка; {ВключатьЛоги :Логическое};
{СредствамиСУБД :Логическое}; {ПоказатьМастер :Логическое});

InfobaseBackup (InfoBase :String; File :String; {IncludeLogs :Logical};
{NativeBackup :Logical}; {ShowWizard :Logical});

Аргументы

ИнфБаза - имя информационной базы;

Файл - имя файла, в котором хранится резервная копия с данными информационной базы. Обычно эти данные хранятся в папке

ВключатьЛоги - необязательный логический параметр, если он равен True, то включаются файлы логирования измененных записей *.log;

СредствамиСУБД - необязательный логический параметр, при значении равном True резервная копия будет восстанавливаться в ИБ на той же самой СУБД. Включение данного флага позволяет существенно ускорить процесс резервного копирования;

ПоказатьМастер - необязательный логический параметр, если он равен True, то запускается мастер резервного копирования.

Назначение

Процедура позволяет получить резервную копию информационной базы с использованием [мастера](#) создания резервной копии информационной базы или без него.

Пример

```
inobject private
  var Admin :Administration;

proc ButExport_OnClick(Sender :Button);
  Admin.InfobaseBackup(BaseInfo.Name,
    '*,*',True,True,True);
end;
```

Здесь Admin - объект, созданный функцией [Создать](#).

Описание

```
СборкаМусора (ИнфБаза :Строка {; ПоказатьМастер :Логическое});  
GarbageCollection(InfoBase :String {; ShowWizard :Logical});
```

Аргументы

ИнфБаза - имя информационной базы;

ПоказатьМастер - необязательный логический параметр, если он равен True, будет запущен [мастер сборки мусора](#). По умолчанию значение аргумента ПоказатьМастер = True.

Назначение

Выполняет операцию [сборки мусора](#) для заданной информационной базы.

Пример

```
inclass public  
  stored var Infobase :String;  
  stored var ShowWizard1 :Logical;  
inobject private  
  var Admin :Administration;  
  
  proc ButGarbage_OnClick(Sender :Button);  
    Admin.GarbageCollection(Infobase, ShowWizard1);  
  end;
```

Здесь Admin - объект, созданный функцией [Создать](#).

Описание

ЭкспортРепликаций (ИнфБаза :Строка; Схема :Строка; {Участник :Строка}; {Пакет :Целое};
{ПоказатьМастер :Логическое});

ReplicationExport (InfoBase :String; Schema :String; {Point :String}; {Packet :Integer};
{ShowWizard :Logical});

Аргументы

ИнфБаза - имя информационной базы;

Схема - название схемы репликации;

Участник - имя участника схемы репликации. Если указана пустая строка, будут выгружены пакеты для всех участников. По умолчанию значение аргумента Участник=""; (пустая строка);

Пакет - номер пакета, начиная с которого будут выгружены данные репликации. Значение по умолчанию = -1. Номер пакета может принимать значения:

-1 - будут выгружены все модификации, произошедшие после последней выгрузки;

0 - будут выгружены все нужные данные для синхронизации (принудительная синхронизация всех документов).

ПоказатьМастер - необязательный логический параметр, если он равен True, будет запущен мастер экспорта пакетов репликации. По умолчанию значение аргумента ПоказатьМастер = True.

Назначение

Процедура предназначена для экспорта репликаций для одного или всех участников с использованием мастера экспорта пакетов репликации или без него.

Пример

```
inclass public
  stored var Infobase :String;
  stored var Schema :String := 'Схема1';
  stored var Point :String;
  stored var ShowWizard1 :Logical;
inobject private
  var Admin :Administration;

  proc ButExport_OnClick(Sender :Button);
    Admin.ReplicationExport(Infobase,
      Schema, Point, , ShowWizard1);
  end;
```

Здесь Admin - объект, созданный функцией [Создать](#).

Описание

```
ПередачаРепликаций(Файлы :Строка[]; ЦелевойСервер :Администрирование;  
{РежимУдаления :Администрирование.РежимУдаления}; {ПоказатьМастер :Логическое}) :Логическое;
```

```
ReplicationTransfer (Files :String[]; DestinationServer :Administration;  
{DeleteMode :DeleteMode}; {ShowWizard :Logical}) :Logical;
```

Аргументы

Файлы - массив имен файлов репликации, подлежащих отправке, возможно использование масок файлов;

ЦелевойСервер - сервер, на который требуется отправить файлы репликации;

РежимУдаления - режим удаления успешно отправленных исходных файлов, который определяется заданной константой перечислимого типа [РежимУдаления](#). По умолчанию задано значение dmDeleteToRecycle;

ПоказатьМастер - необязательный логический параметр, по умолчанию он равен True, т.е. передача пакетов репликации выполняется с использованием мастера. Если параметр ПоказатьМастер = False, то мастер не используется.

Назначение

Функция предназначена для отправки файлов репликации с сервера, представленного объектом **Administration**, у которого вызывается данная функция.

Функция возвращает значение True, если передача файлов состоялась и - False, если не было передано ни одного файла.

Описание

```
Создать({Сервер :Строка}; {Пользователь :Строка}; {Пароль :Строка}) :Администрирование;  
Create({Server :String}; {User :String}; {Password :String}) :Administration;
```

Аргументы

Сервер - имя сервера;
Пользователь - имя пользователя;
Пароль - пароль.

Назначение

Функция создает новый объект класса **Администрирование**, причем, все аргументы функции являются необязательными..

Основное назначение функции обеспечить реализацию конструктора объектов в базовом классе **Администрирование**.








Пример

```
inclass public  
  stored var Server :String;  
  stored var User :String;  
  stored var Password :String;  
inobject private  
  var Admin :Administration;  
  
proc ButCreate_OnClick(Sender :Button);  
  Server=SessionInfo.ServerName;  
  User= SessionInfo.UserName;  
  Admin=Administration.Create(Server, User, Password);  
end;
```

Класс *ДвоичныйОбъект / BinaryObject* является наследником класса *Объект* и сохраняет все его свойства и методы, описанные в [разделе Объект](#).

Класс *ДвоичныйОбъект*, в свою очередь, является родительским для производного от него класса [Изображение](#).

Непосредственно в классе *ДвоичныйОбъект / BinaryObject* определены следующие свойства:

-  [Функция Создать / Create](#)
-  [Процедура ЗагрузитьИзФайла / LoadFromFile](#)
-  [Процедура ЗаписатьВФайл / SaveToFile](#)
-  [Процедура ВставитьИзБуфера / PasteFromClipboard](#)
-  [Процедура КопироватьВБуфер / CopyToClipboard](#)
-  [Процедура Очистить / Clear](#)
-  [Поле Пустой / Empty](#)

Описание

Пустой :Логическое;

Empty :Logical;

Назначение

Поле позволяет узнать, содержит ли переменная класса [ДвоичныйОбъект](#) конкретное значение. Объект может быть пустым, если в него еще ничего не загружалось (из файла или из буфера обмена), либо он был очищен с помощью метода [Очистить](#).

Поле доступно только на чтение.

Пример

```
proc ВставитьИзБуфера(Объект:ДвоичныйОбъект);
var x:integer;
  if NOT Объект.Пустой then
    x = ВопрДаОтказ("Объект непустой. "+
      "Старое содержимое будет уничтожено. Продолжить?");
    if x = кмдОтказ then
      return;
    end;
  end;
  Объект.ВставитьИзБуфера;
end;
```

Описание

```
ВставитьИзБуфера;  
PasteFromClipboard;
```

Назначение

Загружает в двоичный объект содержимое буфера обмена Windows. Предыдущее содержимое объекта (если он был не пуст) удаляется.

Пример

```
proc НажатаКнопкаИзБуфера(Объект:Строка);  
  var Об: ДвоичныйОбъект;  
  S: String;  
  Об = ДвоичныйОбъект.Создать;  
  Об.ВставитьИзБуфера;  
  if ChooseFile(S, 'Выберите имя для выходного файла') = cmOk then  
    Об.ЗаписатьВФайл(S);  
  end;  
end;
```

Описание

```
ЗагрузитьИзФайла (Имя:Строка) ;  
LoadFromFile (Name:String) ;
```

Аргументы

Имя - название файла для загрузки.

Назначение

Загружает в двоичный объект содержимое указанного файла. Предыдущее содержимое объекта (если он был не пуст) удаляется.

Пример

```
-- Так можно скопировать файл любой природы  
-- если он, конечно, не слишком длинный.  
proc КопированиеФайла (ИмяВхФайла:Строка; ИмяВыхФайла :Строка);  
  var Об : ДвоичныйОбъект;  
  Об = ДвоичныйОбъект.Создать;  
  Об.ЗагрузитьИзФайла (ИмяВхФайла) ;  
  Об.ЗаписатьВФайл (ИмяВыхФайла) ;  
end;
```

Описание

```
ЗаписатьВФайл(Имя:Строка);  
SaveToFile(Name:String);
```

Аргументы

Имя - название файла для записи.

Назначение

Сохраняет содержимое двоичного объекта в указанный файл.

Пример

```
-- Так можно скопировать файл любой природы  
-- если он, конечно, не слишком длинный.  
proc КопированиеФайла (ИмяВхФайла:Строка; ИмяВыхФайла :Строка);  
  var Об : ДвоичныйОбъект;  
  Об = ДвоичныйОбъект.Создать;  
  Об.ЗагрузитьИзФайла(ИмяВхФайла);  
  Об.ЗаписатьВФайл(ИмяВыхФайла);  
end;
```

Описание

КопироватьВБуфер
CopyToClipboard;

Назначение

Записывает содержимое двоичного объекта в буфер обмена Windows.

Пример

```
proc НажатаКнопкаВБуфер(Объект:Строка);  
  var ОБ: ДвоичныйОбъект;  
  S: String;  
  if ChooseFile(S, 'Выберите имя файла для загрузки') = cmOk then  
    ОБ = ДвоичныйОбъект.Создать;  
    ОБ.ЗагрузитьИзФайла(S);  
    ОБ.КопироватьВБуфер;  
  end;  
end;
```

Описание

```
Очистить ;  
Clear ;
```

Назначение

Освобождает внутренние структуры данных, связанные с хранимым в объекте значением.

Описание

Создать : [ДвоичныйОбъект](#);
Create : [BinaryObject](#);

Назначение

Создает новый объект класса [ДвоичныйОбъект](#).

Пример

```
-- Так можно скопировать файл любой природы
-- если он, конечно, не слишком длинный.
proc КопированиеФайла (ИмяВхФайла:Строка; ИмяВыхФайла :Строка);
  var Об :ДвоичныйОбъект;
  Об = ДвоичныйОбъект.Создать;
  Об.ЗагрузитьИзФайла (ИмяВхФайла);
  Об.ЗаписатьВФайл (ИмяВыхФайла);
end;
```

Класс *Изображение* / *Image* является производным от класса *ДвоичныйОбъект* и позволяет работать с изображениями (картинками) в формате: bmp, jpg, ico, wmf или emf. Класс *Изображение* наследует все свойства родительских классов [Объект/Object](#) и [ДвоичныйОбъект](#).

С помощью методов [LoadFromFile](#), [SaveToFile](#), [CopyToClipboard](#) и [PasteFromClipboard](#) родительского класса *ДвоичныйОбъект* можно соответственно загрузить изображение (картинку) в одном из уже перечисленных форматов из файла, сохранить содержимое картинки в файл, копировать картинку в буфер обмена и из буфера обмена.

Непосредственно в классе *Изображение* определены следующие свойства:

 [Создать / Create](#)

 [Печать / Print](#)

Процедура Печать / Print

Описание

```
Печать[[[Принтер:Строка] [; Заголовок:Строка] ]];  
Print [[[Printer:String] [; Title:String] ]];
```

Аргументы

Принтер - название одного из принтеров, установленных в системе Windows. Если параметр опущен, используется принтер по умолчанию.

Заголовок - заголовок, под которым будет выполняться задание на печать. Если параметр опущен, используется пустая строка.

Назначение

Выводит на печать изображение.

Пример

```
proc ButPrintClick(Sender :String);  
  if Image1 <> nil then  
    -- печать на принтер по умолчанию  
    Image1.Print( , "Изображение" );  
  end;  
end;
```

Описание

Создать : [Изображение](#) ;
Create : [Image](#) ;

Назначение

Создает новый объект класса [Изображение](#);

Пример





```
Class "Картинка";

  Image1: Image;
  Picture1 :Picture;
  -- объект шаблона, связанный с Image1
  PicFile :String;

  proc ButFileClick(Sender :String);
    var S :String;
    S = PicFile;
    if ВыборФайла(S, "Выберите файл",
      "BMP-файл|*.bmp|JPG-файл|*.jpg") = cmOk then
      if Image1 = nil then
        Image1 = Image.Create;
      end;
      Image1.ЗагрузитьИзФайла(S);
      PicFile = S;
    end;
  end
```

Класс *Окно* (*Window*), производный от класса *Объект*, предназначен для выполнения базовых операций по управлению оконным интерфейсом программы. Он наследует от родительского класса [Объект](#) все свойства и методы.

Непосредственно в классе *Окно* определены следующие свойства:

-  [Поле Надпись / Caption](#)
-  [Поле Слева / Left](#)
-  [Поле Сверху / Top](#)
-  [Поле Ширина / Width](#)
-  [Поле Высота / Height](#)
-  [Поле Развернутое / Maximized](#)
-  [Поле Активное / Active](#)
-  [Поле Модальное / Modal](#)
-  [Поле Закрывается / Closing](#)
-  [Поле DockAlign](#)
-  [Поле ТекущийОбъект / CurrentObject](#)
-  [Процедура ЗагрузитьИконку / LoadIcon](#)
-  [Процедура Наверх / GoTop](#)

Для идентификации областей окна используются константы перечислимые типы:

-  [СтилиОкон / WindowStyles](#)
-  [ОбъектыФормы / FormObjects](#)

Внимание! Создать объект класса *Окно* напрямую нельзя. Окна создаются только внутри системы и используются как свойства других объектов (например, экранных форм), причем такие свойства (с типом *Окно*) имеют атрибут "только чтение". Вместе с тем, программа, написанная на языке ТБ.Скрипт, может получить доступ к объекту *Окно*, являющемуся свойством другого объекта, и изменять свойства самого окна.

Для идентификации 4 областей, которые могут присутствовать на форме (т.е. шаблона, табличной части картотеки, дерева картотеки и подтаблицы картотеки) в классе **Окно** введен перечислимый тип **ОбъектыФормы / FormObjects** со следующими константами:

- Шаблон / Template;
- Картотека / Cardfile;
- ДеревоКартотеки / CardfileTree;
- ПодтаблицаКартотеки / CardfileSubtable.

Следует различать объекты формы и объекты шаблона.

Бланки в системе могут открываться в разных режимах, например, как дочернее окно, модальное окно, плавающее окно или встраиваемое окно.

Для идентификации видов окон, используемых программой, в классе **Окно** введен перечислимый тип **СтилиОкон / WindowStyles** со следующими константами:

- АвтоОпределение / AutoDetect;
- ДочернееОкно / ChildWindow;
- МодальноеОкно / ModalWindow;
- ПлавающееОкно / PopupWindow;
- ВстраиваемоеОкно / DockedWindow.

Константа AutoDetect используется для автоматического определения типа окна, в тех случаях, когда в функциях [ОткрытьБланк / OpenBlank](#), [ОткрытьБланкРедактор / OpenBlankEditor](#), [ОткрытьКартотеку / OpenCardfile](#) и в процедуре [ПостроитьОтчет / BuildReport](#) опущен параметр **WindowStyle**, по умолчанию он считается равным AutoDetect. Т.е. используется значение заданное в шаблоне, если же приложение уже работает в модальном режиме - то новый бланк тоже откроется модально.

Явное задание константы AutoDetect позволяет исключить лишние проверки в коде бланка, т.к. в зависимости от условий программа сама определяет способ открытия окна.

Описание

```
DockAlign : Шаблон.ТипыРасположения;  
DockAlign : Template.AlignTypes;
```

Назначение

Функция используется в тех случаях, когда требуется открывать *встраиваемые окна*, т.е. плавающие окна, прижатые к заранее заданной границе главного окна. Функция используется в обработчике **ПриОткрытии** (см. пример). В зависимости от значения, хранимого в поле, т.е. от выбранной константы [перечислимого типа](#), встраиваемое окно будет прижато:

- к верхнему краю главного окна (константа РасположитьСверху / AlignTop);
- к нижнему краю главного окна (константа РасположитьСнизу / AlignBottom);
- к левому краю главного окна (константа РасположитьСлева / AlignLeft);
- к правому краю главного окна (константа РасположитьСправа / AlignRight).

Если эта функция не используется, то по умолчанию *встраиваемое окно открывается как отдельное окно*, если в диалоге ["Свойства шаблона"](#) на странице "Окно" в поле **Стиль** задано значение "Встраиваемое окно". Этот же результат можно получить программно, если открыть форму с помощью команды типа:

```
MyBlank.ShowForm(Window.DockedWindow);
```

Пример

```
Window.DockAlign = Template.AlignLeft;  
-- Если этот код написать в обработчике ПриОткрытии, то форма шаблона  
-- открывается во встраиваемом окне, прижатом к левому краю главного окна
```

Описание

Активное: Логическое;

Active: Logical;

Назначение

Позволяет узнать, является ли соответствующее окно в данный момент времени активным (Active = True) или нет (Active = False). Напоминаем, что активным считается окно, на котором установлен фокус. В каждый момент времени только одно окно может быть активным.

Поле доступно только на чтение.

Пример

```
-- код внутри бланка
var W : Window;
W = Window; -- получить окно бланка
if W.Active then
  Message("Окно является активным.");
fi;
```

Описание

Высота: Целое;
Height: Integer;

Назначение

Позволяет получить и установить высоту окна в пикселях.

Пример

```
-- код внутри бланка
var W : Window;
W = Window; -- получить окно бланка
W.Width = 600; -- установить ширину
W.Height = 300; -- установить высоту
```


Описание

Закрывается: Логическое;

Closing: Logical;

Назначение

Свойство доступно только на чтение и возвращает значение True, если окно находится в процессе закрытия. Свойство может использоваться в бланке-редакторе в обработчиках событий [ПриЗаписи](#) или [ПриОтмене](#) для того, чтобы узнать, вызван ли обработчик по причине закрытия окна или по какой-либо другой причине.

Однако, надо учитывать, что даже, если в одном из этих обработчиков событий свойство Window.Closing имеет значение True, то нет гарантии, что окно будет закрыто, т.к. процесс закрытия может быть прерван позднее.

Описание

Модальное: Логическое;
Modal: Logical;

Назначение

Позволяет узнать, является ли окно в данный момент модальным (Modal = True) или нет (Modal = False).

Поле доступно только на чтение.

Пример

```
-- код  бланка
var W : Window;
W = Window;
if not W.Modal:
  -- если окно бланка не модальное
  W.GoTop;
  -- переместить его на передний план
fi;
```

Описание

Надпись: Строка;
Caption: String;

Назначение

Позволяет получить и установить заголовок окна. Это та же самая строка, которая изначально задается разработчиком в cod-файле в заголовке класса/бланка, т.е. после ключевого слова **Класс** или **Бланк** в кавычках.

Строка может содержать два варианта надписи - один для вывода в заголовке окна, а второй - сокращенный, для вывода на закладке окна. Два этих варианта разделяются символом вертикальная черта '|'. Если этого символа нет, то вся строка считается единым заголовком, который будет выводиться как в верхней части окна, так и на его закладке.

Пример

```
-- код внутри бланка
var W : Window;
W = Window; -- получить окно бланка
Message("Окно имеет заголовок:" + W.Caption);
-- устанавливаем новый заголовок и текст для закладки
W.Caption = "Новое название|Новое";
```

Описание

Развернутое: Логическое;

Maximized: Logical;

Назначение

Позволяет узнать, максимизировано ли в данный момент окно, а также переключать его состояние из максимизированного на обычное и обратно.

Если окно открыто в модальном режиме, оно максимизируется на весь экран. Если окно открыто в немодальном режиме и, следовательно, является дочерним MDI-окном главного окна Студии, оно разворачивается внутри рабочей области Студии. При переключении данного свойства у дочернего MDI-окна оно синхронно меняется и у всех остальных немодальных окон.

Пример

```
var x:Шаблон1;  
x = Шаблон1.СоздатьВидимым;  
x.Window.Maximized = true;
```

Описание

Сверху: Целое;

Top: Integer;

Назначение

Позволяет получить и установить положение верхней границы окна в пикселях. Значение берется относительно верхней границы экрана или главного окна программы в зависимости от режима (модальный/немодальный) отображения окна.

Пример

```
-- код внутри бланка
var W : Window;
W = Window; -- получить окно бланка
W.Left = 50; -- установить левую границу
W.Top = 50; -- установить верхнюю границу
```

Описание

Слева: Целое;
Left: Integer;

Назначение

Позволяет получить и установить положение левой границы окна в пикселях. Значение берется относительно левой границы экрана или главного окна программы в зависимости от режима (модальный/немодальный) отображения окна.

Пример

```
-- код внутри бланка
var W : Window;
W = Window; -- получить окно бланка
W.Left = 50; -- установить левую границу
W.Top = 50; -- установить верхнюю границу
```

Описание

ТекущийОбъект : [Окно.ОбъектыФормы](#);
CurrentObject : [Window.FormObjects](#);

Назначение

Позволяет узнать и установить текущий (активный) объект формы, заданный с помощью одной из констант [Окно.ОбъектыФормы](#). В результате установки данного свойства фокус ввода на форме перемещается в соответствующую область.

Следует различать объекты формы и объекты шаблона.

Пример

```
proc FocusSubtable;  
    CurrentObject = Window.CardfileSubtable;  
end;
```

Описание

Ширина: Целое;
Width: Integer;

Назначение

Позволяет получить и установить ширину окна в пикселях.

Пример

```
-- код внутри бланка
var W : Window;
W = Window; -- получить окно бланка
W.width = 600; -- установить ширину
W.Height = 300; -- установить высоту
```


Описание

```
ЗагрузитьИконку(ИмяФайла :Строка);  
LoadIcon(IconFileName :String);
```

Аргументы

ИмяФайла - имя файла с иконкой и путь к нему.

Назначение

Процедура загружает указанную иконку и меняет на нее значок окна.

Если файл иконки лежит в [каталоге](#) проекта, то доступ к нему может осуществляться с использованием макроса %projects%.

Если передаваемая строка не содержит пути, файл ищется в текущем каталоге, который определяется способом запуска программы. Рекомендуется всегда указывать имя и путь.

Если имя файла пустое, восстанавливается стандартная иконка.

Пример

```
-- код внутри бланка  
proc BlankOnOpen(Create :Logical);  
  try  
    LoadIcon(  
      SystemPath(spProject, ClassProject)  
      +  
      "document.ico");  
  except  
  end;  
end;
```

Описание

Наверх;
GoTop;

Назначение

Перемещает окно на передний план.

Пример

```
-- код внутри бланка
var W : Window;
W = Window;
  -- получить окно бланка
if not W.Modal :
  -- если оно не модальное
  W.GoTop;
  -- переместить его на передний план
fi;
```








Класс Принтер / Printer

Класс Объект : [Сервисные классы](#)

Класс *Принтер* предоставляет расширенный интерфейс к механизму печати Windows. С его помощью можно узнать список установленных в системе принтеров, сменить текущий принтер или режим печати. Также он позволяет группировать печать множества документов в рамках одного задания на печать, что значительно ускоряет процесс печати.

Для получения ссылки на объект данного класса используется свойство [Принтер](#) класса [Консоль](#)

В классе *Принтер* определены следующие свойства:

-  Поле [КоличествоПринтеров / PrintersCount](#)
-  Поле [Принтер / Printer](#)
-  Поле [ТекущийПринтер / CurrentPrinter](#)
-  Поле [АльбомныйРежим / LandscapeMode](#)
-  Поле [Печатает / Printing](#)
-  Проц [НачатьПечать / BeginPrint](#)
-  Проц [ЗавершитьПечать / EndPrint](#)
-  Проц [ОтменитьПечать / AbortPrint](#)

поле КоличествоПринтеров / PrintersCount

Класс Объект : [Принтер](#)

Описание

```
поле КоличествоПринтеров :Целое;  
field PrintersCount :Integer;
```

Назначение

Позволяет узнать количество принтеров, установленных в Windows. Для получения имени принтера используется индексированное свойство [Принтер](#).

Свойство доступно только для чтения.

Пример

[Установка текущего принтера](#)

поле Принтер / Printer

Класс Объект : [Принтер](#)

Описание

```
поле Принтер [Индекс :Целое] :Строка;  
field Printer [Index :Integer] :String;
```

Назначение

Позволяет получить имя любого принтера, установленного в Windows. Для получения количества установленных принтеров используется свойство [КоличествоПринтеров](#).

Имя принтера, полученное с помощью данного свойства, в дальнейшем может использоваться для установки текущего принтера, с помощью свойства [ТекущийПринтер](#), или в процедуре [Печать](#).

Свойство доступно только для чтения.

Пример

[Установка текущего принтера](#)

Поле ТекущийПринтер / CurrentPrinter

Класс Объект : [Принтер](#)

Описание

```
поле ТекущийПринтер :Строка;  
field CurrentPrinter :String;
```

Назначение

Возвращает имя текущего принтера, который программа использует для печати. Свойство доступно на запись, что позволяет программно сменить текущий принтер. Установка текущего принтера действует только на работающий экземпляр программы, вплоть до ее перезапуска. Для получения списка имен доступных принтеров используются свойства [КоличествоПринтеров](#) и [Принтер](#)

Если привоить свойству *CurrentPrinter* пустую строку, то будет установлен принтер, который является принтером по умолчанию в Windows.

Внимание: Принтер нельзя сменить, если открыто задание на печать, с помощью процедуры [НачатьПечать](#)

Пример

[Установка текущего принтера](#)

Поле АльбомныйРежим / LandscapeMode

Класс Объект : [Принтер](#)

Описание

```
поле АльбомныйРежим :Логическое;  
field LandscapeMode :Logical;
```

Назначение

Позволяет узнать или установить ориентацию бумаги при печати. Если *АльбомныйРежим* = False, то печать идет в книжной ориентации (вертикально), иначе - в альбомной ориентации (горизонтально).

Внимание: Ориентацию бумаги нельзя изменить, если открыто задание на печать, с помощью процедуры [НачатьПечать](#)

Пример

[Печать множества выделенных документов](#)

Поле Печатает / Printing

Класс Объект : [Принтер](#)

Описание

поле Печатает :Логическое;

field Printing :Logical;

Назначение

Позволяет узнать, открыто ли в настоящий момент задание на печать с помощью процедуры [НачатьПечать](#).

Свойство доступно только для чтения.

Описание

```
проц НачатьПечать ( [ИмяЗадания :Строка] );  
проц BeginPrint ( [TaskName :String] );
```

Назначение

Открывает задание на печать с именем *ИмяЗадания*.

Печать множества документов может быть сгруппирована в рамках одного задания, что позволяет значительно ускорить процесс. Процедура *НачатьПечать* открывает задание, после этого последовательность вызовов процедуры [Печать](#) будет буферизироваться. Собственно печать начнется при завершении задания с помощью процедуры [ЗавершитьПечать](#). Если вместо нее вызвать процедуру [ОтменитьПечать](#) то задание будет отменено и ни один документ не будет распечатан.

Вызовы процедур *НачатьПечать* / *ЗавершитьПечать* / *ОтменитьПечать* могут быть вложенными и должны быть сбалансированы, подобно транзакциям. Внимательно изучите пример, демонстрирующий правильную организацию заданий на печать.

Внимание: В рамках задания на печать нельзя сменить текущий принтер или ориентацию бумаги. Соответствующие настройки процедуры [Печать](#) будут проигнорированы. Устанавливайте принтер и ориентацию бумаги до открытия задания, с помощью свойств [ТекущийПринтер](#) и [АльбомныйРежим](#).

Пример

[Печать множества выделенных документов](#)

Поле ЗавершитьПечать / EndPrint

Класс Объект : [Принтер](#)

Описание

```
проц ЗавершитьПечать;  
proc EndPrint;
```

Назначение

Закрывает задание на печать, ранее открытое с помощью процедуры [НачатьПечать](#).

Если задание на печать не было открыто, вызов процедуры *ЗавершитьПечать* приведет к ошибке.

Пример

[Печать множества выделенных документов](#)

Поле ОтменитьПечать / AbortPrint

Класс Объект : [Принтер](#)

Описание

```
проц ОтменитьПечать;  
proc AbortPrint;
```

Назначение

Отменяет задание на печать, ранее открытое с помощью процедуры [НачатьПечать](#).

Если задание на печать не было открыто, вызов процедуры *ОтменитьПечать* приведет к ошибке.

Пример

[Печать множества выделенных документов](#)

Класс *СписокСтрок* (*StringList*), производный от классов [Объект](#) и [List](#), используется для работы со списками строк, которые являются атрибутами других классов языка ТБ.Скрипт, например, в объекте шаблона [Список](#) или клетке шаблона с полем перечисляемого типа.

Класс *СписокСтрок* наследует все свойства и методы от родительского класса [List](#), но в отличие от него может содержать только список элементов со строковыми значениями.

Внимание! Создать объект класса *СписокСтрок* напрямую нельзя. Списки создаются только внутри системы и используются как свойства других объектов, причем такие свойства имеют атрибут "только чтение".

Прикладной программист может изменять список на стадии проектирования с помощью редактора визуальных форм или на стадии исполнения программы с помощью указанных ниже методов и свойств класса *СписокСтрок*:

-  [Поле Количество / Count](#)
-  [Процедура Добавить / Add](#)
-  [Процедура Вставить / Insert](#)
-  [Процедура Удалить / Delete](#)
-  [Процедура Очистить / Clear](#)
-  [Процедура Упорядочить / Sort](#)
-  [Поле Пункты / Items](#)

Описание

Количество :Целое;
Count :Integer;

Назначение

Данное поле позволяет узнать количество пунктов в списке. Поле доступно только на чтение.

Пример

```
for i=1.. ComboBox1.List.Count do  
    ComboBox1.List.Items[i] = Str(i) + " " + ComboBox1.List.Items[i];  
end;
```

Описание

Пункты[Индекс :Целое]:Строка;
Items[Index :Integer]:String;

Назначение

Возвращает строку указанного по порядковому номеру пункта, а также позволяет изменить строку имеющегося пункта. Если указанный индекс выходит за пределы списка, генерируется ошибка.

Пример

```
proc P1;  
  for i=1.. ComboBox1.Count do  
    trace(ComboBox1.List.Items[i]);  
  end;  
end;
```

Процедура Вставить / Insert

Описание

Вставить (Позиция:Целое; Текст:Строка);
Insert (Position:Integer; Text:String);

Аргументы

Текст - произвольный текст;

Позиция - позиция, в которую требуется вставить строку.

Назначение

Вставляет указанную строку в список как новый пункт. Позиция нового пункта определяется значением первого аргумента. Позиция должна лежать в пределах от 1 до числа уже имеющихся в списке пунктов плюс 1.

Пример

```
CurrencyName : String[];  
-- ...  
for i=1.. CurrencyNumber do  
    ComboBox1.List.Insert(i,CurrencyName[i]);  
end;
```

Процедура Добавить / Add

Описание

```
Добавить(Текст :Строка);  
Add(Text :String);
```

Аргументы

Текст - произвольный текст.

Назначение

Добавляет указанную строку в список как новый пункт. Пункт добавляется в конец списка.

Пример

```
CurrencyName : String[];  
-- ...  
for i=1.. CurrencyNumber do  
    ComboBox1.List.Add(CurrencyName[i]);  
end;
```


Описание

Очистить;
Clear;

Назначение

Очищает список (удаляет все пункты).

Пример

```
proc СчитатьВалютыЗаново;  
  ComboBox1.List.Clear;  
  for i=1.. CurrencyNumber do  
    ComboBox1.List.Insert(i,CurrencyName[i]);  
  end;  
end;
```

Процедура Удалить / Delete

Описание

```
Удалить(Позиция :Целое);  
Delete(Position :Integer);
```

Аргументы

Позиция - номер удаляемого пункта списка.

Назначение

Удаляет указанный пункт из списка.

Пример

```
ComboBox1.List.Delete(1);
```

Описание

Упорядочить;
Sort;

Назначение

Сортирует список.

Пример

```
proc ЗаполнитьСписокВалют;  
  ComboBox1.List.Clear;  
  for i=1.. CurrencyNumber do  
    ComboBox1.List.Add(CurrencyName[i]);  
  end;  
  ComboBox1.List.Sort;  
end;
```

Класс *Таймер* (*Timer*), производный от класса *Объект*, позволяет встроить в прикладной проект функции таймера, то есть активизацию тех или иных действий через определенные промежутки времени. Класс *Таймер* наследует все свойства и методы от родительского класса [Объект](#).

Непосредственно в классе *Таймер* определены следующие свойства и методы:



[Функция Создать / Create](#)



[Поле Интервал / Interval](#)



[Поле Активен / Active](#)



[Процедура НаТаймер / OnTimer](#)

Описание

Активен :Логическое;
Active :Logical;

Назначение

Позволяет временно включить или отключить таймер. Когда поле имеет значение TRUE, таймер генерирует событие [НаТаймер](#) через каждый период времени, равный заданному [интервалу](#). По умолчанию, только что созданный таймер не активен.

Пример

```
x : Timer;

proc TimerAlarm(T:Timer);
  Hint("Текущее время:"+Str(Time));
end;

proc Button2Click (Sender :Button);
  -- создаем таймер с владельцем-бланком
  x = Timer.Create(self);
  -- устанавливаем обработчик событий
  x.OnTimer = "TimerAlarm";
  -- активизируем таймер
  x.Active = true;
end;
```

Описание

Интервал :Целое;
Interval :Integer;

Назначение

Позволяет узнать и изменить интервал времени между двумя последовательными событиями, генерируемыми таймером. Время задается в миллисекундах. По умолчанию, интервал нового таймера равен 1000 миллисекунд.

Пример

```
x : Timer;

proc TimerAlarm(T:Timer);
  Hint("Текущее время:"+Str(Time));
end;

proc Button2Click (Sender :Button);
  -- создаем таймер с владельцем-бланком
  x = Timer.Create(self);
  -- задаем интервал 10 секунд
  x.Interval = 10000;
  -- устанавливаем обработчик событий
  x.OnTimer = "TimerAlarm";
  -- активизируем таймер
  x.Active = true;
end;
```

Описание

```
НаТаймер(Таймер :Таймер);  
OnTimer(Timer :Timer);
```

Аргументы

Таймер - указатель на объект-таймер, от которого пришло событие. Таким образом, один обработчик события может использоваться для контроля нескольких таймеров.

Назначение

Позволяет установить имя процедуры-обработчика событий, генерируемых таймером. Событие генерируется с периодом, заданным с помощью свойства [Интервал](#). Если таймер не [активен](#), событие не происходит.

Процедура должна быть определена в классе того объекта, который является владельцем таймера (что указывается при создании таймера).

Пример

```
x : Timer;  
  
proc TimerAlarm(T:Timer);  
  Hint("Текущее время:"+Str(Time));  
end;  
  
proc Button2Click (Sender :Button);  
  -- создаем таймер с владельцем-бланком  
  x = Timer.Create(self);  
  -- устанавливаем обработчик событий  
  x.OnTimer = "TimerAlarm";  
  -- активизируем таймер  
  x.Active = true;  
end;
```

Описание

Создать (Владелец : [ПользовательскийОбъект](#)) : Таймер;
Create (Owner : [UserObject](#)) : Timer;

Аргументы

Владелец - пользовательский объект, которому будут направляться уведомления о генерируемых таймером событиях. Владелец таймера обязательно должен быть объектом пользовательского класса, то есть класса, реализованного на ТБ.Скрипт (не встроенного).

Назначение

Создает новый объект класса Таймер. По умолчанию, новый таймер не [активен](#), а его [интервал](#) установлен равным 1 секунде.

Пример

```
x : Timer;  
proc Button2Click (Sender :Button);  
  -- создаем таймер с владельцем-бланком  
  x = Timer.Create(self);  
  -- выводим текущие свойства таймера  
  message(x.Active);  
  message(x.Interval);  
end;
```


Класс *ТекстовыйФайл* (*TextFile*) является производным от класса *Объект*. С помощью объектов класса *ТекстовыйФайл* реализуется работа с текстовыми файлами: создание, запись, чтение. *ТекстовыйФайл* наследует от родительского класса все свойства, описанные в [разделе Объект](#).

В классе *ТекстовыйФайл* определены следующие свойства:

-  [Функция Создать / Create](#)
-  [Поле Имя / Name](#)
-  [Поле Размер / Size](#)
-  [Поле Позиция / Position](#)
-  [Поле КонецФайла / EOF](#)
-  [Функция Считать / Read](#)
-  [Процедура Записать / ДобавитьСтроку / Write](#)
-  [Функция СчитатьСтроку / ReadLn](#)
-  [Процедура ЗаписатьСтроку / WriteLn](#)
-  [Функция УстПозицию / Seek](#)
-  [Процедура Сбросить / Flush](#)

Поле Имя / Name

Описание

Имя:Строка;
Name:String;

Назначение

Возвращает полный путь, включающий имя файла.

Пример

```
var O : TextFile;  
O = TextFile.Create("Text.txt",fmOpenReadWrite);  
Message(O.Name);
```

Описание

КонецФайла :Логическое;
EOF :Logical;

Назначение

Возвращает значение True, если указатель файла достиг конца, и False - в противном случае.

Пример

```
var Text : TextFile;  
Text = TextFile.Create("TextA.txt",fmOpenReadWrite);  
while True do - эмулируем смещение в конец файла  
    Text.ReadLn;  
    if Text.EOF then -- если конец, прекращаем цикл  
        break;  
    fi;  
end;  
Text.WriteLine("abcd"); -- дописываем текст в конец файла
```

Описание

Позиция :Целое;
Position :Integer;

Назначение

Свойство возвращает текущую позицию в файле и доступно только на чтение.

Пример

```
var Text : TextFile;  
Text = TextFile.Create("TextA.txt",fmOpenReadWrite);  
Message("Текущая позиция:" + Str(Text.Position));
```

Поле Размер / Size

Описание

Размер :Целое;
Size :Integer;

Назначение

Возвращает размер файла в байтах.

Пример

```
var Text : TextFile;  
Text = TextFile.Create("TextA.txt",fmOpenReadWrite);  
Message("Размер файла:" + Str(Text.Size));
```

Описание

```
Записать(Текст:Строка);  
Write(Text:String);
```

Аргументы

Текст - строка, которую требуется записать в файл.

Назначение

Записывает в файл строку. Символ перевода строки не добавляется. Указатель внутри файла перемещается вперед на длину строки.

Пример

```
var O : TextFile;  
O = TextFile.Create("Text.txt",fmOpenReadWrite);  
O.Write("Новая строка");
```

Описание

```
ЗаписатьСтроку(Текст:Строка);  
WriteLn(Text:String);
```

Аргументы

Текст - строка, которую требуется записать в файл.

Назначение

Записывает в файл строку, добавляя в конце строки символ перевода строки. Указатель внутри файла перемещается вперед на длину строки.

Пример

```
var O : TextFile;  
O = TextFile.Create("Text.txt",fmOpenReadWrite);  
O.WriteLn("Новая строка");
```

Описание

Сбросить ;
Flush ;

Назначение

Процедура записывает содержимое буфера операционной системы Windows, на диск. Такая возможность позволяет в случае незапланированного выключения компьютера сохранить на диске информацию, помещенную ранее в буфер обмена.

Описание

Создать(Имя:Строка; Режим:Целое [;Кодировка:Логическое]) :ТекстовыйФайл;
Create(Name:String; How:Integer [;CharCode:Logical]):TextFile;

Аргументы

Имя - имя файла (может включать полный путь);

Режим - одна из predefined [констант](#), задающая режим работы с файлом;

Кодировка - логическое значение задает кодировку текста: Windows, если True или опущено, и кодировка DOS, если False.

Назначение

Создает или открывает файл с указанным именем в каталоге текущего проекта или в другом каталоге, если через аргумент **Имя** передается полный путь. Режим работы с файлом определяется вторым аргументом, который может быть равен одной из predefined констант:

- **рфСоздать (fmCreate)** - создать файл на запись. Если файл уже существовал, все его содержимое удаляется, а длина устанавливается равной нулю;
- **рфОткрНаЧтение (fmOpenRead)** - открыть существующий файл на чтение. Указатель файла устанавливается на начало. Если файла нет, генерируется исключительная ситуация;
- **рфОткрНаЗапись (fmOpenWrite)** - открыть существующий файл на запись. Содержимое существующего файла не удаляется, но указатель файла устанавливается на начало и при последующих операциях записи старая информация будет последовательно затираться новой; размер файла не меняется вплоть до того момента, когда объем записываемой информации превышает тот, что уже был в файле, после чего размер файла увеличивается синхронно с записью. Если файла нет, генерируется исключительная ситуация;
- **рфОткрНаЧтениеЗапись (fmOpenReadWrite)** - открыть существующий файл на запись и чтение; содержимое существующего файла не удаляется. Указатель файла устанавливается на начало. Последующие операции чтения и записи смещают указатель по файлу. Если указатель находится в пределах размера файла, записываемая информация "ложится" поверх старых данных, а в противном случае файл соответствующим образом расширяется; если файла нет, генерируется исключительная ситуация.

Необязательный параметр **Кодировка** задает кодовую таблицу текста. По умолчанию считается, что текст в кодировке Windows (CP-1251). Это эквивалентно значению True. Если передается значение False - текст в кодировке DOS (CP-866).

Пример

```
var O : TextFile;  
O = TextFile.Create("Text.txt",fmOpenReadWrite);  
O.ReadLn;  
O.WriteLine("исправление");  
O = nil;
```

Описание

```
Считать(Количество :Целое) :Строка;  
Read(Count :Integer) :String;
```

Аргументы

Количество - количество байт, которое требуется считать из любого нетекстового файла.

Назначение

Функция позволяет считать определенное количество байт из любого нетекстового файла. В случае если заказанное число байт превышает оставшееся до конца файла, то длина возвращаемой строки окажется меньше значения, заданного в параметре **Количество**. Если текущая позиция стоит за концом файла функция вернет пустую строку.

Описание

```
СчитатьСтроку :Строка;  
ReadLn :String;
```

Назначение

Считывает из файла следующую строку (последовательность символов, завершающуюся символом перевода строки) и возвращает ее. Сам символ перевода строки в возвращаемую строку не включается. Указатель внутри файла перемещается на следующий за строкой символ. Если указатель стоит в конце файла, функция возвращает пустую строку, однако это нельзя использовать как индикатор конца файла, так как в файле могут встречаться пустые строки. Для определения того, достигнут ли конец файла, следует использовать функцию [КонецФайла](#).

Пример

```
var O : TextFile;  
var S : String;  
O = TextFile.Create("Text.txt",fmOpenReadWrite);  
S = O.ReadLn;
```

Описание

```
УстПозицию(Позиция:Целое [; ИсхПозиция:Целое):Целое;  
Seek (Position :Integer[; Origin :Integer) :Integer;
```

Аргументы

Позиция - текущая позиция курсора;
ИсхПозиция - заданная позиция курсора.

Назначение

Функция перемещает курсор с текущей позиции курсора на заданную и возвращает номер установленной позиции.

Класс *Хранилище* / *Storage* является производным от родительского класса [Объект](#) и наследует от него все свойства и методы.

Экземпляр класса *Storage* может иметь произвольное количество полей типа вариант. При установке поля с именем, которого в объекте до этого не было, такое поле автоматически добавляется. Например,

```
var O : Storage;  
O = Storage.Create;  
O.A = 12;  
O.B = 3;
```

Здесь в предварительно созданный экземпляр O класса *Storage* добавляются поля A и B, установленные соответственно в 12 и 3. Также для обращения к полям хранилища можно использовать методы **SetField** и **GetField**.

Непосредственно в классе *Хранилище* / *Storage* определены следующие свойства:



[Функция Создать / Create](#)



[Функция ЕстьПоле / FieldExist](#)



[Процедура УдалитьПоле / DeleteField](#)



[Процедура УдалитьВсеПоля / DeleteAllFields](#)

Описание

```
УдалитьВсеПоля;  
DeleteAllFields;
```

Назначение

Процедура удаляет из хранилища информацию о всех заведенных в нем полях.

Пример

```
var Obj : Storage;  
Obj = Storage.Create;  
Obj.A = 12; -- создаем поле А и записываем в него число 12  
Obj.B = 5; -- создаем поле В и записываем в него число 5  
Obj.C = Obj.A* Obj.B;  
Obj.DeleteAllFields; -- удаляем поля А,В,С
```

Описание

```
УдалитьПоле(Имя:Строка);  
DeleteField(Name:String);
```

Аргументы

Имя - название поля.

Назначение

Процедура удаляет указанное поле из хранилища.

Пример

```
var Obj : Storage;  
Obj = Storage.Create;  
Obj.A = 12; -- создаем поле A и записываем в него число 12  
-- ...  
Obj.DeleteField("A"); -- больше это поле не доступно
```

Функция ЕстьПоле / FieldExist

Описание

```
ЕстьПоле(Имя:Строка) :Логическое;  
FieldExist(Name:String) :Logical;
```

Аргументы

Имя - название поля.

Назначение

Функция возвращает значение TRUE, если поле с указанным именем существует в хранилище, и FALSE - в противном случае.

Пример

```
var Obj : Storage;  
Obj = Storage.Create;  
Obj.A = 12;  
trace(Obj.FieldExist("А")); -- выводит "ИСТИНА"  
trace(Obj.FieldExist("В")); -- выводит "ЛОЖЬ"
```


Описание

Создать : [Хранилище](#);

Create : [Storage](#);

Назначение

Создает новый объект класса *Хранилище*. После того как в объекте будут динамически доопределены переменные, над ними можно будет выполнять вычисления с помощью функции [Вычислить](#). Также для обращения к полям объекта класса *Хранилище* можно использовать методы [SetField](#) и [GetField](#).

Пример

```
var глобличныеНастройки :Storage;

proc СоздатьХранилище;
  if (глобличныеНастройки = nil) then
    глобличныеНастройки = Storage.Create;
  end;
end;
```

Класс *ХранилищеПараметров / ParamStorage* является производным от родительского класса [Объект](#) и наследует от него все свойства и методы.

Объекты класса *ХранилищеПараметров* предназначены для хранения значений параметров типов счетов, их можно использовать в качестве параметров типовой операции. Объекты позволяют передавать в типовую операцию картотечных журналов, а затем в операторы "Проводка", "Дебет", "Кредит" список дополнительной аналитики. Поля объекта можно просматривать и изменять внутри типовой операции.

Привязка параметра такого типа должна осуществляться к массиву структур, при этом в структуре должны быть поля, имеющие следующие имена:

Название| Name - строковое, имя параметра;

Значение| Value - значение, ссылка на признак;

Расщепление| Split - целое, значение относится к дебету или кредиту (1-только к дебету, 2-только к кредиту, 0 - к обоим).

Поле **Расщепление** является необязательным, если оно отсутствует, все указанные значения будут относиться и к дебету и кредиту. Для передачи в проводку или полупроводку нужно присвоить "фиктивному" параметру проводки с именем **Признаки** значение параметра типа *ХранилищеПараметров*.

Дополнительную аналитику также можно добавлять и в табличные журналы, в записи которых имеется подтаблица **Признаки** для хранения дополнительной аналитики со следующими полями:





Название - строковое, имя параметра счета;

Значение2 - ссылочное значение;

Расщепление - целое, расщепление по дебету/ кредиту.

Для доступа к таблице нужно выполнить команду **Сервис| Просмотр записей** и в диалоге ["Записи"](#) выбрать класс записей TabJug. Данная подтаблица используется как источник данных для параметров типовых операций типа *ХранилищеПараметров*.

Непосредственно в классе *ХранилищеПараметров / ParamStorage* определены следующие свойства:

-  [Функция Создать / Create](#)
-  [Функция ЕстьПоле / FieldExists](#)
-  [Процедура УдалитьПоле / DeleteField](#)
-  [Процедура УдалитьВсеПоля / DeleteAllFields](#)

Описание

```
УдалитьВсеПоля;  
DeleteAllFields;
```

Назначение

Процедура удаляет информацию о всех полях, заведенных в объекте класса *ХранилищеПараметров*.

Пример

```
var Obj : ParamStorage;  
Obj = ParamStorage.Create;  
-- .....  
Obj.DeleteAllFields; -- удаляем все поля
```

Описание

```
УдалитьПоле(ИмяПоля :Строка);  
DeleteField(Fieldname :String);
```

Аргументы

ИмяПоля - название поля.

Назначение

Процедура удаляет указанное поле из объекта класса *ХранилищеПараметров*.

Пример

```
var Obj : ParamStorage;  
Obj = ParamStorage.Create;  
Obj.Признак = "значение признака"; -- создаем поле  
-- ...  
Obj.DeleteField("Признак"); -- удаляем поле
```

Описание

ЕстьПоле(ИмяПоля:Строка) :Логическое;
FieldExists(FieldName:String) :Logical;

Аргументы

ИмяПоля - название поля.

Назначение

Функция возвращает значение True, если в объекте класса *ХранилищеПараметров* существует поле с указанным именем, и False - в противном случае.

Пример

```
var Obj : ParamStorage;  
Obj = ParamStorage.Create;  
-- ....  
if (Obj.FieldExists("Признак")) then  
    Message("Поле с именем 'Признак' существует");  
fi;
```

Описание**Создать** : [ХранилищеПараметров](#)**Create** : [ParamStorage](#);**Назначение**

Создает новый объект класса *ХранилищеПараметров*. Объекты этого класса применяются в качестве параметров типовой операции и служат для хранения значений дополнительной аналитики, которыми помечаются проводки, сформированные этой типовой операцией.

Пример

```

опер БанковскаяВыписка
  (аСумма          :Numeric;
   аВалюта         :спрЕдИзм          = nil;
   аСчетДебета     :БазовыйСчет;
   аСчетКредита    :БазовыйСчет;
   аКонтрагент     :спрКонтрагент     = nil;
   аБанк           :спрБанковскийСчет = nil;
   аПроводитьСтроку :Logical          = True;
   аСчетФактура    :Integer           = 0;

   аНаше           :спрНашиПредприятия;
   аДатаПроводки   :Date;
   аНомерСтроки    :Integer           = 0;
   аДопПарам       :ParamStorage      = nil;
   аДопПарам_Общ   :ParamStorage      = nil;
   аОснование      :String            = "");

  var vClassSign :Class;

  if аПроводитьСтроку then
    if аДопПарам=nil then аДопПарам = ParamStorage.Create; end;
    ОбъединитьДопПараметры(аДопПарам_Общ, аДопПарам);
    аДопПарам.Счет_в_Банке = аБанк;
    аДопПарам.Контрагент   = аКонтрагент;
    ....
  end;
end;

прос ОбъединитьДопПараметры
( аCommExParameters :ParamStorage = nil;
  var аExParameters :ParamStorage = nil);

  var I          :Integer;
  var vParam     :String;
  var vValue     :Variant;
  ....
end

```

Класс *Шрифт (Font)*, производный от родительского класса [Объект](#), используется для работы со шрифтами Windows, используемыми в большинстве объектов ТБ.Скрипт, имеющих пользовательский интерфейс.

Ниже перечислены наследуемые переопределенные свойства, которые вводятся в классе *Шрифт*:

- [Поле Имя / Name](#)
- [Поле Размер / Size](#)
- [Поле Цвет / Color](#)
- [Поле Жирный / Bold](#)
- [Поле Наклонный / Italic](#)

Внимание! Создать объект класса *Шрифт* напрямую нельзя. Шрифты создаются только внутри системы и используются как свойства других объектов, причем такие свойства имеют атрибут "только чтение". Вместе с тем, программа на ТБ.Скрипт может получить доступ к объекту *Шрифт*, являющемуся свойством другого объекта, и изменять свойства самого шрифта.

Описание

Жирный : Логическое;
Bold: Logical;

Назначение

Позволяет узнать, используется ли жирное начертание в текущем шрифте, и изменить этот параметр.

Пример

```
-- меняем шрифт текста на кнопке
proc P1(B : Button);
  B.Font.Name = "MS Sans Serif";
  B.Font.Size = 10;
  B.Font.Bold = TRUE;
  B.Font.Italic = FALSE;
end;
```


Описание

Имя : Строка;
Name : String;

Назначение

Позволяет узнать имя используемого шрифта и изменить его.

Пример

```
-- меняем шрифт текста на кнопке
proc P1(B : Button);
  B.Font.Name = "MS Sans Serif";
  B.Font.Size = 10;
  B.Font.Bold = TRUE;
  B.Font.Italic = FALSE;
end;
```

Описание

Наклонный: Логическое;
Italic: Logical;

Назначение

Позволяет узнать, используется ли наклонное начертание в текущем шрифте, и изменить этот параметр.

Пример

```
-- меняем шрифт текста на кнопке
proc P1(B : Button);
  B.Font.Name = "MS Sans Serif";
  B.Font.Size = 10;
  B.Font.Bold = FALSE;
  B.Font.Italic = TRUE;
end;
```

Описание

Размер : Целое;
Size : Integer;

Назначение

Позволяет узнать размер используемого шрифта и изменить его.

Пример

```
-- меняем шрифт текста на кнопке
proc P1(B : Button);
  B.Font.Name = "MS Sans Serif";
  B.Font.Size = 10;
  B.Font.Bold = TRUE;
  B.Font.Italic = FALSE;
end;
```

Описание

Цвет : Целое;
Color : Integer;

Назначение

Позволяет узнать цвет используемого шрифта и изменить его. По умолчанию цвет – черный.

Пример

```
-- задает код цвета по трем составляющим: красной, зеленой и синей
-- интенсивность каждой составляющей варьируется от 0 до 255
func Colour(F: Font; Red:Integer; Green:Integer; Blue:Integer): Integer;
var R, G, B: Integer;
var C:Integer;
  -- составляющие цвета должны лежать в пределах 0-255
  R = Mod(Red,256);
  G = Mod(Green,256);
  B = Mod(Blue,256);
  C = R + G*256 + B*256*256;
  F.Color = C;
  return C;
end;
```

Класс *ЭлементТаблицы/GridItem* является производным от класса *Объект*. С помощью класса *ЭлементТаблицы/GridItem* реализуется работа с элементами таблицы класса [Таблица/Grid](#). *ЭлементТаблицы* наследует от родительского класса все свойства, описанные в [разделе Объект](#).

В классе *ЭлементТаблицы* определены следующие свойства:

-  [Поле Значение / Value](#)
-  [Поле Количество / Count](#)
-  [Поле Пункты / Items](#)
-  [Поле Родитель / Parent](#)
-  [Поле РодИндекс / ParentIndex](#)
-  [Поле ЭтоГруппа / IsGroup](#)
-  [Поле Раскрыта / Opened / Раскрыт](#)
-  [Поле Изображение / Image](#)
-  [Процедура Добавить / Add](#)
-  [Процедура Вставить / Insert](#)
-  [Процедура Удалить / Delete](#)

Поле Значение / Value

Описание

Значение: Вариант;

Value:Variant;

Назначение

Свойство позволяет узнать и изменить значение элемента таблицы. Если данное поле является массивом, то будет заполнены все колонки в одной строке таблицы. Иначе, только элемент одной колонки.

Поле доступно как на чтение, так и на запись.

Поле Изображение / Image

Описание

Изображение : [Изображение](#) ;

Image : [Image](#) ;

Назначение

Поле позволяет получить или установить рисунок на элемент таблицы.

Поле доступно на чтение и на запись.

Поле Количество / Count

Описание

Количество: Целое;

Count : Integer;

Назначение

Поле позволяет узнать количество элементов в таблице. Группа считается одним элементом, т.е. элементы в группе и ее подгруппах не учитываются.

Поле доступно только на чтение.

Пример

```
Grid1 :Grid;
-- ...
К :Integer;

-- узнаем количество корневых элементов таблицы

К = Grid1.Root.Count;
-- ...
```


Описание

Пункты[Индекс :Целое] :[ЭлементТаблицы](#);
Items[Index :Integer] :[GridItem](#);

Назначение

По заданному индексу позволяет прочитать элемент таблицы. Поле доступно только на чтение.

Индекс должен лежать в пределах от 1 до общего числа элементов в перечислении (свойство [Count](#)).
Иначе генерируется ошибка.

Пример

```
Grid1 :Grid;  
  
Proc P1(Sender :Button);  
  var i :Integer;  
  -- ....  
  for i=1.. Grid1.Root.Count do  
    -- поочередно выдаем сообщение о значении корневых элементов таблицы, начиная  
    с первого элемента до его количества.  
    Message(Grid1.Root.Items[i].Value);  
  end;  
end;
```

Описание

Раскрыта :Логическое;

Раскрыт :Логическое;

Opened :Logical;

Назначение

Свойство позволяет узнать и установить у группы признак раскрытости. Если свойство возвращает True, то группа таблицы будет раскрыта. False - наоборот.

Поле доступно на чтение и на запись.

Описание

РодИндекс :Целое;
ParentIndex :Integer;

Назначение

Свойство возвращает номер порядкового элемента в группе таблицы.

Поле доступно только на чтение.

Пример

```
Grid1 :Grid;  
  
vItem :GridItem;  
  
vItem = Grid1.Current;  
  
-- ....  
if vItem.ParentIndex = vItem.Parent.Count then  
    Message ("В группе " + Str(vItem.ParentIndex) + " элементов" );  
end;  
  
-- ....
```

Описание

Родитель : [ЭлементТаблицы](#);

Parent : [GridItem](#);

Назначение

Свойство возвращает указатель на родительский элемент (группу) таблицы.

Поле доступно только на чтение.

Пример

```
Grid1 :Grid;  
  
vItem :GridItem;  
  
-- ...  
vItem = Grid1.Current.Parent;  
  
if vItem.Count > 5 then  
    Message ("В группе " + Str(vItem.Count) + " элементов");  
end;  
-- ...
```

Поле ЭтоГруппа / IsGroup

Описание

ЭтоГруппа :Логическое;

IsGroup :Logical;

Назначение

Свойство позволяет узнать и установить у элемента признак группы. Если свойство возвращает True, то элемент таблицы будет являться группой. False - наоборот.

Поле доступно на чтение и на запись.

Процедура Вставить / Insert

Описание

Вставить(Индекс :Целое; Значение :Вариант);
Insert(Index :Integer; Value :Variant);

Аргументы

Индекс - индекс, на основании которого определяется позиция вставляемого значение;
Значение - вставляемое значение.

Назначение

Вставляет указанное значение в позицию, индекс которой на единицу больше, заданного в первом параметре. Значение индекса должно лежать в пределах от 1 до количества, уже имеющихся индексов, плюс один.

Пример

```
Grid1 :Grid;  
  
proc ButInsert_OnClick(Sender :Button);  
  var I, J :Integer;  
  
  J = Grid1.CurrentRow;  
  for I = 1..5 do  
    Grid1.Root.Insert(J, "12345")  
  end;  
  -- ...  
end;
```

Процедура Добавить / Add

Описание

```
Добавить(Значение :Вариант);  
Add(Value :Variant);
```

Аргументы

Значение - добавляемое значение.

Назначение

Процедура позволяет добавить элемент в конец списка таблицы.

Пример

```
Grid1 :Grid;  
  
proc ButAdd_OnClick (Sender :Button);  
  var I :Integer;  
  
  for I = 1..10 do  
    Grid1.Root.Add("1234");  
  end;  
  -- ....  
end;
```

Процедура Удалить / Delete

Описание

```
Удалить(Индекс :Целое);  
Delete(Index :Integer);
```

Аргументы

Индекс - номер удаляемого элемента таблицы.

Назначение

Удаляет указанный элемент из таблицы.

Пример

```
Grid1 :Grid;  
  
proc ButDelete_OnClick (Sender :Button);  
  var I, N :Integer;  
  -- ....  
  N = Grid1.Root.Count;  
  if I <= N then  
    Grid1.Root.Delete(I);  
  end;  
  -- ...  
end;
```


DBFФайл / DBFFile
OLEДокумент / OLEDocument
OLEКонтейнер / OLEContainer
АвтоОбъект / AutoObject
График / Chart
ГрафикОтчета / ReportChart
ДвоичныйОбъект / BinaryObject
ДеревоКартотеки / CardTree
Запись / Record / Документ / Document
Запрос / Query
ЗапросАналитики / AnalitQuery
ЗапросПолупроводок / HTransQuery
ЗапросСчетов / AccQuery
Изображение / Image
Изоляция / Isolation
Импортер / Importer
Картотека / CardFile
КлеткаШаблона / TemplateCell
Кнопка / Button
Надпись / Label
НастройкиГрафика / ChartSettings
ОбластьШаблона / TemplateRange
Объект - свойства базового класса
ОбъектШаблона / TemplateObject
Окно / Window
Операции / Operation
ОписаниеПартии / PartyInfo
Отчет / Report
Переключатель / RadioButton
Подтаблица / Subtable
ПодтаблицаКартотеки / SubCardfile
ПодтаблицаШаблона / TemplateSubCardfile
Полупроводка / HTrans
ПользовательскийОбъект / UserObject
ПраваДоступа / AccessRights
Признак / Sign
Проигрыватель / MediaPlayer
Рамка / Bevel
Редактор / Edit
Рисунок / Picture
СекцияШаблона / TemplateSection
Список / ComboBox
СписокЗаписей / RecordList
СписокСтрок / StringList
СтолбецКартотеки / CardfileColumn
СтолбецШаблона / TemplateColumn
СтрокаШаблона / TemplateRow
Структура / Structure
Таймер / Timer
ТекстовыйФайл / TextFile
Транзакция / Transaction
Флаг / CheckBox
Форма / Form
ФормаБланка / BlankForm
ФормаКартотеки / CardForm
ФормаОтчета / ReportForm
Хранилище / Storage
Шаблон / Template
Шрифт / Font
Экспортер / Exporter

Все классы, производные от родительского класса [Объект](#), сгруппированы по своему назначению на группы.

В группу пользовательских классов входят следующие классы, перечисленные с учетом их иерархии:

- [ПользовательскийОбъект / UserObject](#)
- [ДинамическийОбъект / DynamicObject](#)
- [Форма / Form](#)
 - [ФормаБланка / BlankForm](#)
 - [ФормаОтчета / ReportForm](#)
 - [ФормаКартотеки / CardForm](#)
- [Команда / Command](#)
- [ПраваДоступа / AccessRights](#)
- [ОбщийДоступ / CommonAccess](#)

Классы визуальных форм

Все классы форм являются производными от родительских классов [Объект](#) и [ПользовательскийОбъект](#) и наследует от них все свойства и методы.

Классы форм применяются при разработке экранных форм бланков, картотек и отчетов программным способом и позволяют упростить этот процесс за счет использования всех возможностей, предоставляемые графической операционной системой Windows.

Классы визуальных форм позволяют создавать формы любой степени сложности и включают следующие классы:

[Форма](#)

[ФормаБланка](#)

[ФормаКартотеки](#)

[ФормаОтчета](#)






Класс *Форма* используется в качестве базового класса для производных классов ФормаБланка, ФормаКартотеки и ФормаОтчета, которые обеспечивают работу с пользовательскими электронными формами (бланками, отчетами и картотеками) Студии.


Форма / Form

Класс *Форма* / *Form* является производным от классов [Объект](#) и [ПользовательскийОбъект](#) и наследует все свойства родительских классов.

Класс *Форма* используется в качестве базового класса для производных классов [ФормаБланка](#) [ФормаКартотеки](#) [ФормаОтчета](#), которые обеспечивают работу с пользовательскими электронными формами (бланками, отчетами и картотеками) Студии.

Непосредственно в классе *Форма* / *Form* определены следующие свойства и перечислимые типы:

-  [Поле Объекты / Objects](#)
-  [Функция СоздатьВидимым / CreateVisible](#)
-  [Функция ВыполнитьДиалог / ExecuteDialog](#)
-  [Функция ПоказатьФорму / ShowForm](#)
-  [Функция ИмяФайлаКласса / ClassFileName](#)
-  [Поле ТипКласса / ClassType](#)
-  [Поле Окно / Window](#)
-  [Поле Шаблон / Template](#)
-  [Поле РодФрейм / ParentFrame](#)
-  [Поле Виден / Видна / Visible](#)
-  [Поле ИмяРаскладки / LayoutName](#)
-  [Функция Выполнить / Execute](#)
-  [Функция Показать / Show](#)
-  [Процедура Заккрыть / Close](#)
-  [Процедура Печать / Print](#)
-  [Функция Экспорт / Export](#)
-  [Процедура ЗагрузитьШаблон / LoadTemplate](#)
-  [Процедура ЗагрузитьШаблонДоп / LoadTemplateEx](#)
-  [Процедура ЗагрузитьФорму / LoadForm](#)
-  [Процедура ПереинициализироватьПеременные / ReinitVariables](#)
-  [Процедура СохранитьРаскладку / SaveLayout](#)
-  [Процедура ЗагрузитьРаскладку / LoadLayout](#)

-  [Тип ТипыЭкспорта / ExportTypes](#)
-  [Тип ОпцииЭкспорта / ExportOptions](#)

Перечислимый тип **ОпцииЭкспорта / ExportOptions** в классе **Форма** предназначен для задания опций экспорта, которые используются в функции [Экспорт](#).

В данном типе определены следующие константы общего назначения:

- **ПоказыватьМастер|ShowWizard** - логический параметр, по умолчанию его значение равно False. Если значение равно True, будет открыт мастер экспорта бланка в соответствующий формат;
- **ИгнорироватьЦвета|IgnoreColors** - логический, определяющий, следует ли при экспорте учитывать заданные цвета элементов, по умолчанию значение параметра равно False;
- **КакПриПечати|AsPrinted** - логический параметр, по умолчанию его значение равно False. При экспорте учитывается видимость элементов на экране (КакПриПечати=False) или видимость при печати (КакПриПечати=True).

Данный тип включает также специальные константы логического типа, которые используются только при экспорте в формат HTML (как правило, они увеличивают размер файла HTML):

- **СовместимостьCWord|WordCompatible** - добавление параметров, для лучшего открытия HTML в MSWord (True), по умолчанию значение параметра равно False;
- **СовместимостьCExcel|ExcelCompatible** - добавление параметров для лучшего открытия HTML в MSExcel (True), по умолчанию значение параметра равно False;
Внимание. Следует иметь в виду, что HTML-файл, записанный с опцией СовместимостьCExcel=True, при открытии не с помощью программы Excel может оказаться менее похожим на то, что отображается на бланке. Например, числа потеряют форматирование (если формат численной клетки = ",#0.0 руб", то вместо "5'250.5 руб" в ТБ будет показано "5250.5" в Explorer'e).
- **РазбиватьНаТаблицы|BreakByTables** - для каждой секции формирует отдельную HTML-таблицу;
- **ВключатьСубФреймы|IncludeSubFrames** - делает выгрузку всех дочерних фреймов для объекта, переданного через параметр Фрейм [функции Экспорт](#).

Перечислимый тип **ТипыЭкспорта / ExportTypes** в классе **Форма** предназначен для задания формата файла, в который будут экспортироваться данные из текущей формы.

В данном типе определены следующие константы:

- **ToBMP** - экспорт изображение формы в формат BMP;
- **ToJPG** - экспорт изображение формы в формат JPG;
- **ToGIF** - экспорт изображение формы в формат GIF;
- **ToRTF** - экспорт формы в документ MS WORD;
- **ToHTML** - экспорт формы в формат документа HTML;
- **ToTPL** - экспорт формы в шаблон бланк.

Константы, определенные в данном типе, используются в функции [Экспорт](#).

Поле Виден / Видна / Visible

Описание

Виден : Логическое;
Visible : Logical;

Назначение

Свойство определяет, видна ли форма или нет (фактически, это индикатор того, создано ли окно для формы). Изменяя значение поля, можно отображать и временно скрывать форму.

Пример

```
var F: БланкНакладная;  
F = БланкНакладная.Создать;  
F.Visible = True;
```

Описание

ИмяРаскладки :Строка;
LayoutName :String;

Назначение

Свойство позволяет узнать или установить имя, под которым будет сохраняться пользовательская настройка картотеки. В свойстве **ИмяРаскладки** указывается *логическое имя* (короткое имя файла без расширения). По умолчанию оно совпадает с именем класса (с проектом). Его можно изменить в обработчике **ПриСоздании|OnCreate**. Если в поле хранится пустая строка (ИмяРаскладки = ""), то пользовательские настройки сохраняться не будут.

Свойство используется при открытии или закрытии картотеки. Если требуется открыть картотеку под другим именем (с другими настройками) "на лету", не закрывая ее, то используется процедура [СохранитьРаскладку](#).

Замечание. В настоящее время свойство действует только для картотечных форм. Реально раскладка сохраняется в папке в файле с расширением *.bro. Поле не должно содержать символов ("*", ",", " и т.п.), недопустимых в именах файлов.

Пример

```
-- фрагмент кода
var FCurLayout :String;

func Card_OnChangeGroup(Group :Record) :Logical;
var vNewLayout :String;
if FCurLayout <> "" then
    SaveLayout(FCurLayout);
    FCurLayout = "";
end;
vNewLayout = LayoutName + "." +
    If(Group <> nil, Group.ExtID, "nil");
LoadLayout(vNewLayout);
Result = True;
end;
```


Описание

Объекты[Индекс: Целое] : [Форма](#);
Objects [Index: Integer] : [Form](#);

Аргументы

Индекс - номер формы, который может изменяться в пределах от 1 до числа форм.

Назначение

Данное свойство позволяет получить доступ к любой из созданных к данному моменту форм по номеру. Поле доступно только на чтение.

Пример

```
-- по нажатию кнопки скрываем все экземпляры
-- документа кроме одного
proc Button2Click (Sender :Button);
var i: integer;
for i=1..self.ЧислоОбъектов do
  if self.Объекты[i] <> self then
    if self.Объекты[i].Виден then
      self.Объекты[i].Виден = false;
    end;
  end;
end;
end;
```

Описание

Окно : [Окно](#);
Window : [Window](#);

Назначение

Свойство обеспечивает доступ к объекту типа [Окно](#), в котором отображается форма, и тем самым дает возможность изменять атрибуты окна. Поле доступно только на чтение.

Пример

```
proc ПриСчитывании;  
  var DocEx:Движение.ПриходРасход;  
  -- если этот документ уже открыт в одном из окон  
  DocEx=ПроверитьЧтоУжеОткрыт(Document);  
  if DocEx<>NIL then  
    message('Документ уже открыт');  
    -- показываем пользователю это окно  
    DocEx.Window.Наверх;  
    -- и закрываем новое, в котором отпала необходимость  
    Close;  
  fi;  
end;
```

Класс Объект : [ПользовательскийОбъект](#) : [Форма](#)

Поле РодФрейм / ParentFrame

Описание

РодФрейм : [ФреймШаблона](#);

ParentFrame : [TemplateFrame](#);

Назначение

Возвращает nil, если форма не была подгружена во фрейм. Эти сведения необходимы при открытии бланка. Поле доступно только на чтение.

Пример

Описание

ТипКласса : Класс [Форма](#);
ClassType: Class [Form](#);

Назначение

Возвращает указатель на класс **Форма** (именно на класс, а не на объект класса). Указатель позволяет создавать объекты на базе существующих (клонировать, т.е. создавать их точную копию), проверять, принадлежат ли объекты одному классу или разным, а также определять наследственные связи в иерархии классов.

Поле доступно только на чтение.

Пример

```
func F1(Ob1:Form; Ob2:Form):Logical;  
  var L : Logical;  
  if Ob1.ClassType = Ob2.ClassType then  
    Message("Объекты принадлежат одному классу");  
    L = TRUE;  
  else  
    Message("Объекты принадлежат разным классам");  
    L = FALSE;  
  fi;  
  return L;  
end;
```

Поле Шаблон / Template

Описание

Шаблон : [Шаблон](#);
Template : [Template](#);

Назначение

Свойство обеспечивает доступ к объекту класса [Шаблон](#), на основе которого создана форма. С помощью данного поля программа может управлять атрибутами шаблона. Поле доступно только на чтение.

Пример

```
-- если нужно обновить (перерисовать) форму на экране  
ФормаНакладная.Template.Update;
```

Описание

```
ЗагрузитьРаскладку({ИмяРаскладки :Строка});  
LoadLayout({LayoutName :String});
```

Аргументы

ИмяРаскладки - логическое имя (короткое имя файла без расширения), по которому будет восстанавливаться настройка. Если имя не задано, то в качестве имени настройки берется имя формы.

Назначение

Процедура позволяет восстанавливать пользовательскую настройку формы в любой момент времени, не переоткрывая картотеку, при условии, что эти настройки ранее были [сохранены в заданном файле](#).

Замечание. В настоящее время этот программный интерфейс действует только для картотек.

Пример

```
-- фрагмент кода  
var FCurLayout :String;  
  
func Card_OnChangeGroup(Group :Record) :Logical;  
  var vNewLayout :String;  
  if FCurLayout <> "" then  
    SaveLayout(FCurLayout);  
    FCurLayout = "";  
  end;  
  vNewLayout = LayoutName + "." +  
    If(Group <> nil, Group.ExtID, "nil");  
  LoadLayout(vNewLayout);  
  Result = True;  
end;
```

Описание

```
ЗагрузитьФорму(Фрейм : ФреймШаблона; Форма : Форма);  
LoadForm(Frame : TemplateFrame; Form : Form);
```

Аргументы

Фрейм - ссылка на фрейм, в который загружается заданная форма;

Форма - ссылка на форму, загружаемую во фрейм шаблона бланка, из которого вызывается данный метод.

Назначение

Процедура загружает заданную по ссылке форму во фрейм того бланка, который вызывает данный метод.

Пример

```
InObject Private  
    ФреймДолжностиПрофессии :TemplateFrame;  
  
    proc шаблон_ПриОткрытии(Create :Logical);  
        inherited шаблон_ПриОткрытии(Create);  
        LoadForm(ФреймДолжностиПрофессии,  
            Справочники.картРольСубъекта.Create);  
    end;
```

Описание

```
ЗагрузитьШаблон(ИмяФайла :Строка [; НомерСекции :Целое} [; СчитыватьОбъекты :Логическое]] );  
  
LoadTemplate( FileName :String [; NumberSection :Integer} [; LoadObjects :Logical]] );
```

Аргументы

ИмяФайла - имя файла (по умолчанию, с расширением TPL), из которого будет загружаться шаблон;
НомерСекции - номер секции текущего шаблона, за которой следует загрузить шаблон;
СчитыватьОбъекты - признак, указывающий нужно ли загружать объекты, находящиеся на подгружаемом шаблоне, или нет.

Назначение

Загружает секции указанного шаблона в текущую форму (форму бланка, отчета, картотеки). Элементы управления, такие как кнопки, флаги, переключатели и т.д., из указанного шаблона переносятся в текущую форму только в том случае, если параметр **СчитыватьОбъекты** равен TRUE. По умолчанию (т.е. если опущен) этот параметр равен FALSE, и объекты не подгружаются.

Внимание. При подгрузке шаблона не подгружается библиотека стилей. Поэтому следите, чтобы библиотеки стилей подгружаемого шаблона и шаблона, в который он подгружается, были одинаковы.

Если второй параметр опущен – шаблон перегружается полностью, если номер указан – то шаблон подгружается за секцией текущей формы с заданным номером. Особый случай, когда **НомерСекции** равен -1: при этом шаблон подгружается в конец формы.

Данную процедуру можно выполнить только при имеющемся объекте шаблона (т.е. шаблон формы, в которую производится загрузка, уже должен быть проинициализирован, что, например, соответствует действительности в момент генерации события **ПриОткрытии** или же позже в любое другое время, когда бланк открыт).

Пример

```
-- код формы бланка  
proc ButtonAppendix_OnClick(Sender :Button);  
    -- по нажатию кнопки в конец бланка подгружаются  
    -- секции приложения  
    Template. LoadTemplate ("Appendix.tpl",-1);  
end;
```


Описание

```
ЗагрузитьШаблонДоп(Фрейм : ФреймШаблона; ИмяФайла :Строка);  
LoadTemplateEx(Frame : TemplateFrame; FileName :String);
```

Аргументы

Фрейм - ссылка на фрейм, который будет загружаться;

ИмяФайла - имя файла (по умолчанию, с расширением TPL), из которого будет загружаться фрейм шаблон.

Назначение

Загружает указанный фрейм шаблона в текущую форму (форму бланка, отчета, картотеки).

Внимание. При подгрузке шаблона не подгружается библиотека стилей. Поэтому следите, чтобы библиотеки стилей подгружаемого фрейма шаблона и шаблона, в который он подгружается, были одинаковы.

Данную процедуру можно выполнить только при имеющемся объекте шаблона, т.е. шаблон формы, в которую производится загрузка, уже должен быть проинициализирован, что, например, соответствует действительности в момент генерации события **ПриОткрытии** или же позже в любое другое время, когда бланк открыт.

Пример

```
RootRow :TemplateRow;  
AllFrames :TemplateFrame;  
MySect :TemplateSection;  
IndexFrame : Integer;  
  
proc Button1_OnClick(Sender :Button);  
  var C :class;  
  var I,J :Integer;  
  C = FindClass(BlankName);  
  if C <> nil and C is class BlankForm then  
    AllFrames.Clear;  
    LoadTemplateEx(AllFrames, C.ClassFileName('tpl'));  
    IndexFrame = RootRow.Number - 1;  
    for I = (IndexFrame + 1)..MySect.RowsCount do  
      for J = 1..MySect.ColumnsCount do  
        MySect.Cell[J, i].Contents = '';  
        MySect.Cell[j, i].Color = RGB(255, 255, 255);  
      end;  
    end;  
    ...  
  end;  
end;
```

Описание

```
Закреть[(КодВозврата: Целое)];  
Close[(ResultCode: Integer)];
```

Аргументы

КодВозврата - необязательный аргумент, значение, которое будет возвращено окном формы, если оно было открыто в модальном режиме. Если параметр опущен, то при закрытии окна, по умолчанию, возвращается cmCancel (кmdОтказ). Другие стандартные значения: cmOK (кmdВерно), cmYes (кmdДа), cmNo (кmdНет).

Назначение

Процедура закрывает экземпляр формы. Если форма была открыта в модальном режиме, с помощью необязательного параметра задается код возврата (ModalResult).

Этот метод наследуется производными классами.

Пример

```
proc КнСохранить(В:Button);  
    Close(cmOK);  
end;  
  
proc КнОтменить(В:Button);  
    Close(cmCancel);  
end;
```

Описание

ПереинициализироватьПеременные;
ReinitVariables;

Назначение

Данный метод необходимо вызывать после программной вставки/удалении столбцов или строк, чтобы синхронизировать такие свойства объекта как [СтолбецШаблона](#) с их истинными значениями.

Пример

```
proc шаблон_ПриОткрытии(Create :Logical);  
    ....  
    ReinitVariables;  
    ...  
end;
```

Описание

```
Печать[[[ИмяПринтера: Строка]; [СхемаНастроек: Строка]]];  
Print[[[PrinterName: String]; [Scheme: String]]];
```

Аргументы

ИмяПринтера - имя принтера, определенное на системном уровне;

СхемаНастроек - название схемы настроек, заведенных пользователем для работы с принтером в проектах Студии.

Назначение

Процедура выводит на печать открытую форму (то есть форма должна быть отображена в окне). Если какой-либо или оба параметра опущены, используются принтер и настройки по умолчанию.

Пример

```
proc BtnClick(Sender:String);  
  if Sender = "BtnOk" then  
    if (State = Template.Edited) Or (State = Template.Created) then  
      Document.Post;  
    end;  
    Close(cmOk);  
  elseif Sender = "BtnCancel" then  
    if (State = Template.Edited) Or (State = Template.Created) then  
      Document.Cancel;  
    end;  
    Close(cmCancel);  
  elseif Sender = "BtnPrint" then  
    Self.Print; -- печатаем форму  
  end;  
end;
```

Описание

```
СохранитьРаскладку({ИмяРаскладки :Строка});  
SaveLayout({LayoutName :String});
```

Аргументы

ИмяРаскладки - необязательное логическое имя (короткое имя файла без расширения), под которым будет сохраняться настройка. Заданное имя используется при [восстановлении настройки](#). Если вместо имени задана пустая строка ИмяРаскладки = "", то пользовательские настройки сохраняться не будут.

Замечание. Имя не должно содержать символов ("*", ",", " и т.п.), недопустимых в именах файлов.

Назначение

Процедура позволяет сохранять пользовательскую настройку формы в любой момент времени, не переоткрывая картотеку. Если аргумент не задан, то в качестве имени настройки берется имя формы.

Замечание. В настоящее данное свойство действует только для картотечных форм. Реально раскладка сохраняется в папке в файле с расширением *.bro.

Пример

```
-- фрагмент кода  
var FCurLayout :String;  
  
func Card_OnChangeGroup(Group :Record) :Logical;  
  var vNewLayout :String;  
  if FCurLayout <> "" then  
    SaveLayout(FCurLayout);  
    FCurLayout = "";  
  end;  
  vNewLayout = LayoutName + "." +  
    If(Group <> nil, Group.ExtID, "nil");  
  LoadLayout(vNewLayout);  
  Result = True;  
end;
```

Функция Выполнить / Execute

Описание

Выполнить : Целое;

Execute : Integer;

Назначение

Функция открывает форму в модальном окне, ожидает закрытия диалога и возвращает результат работы (ModalResult).

Пример

```
var F: БланкНастройки;  
F = БланкНастройки.Создать;  
if cmOK = F.Выполнить then  
    МассивНастроек = F.КакСтроки;  
end;
```

Описание

ВыполнитьДиалог : Целое;
ExecuteDialog : Integer;

Назначение

Функция создает новый объект класса **Форма**. Этот метод наследуется производными классами.

Создаваемая форма производных классов сразу после создания отображается на экране в модальном окне. Возврат управления из функции не происходит до тех пор, пока не закрыто окно диалога. Как только окно закрывается, объект уничтожается, а функция возвращает результат работы диалога (ModalResult).

Пример

```
if кмдВерно = БланкНастройка.ВыполнитьДиалог then  
    Hint("Новые значения приняты");  
end;
```

Описание

```
ИмяФайлаКласса[(ТипФайла :Строка)] :Строка;  
ClassFileName[(FileType :String)] :String;
```

Аргументы

ТипФайла - строка с расширением файла: "tpl", "bro" или "rep"; если параметр опущен, предполагается тип "tpl".

Назначение

Возвращает имя файла (заданного типа), формирующего данный класс. В связи с тем, что классы бланков, картотек и отчетов могут наследовать шаблоны и настройки, имена используемых файлов не всегда совпадают с именем класса.

В случае отсутствия файла заданного типа у данного класса возвращается имя соответствующего файла из класса-предка. Если такой файл отсутствует у всех предков, то возвращается пустая строка.

Пример

```
InObject  
--...  
-- при отсутствии шаблона у формы данного класса  
-- подгружаем стандартный  
if self.ClassType.ClassFileName("tpl") = "" then  
  self.LoadTemplate("default.tpl", -1);  
end;
```


Описание

Показать(СтильОкна : [Окно.СтилиОкон](#)): Целое;
Show(WindowStyle: [Window.WindowStyles](#)): Integer;

Аргументы

СтильОкна - предопределенная [константа](#), которая определяет способ открытия окна. Если параметр не задан, то он считается равным AutoDetect.

Назначение

Функция создает форму (если она не существует) объекта из класса **Форма** или из производного от него класса и отображает ее на экране. В зависимости от результата работы функция возвращает одну из предопределенных констант класса **Консоль**: *кmdВерно*, *кmdОтказ*, *кmdДа* или *кmdНет*.

Форма в системе может открываться в разных режимах: в дочернем, модальном, плавающем или встраиваемом окне, что определяется аргументом **СтильОкна** функции **Show**.

Внимание. Данный метод **Показывать/Show** перекрывает возможности метода [Выполнить / Execute](#). Настоятельно рекомендуется перейти везде на вызов данного метода, так как старые методы в ближайших релизах планируются удалить.

Пример

```
-- фрагмент кода бланка
proc ВыполнитьОтчет( aRec :Ядро.Отчеты );
  var локФормаОтч :ReportForm;

  локФормаОтч =
    CreateReport(ВернутьПолнИмяОтч(aRec));
  if локФормаОтч <> nil then
    локФормаОтч.Show(Window.AutoDetect);
  end;
end;

func ВернутьПолнИмяОтч(aRec
  :Ядро.Отчеты) :String;
  var Rec :Record;
  Rec = Record(aRec);
  if (Rec <> nil) then
    Result = Rec.Name as String;
    while (Rec.GroupDoc <> nil) do
      Result = (Rec.GroupDoc.Name)
        as String + '.' + Result;
      Rec = Rec.GroupDoc as Record;
    end;
  end;
end;
```

Описание

ПоказатьФорму(СтильОкна : [Окно.СтилиОкон](#)) : Целое;
ShowForm(WindowStyle : [Window.WindowStyles](#)) : Integer;

Аргументы

СтильОкна - предопределенная [константа](#), которая определяет способ открытия окна с формой класса. Если параметр не задан, то он считается равным AutoDetect.

Назначение

Функция создает форму (если она не существует) класса **Форма** или производного от него класса и отображает ее на экране. В зависимости от результата работы функция возвращает одну из предопределенных констант класса **Консоль**: *кmdВерно*, *кmdОтказ*, *кmdДа* или *кmdНет*.

Форма в системе может открываться в разных режимах: в дочернем, модальном, плавающем или встраиваемом окне, что определяется аргументом **СтильОкна** функции **Show**.

Внимание. Данный метод **ПоказыватьФорму/ShowForm** перекрывает возможности метода [СоздатьВидимым / CreateVisible](#). Настоятельно рекомендуется перейти везде на вызов данного метода, так как старые методы в ближайших релизах планируется удалить.

Пример

```
proc ПострОтчРасш(аЛюбой :Logical);
var I          :Integer;
var локКлассОтч :Class ReportForm;
var локОтчет    :ReportForm;

локКлассОтч=FindClass('Контроль.Контроль.ОтчетСписокДел')
  as Class ReportForm;
Assert(локКлассОтч <> nil);
if (локКлассОтч.ObjectsCount = 0) then
  локКлассОтч.ShowForm;
else
  for I = 1..локКлассОтч.ObjectsCount do
    локОтчет = локКлассОтч.Objects[I] as ReportForm;
    if аЛюбой or not локОтчет.ПользовОтчет then
      локОтчет.Window.GoTop;
      return;
    end;
  end;
  локКлассОтч.ShowForm;
end;
end;
```

Описание

СоздатьВидимым : [Форма](#);
CreateVisible : : [Form](#);

Назначение

Функция создает новый объект класса **Форма**. Основное назначение функции – обеспечить реализацию конструктора объекта **Форма**. Данный класс выступает в качестве базового для производных классов, где этот метод перекрывается одноименной функцией для создания прикладных объектов, обеспечивая тем самым полиморфизм.

Создаваемая форма производных классов сразу после создания отображается на экране в немодальном окне.

Пример

```
var F: БланкНакладная;  
F = БланкНакладная.СоздатьВидимым;
```

Описание

Экспорт(ИмяФайла :Строка; Формат : [Форма.ТипыЭкспорта](#) [Фрейм : [ФреймШаблона](#) [; Опции : [Форма.ОпцииЭкспорта](#)[]]) :Логическое;

Export(FileName:String; Format : [Form.ExportTypes](#) [; Frame : [TemplateFrame](#) [; Options: [Form.ExportOptions](#)[]]) : Logical;

Аргументы

ИмяФайла - имя файла, в который будет записана форма. Если полный путь не указан, то файл будет записан в подкаталоге текущей сессии

Формат - тип формата экспорта, константа (ToBMP, ToJPG, ToGIF, ToRTF, ToHTML, ToTPL), определенная в типе [Форма.ТипыЭкспорта](#);

Внимание! При экспорте шаблона (задана константа ToTPL) все опции экспорта будут отменены, кроме одной ПоказыватьМастер/ShowWizard.

Фрейм - необязательный параметр, фрейм шаблона, который нужно экспортировать. Значение параметра по умолчанию равно RootFrame. Если данный параметр задан, то он будет корневым в сохраняемом шаблоне;

Опции - необязательный параметр, задает массив опций экспорта, более подробно см. в теме ["Константы опций экспорта"](#).

Назначение

При удачном завершении экспорта функция возвращает True, независимо от значения опции **ПоказатьМастер**. Если значение опции ПоказатьМастер=True и пользователь прервал работу мастера, нажав кнопку **Отмена**, функция возвращает False.

Внимание. В случае выбора ПоказатьМастер=False и при неудачном завершении операции экспорта возникает исключительная ситуация (exception).

Пример

```
-- экспорт формы в HTML, с возможностью выбора пользователем файла.
-- Возвращает полное имя (т.е. имя и путь) сформированного файла
func ЭкспортВХТМЛ synonym ExportToHTML(aForm :Form; aShowChooseFileDialog :Logical;
    aName :String; aWordCompatible :Logical = False;
    aExcelCompatible :Logical = False; aFrame :TemplateFrame = nil) :String;


var Opt[] :Form.ExportOptions;

if aShowChooseFileDialog then
    if (ChooseFile(aName, 'Экспорт в HTML', 'Документы в формате HTML
        (*.htm,*.html)|*.htm,*.html') = cmOk) then
        УстановитьПутьКФайлуДляЭкспорта(ExtractFilePath(aName));
    end;
end;
if not ExistFolder(ВернутьПутьКФайлуДляЭкспорта) then
    CreateFolder(ВернутьПутьКФайлуДляЭкспорта);
end;
Opt = ПолучитьОпцииЭкспорта(False, RTFColor, RTFAsPrinted,
aWordCompatible, aExcelCompatible, False, True);
AForm.Export(AName, Form.ToHtml, aFrame, Opt);
Result = aName;
end;
```

Класс *ФормаБланка / BlankForm*, производный от класса *Форма*, является базовым классом для производных классов, которые обеспечивают работу с пользовательскими бланками Студии.

Класс *ФормаБланка* наследует все свойства родительских классов [Объект](#), [ПользовательскийОбъект](#) и [Форма](#).

Непосредственно в классе *ФормаБланка / BlankForm* определены следующие свойства и перечислимые типы:

-  [Поле Объекты / Objects](#)
-  [Функция СоздатьВидимым / CreateVisible](#)
-  [Функция ВыполнитьБланк / ExecuteBlank](#)
-  [Функция ПоказатьФормуДоп / ShowFormEx](#)

-  [Поле ТипКласса / ClassType](#)
-  [Поля ОпцияБланка / BlankOption](#)
-  [Функция ВыполнитьДоп / ExecuteEx](#)
-  [Функция ПоказатьДоп / ShowEx](#)
-  [Процедура РедакторЗаписать / EditorPost](#)
-  [Процедура РедакторОтменить / EditorCancel](#)

-  [Перечислимый тип ОпцииБланка / BlankOptions](#)

Перечислимый тип **ОпцииБланка/BlankOptions**, определенный в классе **ФормаБланка** предназначен для задания опций, характеризующих работу бланка в режиме открытия, закрытия, сохранения или удаления записей.

В данном типе определены следующие константы:

- **AutoCloseOnDelete** - закрывать бланк автоматически при удалении;
- **AutoOpenOnNew** - открывать бланк автоматически в новом окне при дублировании записи;
- **AutoPostOrCancel** - сохранять запись бланка-редактора.

Константы, определенные в данном типе, используются в свойствах [ОпцияБланка/BlankOption](#) класса **ФормаБланка**.

Описание

Объекты [Индекс: Целое] : [ФормаБланка](#);
Objects [Index: Integer] : [BlankForm](#);

Аргументы

Индекс - номер формы, который может изменяться в пределах от 1 до числа форм бланков.

Назначение

Данное свойство позволяет получить доступ к любой из созданных к данному моменту форм по номеру.
Поле доступно только на чтение.

Пример

См. [пример работы с объектами класса Форма](#).

Описание

ТипКласса : Класс [ФормаБланка](#);
ClassType: Class [BlankForm](#);

Назначение

Возвращает указатель на класс **ФормаБланка** (именно на класс, а не на объект класса). Указатель позволяет создавать объекты на базе существующих (клонировать, т.е. создавать их точную копию), проверять, принадлежат ли объекты одному классу или разным, а также определять наследственные связи в иерархии классов.

Поле доступно только на чтение.

Пример

```
func F1(Ob1:BlankForm; Ob2:BlankForm):Logical;  
  var L : Logical;  
  if Ob1.ClassType = Ob2.ClassType then  
    Message("Объекты принадлежат одному классу");  
    L = TRUE;  
  else  
    Message("Объекты принадлежат разным классам");  
    L = FALSE;  
  fi;  
  return L;  
end;
```


Описание

ОпцияБланка[ОпцияIndex: [ФормаБланка.ОпцииБланка](#)] :Логическое;
BlankOption[Option : [BlankOptions.BlankOption](#)] :Logical;

Аргументы

Опция - одна из predetermined опций бланка, заданная в перечислимом типе [ОпцииБланка](#).

Назначение

Если свойство равно False, то запись может остаться в состоянии "Edit" после закрытия бланка редактора.

Пример

```
proc Button1OnClick(Sender :Button);  
...  
  BlankOption[AutoPostOrCancel] = True;  
end;
```

Описание

```
РедакторЗаписать;  
EditorPost;
```

Назначение

Записывает в картотеку открытый в данный момент в бланке-редакторе документ. Процедура генерирует события шаблона [OnVerify](#) и [OnPost](#).

Пример

```
proc КнСохранить (Sender :String);  
    EditorPost;  
    Close(cmOk);  
end;
```

Описание

```
РедакторОтменить;  
EditorCancel;
```

Назначение

Отменяет изменения, сделанные в документе, загруженном в бланк-редактор. Документ возвращается в состояние, которое он имел до начала редактирования или после последней его записи ([EditorPost](#)).

Пример

```
Пример  
proc КнОтменить (Sender :String);  
    EditorCancel;  
    Close(cmCancel);  
end;
```

Описание

```
ВыполнитьБланк({перем Запись : Запись}): Целое;  
ExecuteBlank({var Record : Record): Integer;
```

Аргументы

Запись - запись для загрузки в бланк-редактор.

Назначение

Создает экземпляр бланка-редактора и открывает его в модальном окне, загружая указанную запись. Если запись не задана, бланк открывается с новым пустым документом. Возвращает одну из констант кмдВерно (cmdOk) или кмдОтказ (cmdCancel), в зависимости от действий пользователя.

После возврата управления из функции параметр **Запись** содержит ссылку на запись, которая была фактически отредактирована бланком. Это может быть не та запись, которая передавалась в момент вызова, если такое предусмотрено логикой работы бланка.

Этот метод наследуется производными классами.

Внимание. Метод **ВыполнитьБланк** в ближайших релизах планируется удалить, поэтому настоятельно рекомендуется везде перейти на вызов метода [ПоказатьФормуДоп](#), который перекрывает возможности этого метода.

Пример

```
proc кнВыбор_ПриНажатии (Sender :Object);  
    Валюты.Курс.ExecuteBlank(Query.Current);  
end;
```

Описание

```
ВыполнитьДоп({var Запись : Запись}: Целое;  
ExecuteEx    ({var Record : Record}: Integer;
```

Аргументы

Запись - документ для загрузки в бланк-редактор.

Назначение

Внимание. Метод **ВыполнитьДоп** в ближайших релизах планируется удалить, поэтому настоятельно рекомендуется везде перейти на вызов метода [ПоказатьДоп](#), который перекрывает возможности этого метода.

Открывает существующий экземпляр бланка-редактора в модальном окне, загружая в него указанную запись. Если запись не задана, бланк открывается с новой пустой записью. Предполагается, что бланк до вызова этого метода невидим, то есть создан, но не показан ни с помощью свойства **Visible**, установленного в значение TRUE, ни с помощью другого, более раннего вызова **ExecuteEx**. Если бланк уже был открыт на экране в немодальном режиме, он не становится модальным в результате вызова **ExecuteEx**.

Возвращает одну из констант кмдВерно (cmdOk) или кмдОтказ (cmdCancel) в зависимости от действий пользователя.

После возврата управления из функции параметр **Запись** содержит ссылку на запись, которая была фактически отредактирована бланком (это может быть уже не та запись, которая передавалась в момент вызова, если такое предусмотрено логикой работы бланка).

Этот метод наследуется объектами производных классов.

Пример

```
proc кнВыбор_ПриНажатии (Sender :Object);  
    self. ExecuteEx (Query.Current);  
end;
```

Описание

```
ПоказатьДоп ({var Запись : Запись}; {СтильОкна :Окно.СтилиОкна}) :Целое;  
ShowEx({var Record : Record}; {WindowState :WindowStyles}) :Integer;
```

Аргументы

Запись - запись для загрузки в бланк-редактор;

СтильОкна - предопределенная константа перечислимого типа [СтилиОкон](#), которая определяет способ открытия бланка-редактора. Если параметр не задан, то он считается равным AutoDetect.

Назначение

Открывает существующий экземпляр бланка-редактора в окне заданного стиля, загружая в него указанную запись. Если она не задана, бланк открывается с новой пустой записью.

Возвращает одну из констант кмдВерно (cmdOk) или кмдОтказ (cmdCancel) в зависимости от действий пользователя.

После возврата управления из функции параметр **Запись** содержит ссылку на запись, которая была фактически отредактирована бланком. Это может быть уже не та запись, которая передавалась в момент вызова, если такое предусмотрено логикой работы бланка.

Этот метод наследуется объектами производных классов.

Описание

```
ПоказатьФормуДоп({var Запись : Запись}; {СтильОкна :Окно.СтилиОкна}) :Целое;  
ShowFormEx({var Record : Record}; {WindowStyle :WindowStyles}) :Integer;
```

Аргументы

Запись - запись для загрузки в бланк-редактор;

СтильОкна - предопределенная константа перечислимого типа [СтилиОкон](#), которая определяет способ открытия бланк-редактора. Если параметр не задан, то он считается равным AutoDetect.

Назначение

Создает экземпляр бланка-редактора и открывает его в окне заданного стиля, загружая указанную запись. Если запись не задана, бланк открывается с новой пустой записью. Возвращает одну из констант кмдВерно (cmdOk) или кмдОтказ (cmdCancel) в зависимости от действий пользователя.

После возврата управления из функции параметр **Запись** содержит ссылку на запись, которая была фактически отредактирована бланком. Это может быть уже не та запись, которая передавалась в момент вызова, если такое предусмотрено логикой работы бланка.

Этот метод наследуется производными классами.

Описание

СоздатьВидимым : [ФормаБланка](#);
CreateVisible : [BlankForm](#);

Назначение

Функция создает новый объект класса **ФормаБланка**. Основное назначение функции - обеспечить реализацию конструктора объектов **ФормаБланка**. Данный класс выступает в качестве базового для производных классов, где этот метод перекрывается одноименной функцией для создания прикладных объектов, обеспечивая тем самым полиморфизм.

Создаваемая форма производных классов сразу после создания отображается на экране в немодальном окне.

Внимание. Метод **СоздатьВидимым** в ближайших релизах планируется удалить, поэтому настоятельно рекомендуется везде перейти на вызов метода [ПоказатьФормуДоп](#), который перекрывает возможности этого метода.















Пример

```
var F: БланкНакладная; -- производный от ФормаБланка класс  
F = БланкНакладная.СоздатьВидимым;
```


Класс *ФормаКартотеки* / *CardForm*, производный от класса *Форма*, используется в качестве базового класса для производных классов, которые обеспечивают работу с пользовательскими картотеками.

Класс *ФормаКартотеки* наследует все свойства родительских классов [Объект](#), [ПользовательскийОбъект](#) и [Форма](#).

Непосредственно в классе *ФормаКартотеки* определены следующие свойства:

-  [Поле Объекты / Objects](#)
-  [Функция СоздатьВидимым / CreateVisible](#)
-  [Функция ВыполнитьКартотеку / ExecuteCard](#)
-  [Функция ПоказатьФормуДоп / ShowFormEx](#)
-  [Поле ТипКласса / ClassType](#)
-  [Поле Запрос / Query](#)
-  [Поле Журнал / Journal](#)
-  [Поле Картотека / CardFile](#)
-  [Поле ПоказыватьДерево / ShowTree](#)
-  [Поле ПоказыватьПодтаблицу / ShowSubtable](#)
-  [Поле ПоказыватьШаблон / ShowTemplate](#)
-  [Поле МожноСкрыватьШаблон / CanHideTemplate](#)
-  [Функция ВыполнитьДоп / ExecuteEx](#)
-  [Функция ПоказатьДоп / ShowEx](#)

Поле Журнал / Journal

Описание

Журнал : [Журнал](#);
Journal : [Journal](#);

Назначение

Возвращает объект класса [Журнал](#), если картотека открыта как журнал, иначе - nil.

Поле доступно только на чтение.

Поле Записи / Records

Описание

Записи :Класс[] Запись;
Records :Class[] Record;

Назначение

Свойство позволяет получить сведения о наличии классов записей картотеки и возвращает массив классов записей картотеки.

Поле доступно только на чтение.

Информация о классах записей картотеки хранится в #prj файле.

Описание

Запрос : [Запрос](#);
Query : [Query](#);

Назначение

Поле позволяет получить доступ к объекту **Запрос**, ассоциированному с данной формой картотеки. Манипулируя свойствами запроса, можно управлять поведением формы картотеки и получать сведения о ее текущем состоянии, в частности, определять текущую (выделенную) запись.

Следует иметь в виду, что в иерархической картотеке в режиме отображения иерархии (то есть, когда одновременно в таблице картотеки выводится содержимое только одной группы, одного уровня иерархии) запрос содержит только документы открытой группы. Поэтому попытка присвоить свойству [Текущий](#) класса **Запрос** некоторый документ, не входящий в открытую группу, генерирует исключение. Иными словами, в запросе иерархической картотеки можно *программно* менять текущий документ только в пределах открытой группы. Если необходимо переместиться из одной группы в другую, следует присваивать свойство **Текущий** класса **Запрос** полю [Картотека](#). При этом система автоматически закрывает старый запрос и построит новый.

Внимание! Свойства класса **Запрос** – **Записи, Фильтр, Упорядочивание** – имеются также и в классе **Картотека**. Изменение этих свойств в запросе действуют лишь до первого изменения содержащихся в картотеке записей. После каждого изменения картотеки система инициализирует вышеуказанные свойства поля **Запрос** соответствующими значениями одноименных свойств поля **Картотека**. Таким образом, если необходимо перманентно изменить какое-либо из свойств, это следует делать через поле **Картотека**, а не **Запрос**.

Это свойство наследуется производными классами.

Пример

```
-- обработчик нажатия кнопки в картотечной форме
proc кнУдалить_ПриНажатии (Sender :String);
    Query.Current.Delete;
    -- аналогичная явная запись была бы
    -- self.Query.Current.Delete;
    -- где, self - это ссылка на текущий экземпляр формы картотеки
end;
```

Описание

Картотека : [ФормаКартотеки](#);
CardFile : [CardForm](#);

Назначение

Поле позволяет получить доступ к объекту [ФормаКартотеки](#), ассоциированному с данной формой картотеки. Манипулируя свойствами картотеки, можно управлять поведением картотеки и получать сведения о ее текущем состоянии.

Внимание! Свойства класса **Картотека** - **Записи, Фильтр, Упорядочивание** – имеются также и в классе **Запрос**, объект которого ассоциирован с формой наравне с картотекой. Изменение этих свойств в запросе действуют лишь до первого изменения содержащихся в картотеке записей. После каждого изменения картотеки система инициализирует вышеуказанные свойства поля [Запрос](#) соответствующими значениями одноименных свойств описываемого поля **Картотека**. Таким образом, если необходимо перманентно изменить какое-либо из свойств, это следует делать через поле **Картотека**, а не **Запрос**.

Это свойство наследуется производными классами.

Пример

```
-- обработчик нажатия кнопки в картотечной форме
proc кнУдалить_ПриНажатии (Sender :String);
var i :Integer;
  if (Cardfile.SelectedCount > 0) then
    -- Cardfile - это свойство текущего экземпляра формы картотеки,
    -- ссылку на который можно опускать, однако она подразумевается:
    -- self.Cardfile.SelectedCount
    if ВопрДаОтказ('Удалить выделенные записи?') = cmOk then
      BeginTransaction([Справочники.КурсыВалют]);
      try
        for i = 1..Cardfile.SelectedCount do
          Hint('Удаление курса валюты на '
            + Str(Cardfile.Selected[i].Дата) + '...');
          (Cardfile.Selected[i].Delete;
        end;
        Hint('Обновление картотеки...');
        EndTransaction;
      except
        AbortTransaction;
        raise;
      end;
    end;
  else
    if ВопрДаОтказ('Удалить величину курса от '
      + Str(Query.Current.Дата) + '?') = cmOk then
      Query.Current.Delete;
      РазрешитьИлиЗапретитьКнопки;
    end;
  end;
end;
```

Описание

МожноСкрыватьШаблон :Логическое;
CanHideTemplate :Logical;

Назначение

Во время сессии свойство позволяет с помощью мыши управлять шириной и видимостью шаблонной части картотеки, и, даже сделать шаблонную часть картотеки невидимой. Для этого значение свойства, имеющего логический тип, должно быть установлено равным True.

Настройка видимости шаблонной части картотеки сохраняется в файле с расширением bro.

Поле доступно на чтение и запись.

Пример

```
-- процедура настройки видимости
proc ButtonClick(B:Button);
    CanHideTemplate = True;
    -- разрешается управлять видимостью
end;
```

Класс Объект : [ПользовательскийОбъект](#) : [Форма](#) : [ФормаКартотеки](#);
Поле Объекты / Objects

Описание

Объекты [Индекс: Целое] : [ФормаКартотеки](#);
Objects [Index: Integer] : [CardForm](#);

Аргументы

Индекс - номер формы, который может изменяться в пределах от 1 до числа форм картотек.

Назначение

Данное свойство позволяет получить доступ к любой из созданных к данному моменту форм картотек по номеру. Поле доступно только на чтение.

Пример

См. [пример работы с объектами класса Форма](#).

Описание

ПоказыватьДерево : Логическое;
ShowTree : Logical;

Назначение

Данное поле позволяет узнать, включен ли режим просмотра дерева в иерархической картотеке (свойство [ПоказыватьИерархию](#) равно TRUE), а также включать/отключать его.

Пример

```
-- процедура включает/отключает показ дерева
proc ButtonClick(B:Button);
  if Картотека.ПоказыватьИерархию then
    ПоказыватьДерево = NOT ПоказыватьДерево;
  end;
end;
```


Описание

ПоказыватьПодтаблицу : Логическое;
ShowSubtable : Logical;

Назначение

Данное поле позволяет узнать, включен ли режим просмотра подтаблицы в окне картотеки, а также включать/отключать его.

Пример

```
-- процедура показывает подтаблицу  
proc ButtonClick(B:Button);  
    ПоказыватьПодтаблицу = TRUE;  
end;
```

Описание

ПоказыватьШаблон :Логическое;
ShowTemplate :Logical;

Назначение

Свойство логического типа позволяет показать шаблон картотеки (свойство равно True) или его скрыть (свойство равно False).

Пример

```
-- процедура показывает шаблон
proc ButtonClick(B:Button);
    ПоказыватьШаблон = True;
end;
```

Описание

ТипКласса : Класс [ФормаКартотеки](#);
ClassType: Class [CardForm](#);

Назначение

Возвращает указатель на класс **ФормаКартотеки** (именно на класс, а не на объект класса). Указатель позволяет создавать объекты на базе существующих (клонировать, т.е. создавать их точную копию), проверять, принадлежат ли объекты одному классу или разным, а также определять наследственные связи в иерархии классов.

Поле доступно только на чтение.

Пример

```
func F1(Ob1:CardForm; Ob2:CardForm):Logical;  
  var L : Logical;  
  if Ob1.ClassType = Ob2.ClassType then  
    Message("Объекты принадлежат одному классу");  
    L = TRUE;  
  else  
    Message("Объекты принадлежат разным классам");  
    L = FALSE;  
  fi;  
  return L;  
end;
```

Описание

ВыполнитьДоп[(перем Запись : [Запись](#))] : Целое; [Фильтр:Строка]] : Целое;
ExecuteEx[(var Record : [Record](#))] : Integer; [Filter:String]] : Integer;

Аргументы

Запись - переменная, определяющая, какую запись следует выделить в картотеке при ее открытии. Через эту же переменную будет возвращен выбранная пользователем запись;

Фильтр - отбора документов, отображаемых в окне картотеки.

Назначение

Открывает окно существующего экземпляра картотеки с учетом фильтра, выделяя в ней указанную запись, если первый необязательный аргумент был задан.

Функция возвращает константу кмдВерно (cmdOk) или кмдОтказ (cmdCancel), описывающую выполненное пользователем действие. Если это cmdOk и при этом первый аргумент был задан, то он содержит выбранную пользователем запись.

Предполагается, что форма до вызова этого метода невидима, то есть она создана, но не показана ни с помощью свойства **Visible**, установленного в значение TRUE, ни с помощью другого, более раннего вызова **ExecuteEx**.

Этот метод наследуется объектами производных прикладных классов.

Внимание. Метод **ВыполнитьДоп** в ближайших релизах планируется удалить, поэтому настоятельно рекомендуется везде перейти на вызов метода [ПоказатьДоп](#), который перекрывает возможности этого метода.

Пример

```
func ВыборТМЦИзСправочника(Тов:DocS.Справочники.Ресурс): DocS.Справочники.Ресурс;  
var РезультатВыбора:Integer;  
var СтарыйТов:DocS.Справочники.Ресурс;  
var Crd:Справочники.Ресурс.Ред_Ресурс;  
  
СтарыйТов=Тов;  
Crd = Справочники.Ресурс.Ред_Ресурс.Create;  
РезультатВыбора=Crd.ExecuteEx (Тов, '');  
return if(РезультатВыбора=cmOK,Тов,СтарыйТов);  
end;
```

Описание

```
ВыполнитьКартотеку({var Запись : Запись}; {Фильтр :Строка} :Целое;  
ExecuteCard({var Record : Record}; {Filter :String}) :Integer;
```

Аргументы

Запись - переменная, определяющая, какую следует выделить в картотеке при ее открытии. Через эту же переменную будет возвращен выбранная пользователем запись;

Фильтр - условие отбора записей, отображаемых в окне картотеки.

Назначение

Создает экземпляр картотеки и открывает ее с учетом фильтра в модальном окне, выделяя в нем указанную запись (если первый необязательный аргумент используется по назначению). Если функция возвращает предопределенную константу **cmOK** и первый аргумент был задан, то он содержит выбранную пользователем запись.

Этот метод наследуется производными классами.

Внимание. Метод **ВыполнитьКартотеку** в ближайших релизах планируется удалить, поэтому настоятельно рекомендуется везде перейти на вызов метода [ПоказатьФормуДоп](#), который перекрывает возможности этого метода.

Пример

```
Товар : Пример.КарточкаТовара;  
прос кнВыбор_ПриНажатии (Sender :Object);  
    Справочник.Товары.ExecuteCard(Товар);  
end;
```

Функция ПоказатьДоп / ShowEx

Описание

```
ПоказатьДоп({var Запись :Запись}; {Фильтр :Строка}; {СтильОкна :Окно.СтилиОкна}) :Целое;  
ShowEx ({var Record :Record}; {Filter :String}; {WindowStyle :WindowStyles}) :Integer;
```

Аргументы

Запись - переменная, определяющая, какую запись следует выделить в картотеке при ее открытии. Через эту же переменную будет возвращен выбранная пользователем запись;

Фильтр - условие отбора записей, отображаемых в окне картотеки;

СтильОкна - предопределенная константа перечислимого типа [СтилиОкон](#), которая определяет способ открытия картотеки. Если параметр не задан, то он считается равным AutoDetect.

Назначение

Открывает окно существующего экземпляра картотеки с учетом фильтра, выделяя в ней указанную запись, если первый необязательный аргумент был задан.

Функция возвращает константу кмдВерно (cmdOk) или кмдОтказ (cmdCancel), описывающую выполненное пользователем действие. Если это cmdOk и при этом первый аргумент был задан, то он содержит выбранную пользователем запись.

Этот метод наследуется объектами производных прикладных классов.

Описание

ПоказатьФормуДоп ({var Запись : [Запись](#)}; {Фильтр :Строка};
{СтильОкна :Окно.СтилиОкна}) :Целое;

ShowFormEx ({var Record : [Record](#)}; {Filter :String}; {WindowState :WindowStyles}) :Integer;

Аргументы

Запись - переменная, определяющая, какую запись следует выделить в картотеке при ее открытии. Через эту же переменную будет возвращена выбранная пользователем запись;

Фильтр - условие отбора записей, отображаемых в окне картотеки;

СтильОкна - предопределенная константа перечислимого типа [СтилиОкон](#), которая определяет способ открытия картотеки. Если параметр не задан, то он считается равным AutoDetect.

Назначение

Создает экземпляр картотеки и открывает ее с учетом фильтра в окне, тип которого указан в третьем аргументе. Если первый необязательный аргумент задан, то при открытии заданная запись выделяется.

Если функция возвращает предопределенную константу **cmOK**, и при этом был задан первый аргумент, то он содержит выбранную пользователем запись.

Этот метод наследуется производными классами.

Пример

```
if Справочники.Товары.ShowFormEx(Doc, '', Window.ModalWindow) = cmOk then
  if Template.CurrentObject = Edit1 then
    Edit1.SelText = '{ ' + Str(Doc.Название) + ' }';
  elseif Template.CurrentEdit <> nil then
    Template.CurrentEdit.SelText = '{ ' + Str(Doc.Название) + ' }';
  end;
end;
```

Описание

СоздатьВидимым : [ФормаКартотеки](#);
CreateVisible : [CardForm](#);

Назначение

Функция создает новый объект класса **ФормаКартотеки**. Основное назначение функции – обеспечить реализацию конструктора объектов **ФормаКартотеки**. Данный класс выступает в качестве базового класса для производных классов, где этот метод перекрывается одноименной функцией для создания прикладных объектов, обеспечивая тем самым полиморфизм.

Создаваемая форма производных классов сразу после создания отображается на экране в немодальном окне.

Внимание. Метод **СоздатьВидимым** в ближайших релизах планируется удалить, поэтому настоятельно рекомендуется везде перейти на вызов метода [ПоказатьФормуДоп](#), который перекрывает возможности этого метода.












Пример

```
proc КартТовары(Sender:Button);  
    var B: CardForm;  
    B=Справочники.Товары.Ред_Товары.CreateVisible;  
end;
```


Класс *ФормаОтчета / ReportForm*, производный от класса *Форма*, является базовым классом для производных классов, которые обеспечивают пользовательский интерфейс для работы со встроенными отчетами Студии.

Класс *ФормаОтчета* наследует все свойства родительских классов [Объект](#), [ПользовательскийОбъект](#) и [Форма](#).

Непосредственно в классе *ФормаОтчета* определены следующие свойства:

-  [Поле Объекты / Objects](#)
-  [Функция СоздатьВидимым / CreateVisible](#)
-  [Поле ТипКласса / ClassType](#)
-  [Поле Отчет / Report](#)
-  [График / Chart](#)
-  [Поле АвтоПостроение / AutoBuild](#)
-  [Процедура СформироватьШаблон / FormTemplate](#)
-  [Процедура Обновить / Update](#)
-  [Процедура Уточнить / Precise](#)
-  [Процедура Синхронизировать / SyncReport](#)
-  [Процедура СинхронизироватьРасш / SyncReportEx](#)

Перечислимый тип **ДействияОбновления / UpdateActions**, введенный в классе **ФормаОтчета / ReportForm**, позволяет определить объект шаблона, который будет обновляться при формировании шаблона отчета.

В данном типе определены следующие константы:

- **ОбновлениеКлетки / UpdateCell** - обновление клетки;
- **ОбновлениеСтолбца / UpdateColumn** - обновление столбца.

Константы, определенные в данном типе, используются в обработчике [ПриОбновленииОтчета](#) класса **Шаблон**.

Перечислимый тип **ДействияРаскрытия / ExpandActions**, определенный в классе **ФормаОтчета / ReportForm**, используется для интерактивного раскрытия группы или уточнения в отчетах.

В данном типе определены следующие константы:

- **РаскрытиеИерархии / ExpandHierarchy** - раскрытие иерархии;
- **РаскрытиеУточнения / ExpandPrecision** - раскрытие уточнения;
- **ПолноеРаскрытиеИерархии / FullExpandHierarchy** - полное раскрытие иерархии в параметрических отчетах для заполнения пользовательских показателей.

Константы, определенные в данном типе, используются в событии [ПриПодготовкеОтчета](#) класса **Шаблон**.

Описание

АвтоПостроение :Логическое;
AutoBuild :Logical;

Назначение

Данное поле позволяет определить, следует ли автоматически строить отчет сразу же после открытия формы отчета на экране. Если поле имеет значение TRUE, отчет автоматически строится при открытии (создании) окна. По умолчанию автопостроение включено и имеет смысл только для форм отчетов, имеющих шаблон, то есть для отчетов в форматах *шаблон* или *график*.

Если поле равно FALSE, автоматического построения отчета не происходит, и для этого необходимо явным образом вызвать метод [Построить/Build](#) объекта [Отчет](#).

Если отчет имеет шаблон, то перед построением отчета можно программным способом изменить этот шаблон, и сделанные изменения отразятся на форме отчета после построения.

Пример

```
-- фрагмент кода класса-наследника формы отчета;  
proc BlankOnOpen(Create :Logical);  
  -- отчет должен строиться, только если форма была  
  -- открыта пользователем  
  -- если же форма восстановлена при запуске сессии,  
  -- отчет не должен строиться автоматически  
  Report.AutoBuild = Create;  
end;
```

Поле График / Chart

Описание

График : [ГрафикОтчета](#);

Chart : [ReportChart](#);

Назначение

Поле позволяет получить доступ к объекту класса [ReportChart](#), ассоциированному с данной формой отчета. Манипулируя свойствами графического отчета, можно изменять графическое представление данных на форме.

Это свойство наследуется производными классами. Поле доступно только на чтение.

Пример

```
-- фрагмент кода класса-наследника формы отчета,  
proc ShowTable(x :Integer);  
  if x <= Report.TableCount then  
    Chart.ActiveTable = x;  
  end;  
end;
```

Описание

Объекты [Индекс: Целое] : [ФормаОтчета](#);
Objects [Index: Integer] : [ReportForm](#);

Аргументы

Индекс - номер формы, который может изменяться в пределах от 1 до числа форм отчетов.

Назначение

Данное свойство позволяет получить доступ к любой из созданных к данному моменту форм отчетов по их номерам. Поле доступно только на чтение.

Пример

См. [пример работы с объектами класса Форма](#).

Поле Отчет / Report

Описание

Отчет : [Отчет](#);
Report : [Report](#);

Назначение

Поле позволяет получить доступ к объекту [Отчет](#), ассоциированному с данной формой отчета. Манипулируя свойствами отчета, можно проводить анализ учетных данных в различных разрезах и отображать их на форме.

Это свойство наследуется производными классами. Поле доступно только на чтение.

Пример

```
-- обработчик нажатия кнопки в форме отчета  
прос кнПостроитьОтчет_ПриНажатии (Sender :String);  
    Отчет.Счета = ПолеВводаМаскиСчетов.Текст;  
    Отчет.Построить;  
end;
```

Описание

ТипКласса : Класс [ФормаОтчета](#);
ClassType: Class [ReportForm](#);

Назначение

Возвращает указатель на класс **ФормаОтчета** (именно на класс, а не на объект класса). Указатель позволяет создавать объекты на базе существующих (клонировать, т.е. создавать их точную копию), проверять, принадлежат ли объекты одному классу или разным, а также определять наследственные связи в иерархии классов.

Поле доступно только на чтение.

Пример

```
func F1(Ob1:ReportForm; Ob2:ReportForm):Logical;  
  var L : Logical;  
  if Ob1.ClassType = Ob2.ClassType then  
    Message("Объекты принадлежат одному классу");  
    L = TRUE;  
  else  
    Message("Объекты принадлежат разным классам");  
    L = FALSE;  
  fi;  
  return L;  
end;
```


Описание

Обновить
Update;

Назначение

Обновляет (перерисовывает) форму отчета без перестроения самого отчета. Процедура полезна для тех случаев, когда внутренняя структура отчета редактируется с помощью вспомогательных методов класса [Отчет](#), например, [Удалить](#).

Пример

```
-- фрагмент кода класса-наследника формы отчета;  
-- объект Отчет, связанный с этой формой, строится  
-- при открытии формы  
proc УпроститьОтчет;  
var Num, Result :Integer;  
  Result = Ввод (Num, "Введите номер строки для удаления");  
  if Result = cmOk then  
    -- удаляем указанную строку из 1-ой таблицы  
    Report.CurTable = 1;  
    Report.CurRow = Num;  
    Report.Delete(Report.rdRow);  
    -- обновляем форму, чтобы отразить изменения  
    Update;  
  end;  
end;
```

Описание

```
Синхронизировать(Клетка : КлеткаШаблона);  
SyncReport(Cell : TemplateCell);
```

Аргументы

Клетка - клетка шаблона, по координатам которой требуется синхронизировать отчет.

Назначение

По вызову данной процедуры в объекте класса [Отчет](#), ассоциированному с данной формой отчета, устанавливаются свойства **ТекущаяТаблица**, **ТекущаяСтрока** и **ТекущаяКолонка**, соответствующие положению указанной клетки в секции формы. Таким образом, задаются все 3 координаты, обеспечивающие навигацию по внутренней структуре отчета.

Передаваемая в процедуру клетка должна принадлежать секции, в которую выводятся результаты отчета. Внутренние механизмы работы отчетов на шаблонах обеспечивают хранение для каждой строки секции информации о её связи с некоторой таблицей данных во внутренней структуре объекта [Отчет](#).

Руководствуясь именами строк и столбцов секции, программа также устанавливает соответствие между строками и столбцами секции (с одной стороны) и строками и столбцами таблиц во внутренних данных отчета (с другой стороны).

Принципы проектирования секций на форме отчета рассматриваются в теме ["Разработка шаблона для отчета"](#).

Пример

```
-- фрагмент кода класса-наследника формы отчета  
-- обработчик события перерисовки клетки  
func CellOutput(Cell :TemplateCell; Value :Numeric) :Numeric;  
  -- устанавливаем таблицу, колонку и столбец во внутренней  
  -- структуре данных объекта Report в соответствии с положением  
  -- текущей клетки  
  SyncReport(Cell);  
  -- возвращаем значение показателя из этой ячейки данных отчета  
  return UnitValue(Report.Turn(Roll,1));  
end;
```

Описание

СинхронизироватьРасш(Клетка : [КлеткаШаблона](#); var ИндексПоказателя :Целое; var ОстОбор :
[Отчет.ВидыОстОбор](#) var ДебКре : [Отчет.ВыводОстОбор](#) {; var Знач :Вариант});

SyncReportEx(Cell : [TemplateCell](#)); var IndicatorIndex :Integer; var SumKind : [Report.SumKinds](#); var DebCre :
[Report.SumDebCre](#) {; var ObjVal :Variant});

Аргументы

Клетка - клетка шаблона, по координатам которой требуется синхронизировать отчет;

ИндексПоказателя - номер показателя;

ОстОбор - переменная, которая будет содержать тип показателя, задаваемый одной из
предопределенных констант типа [ВидыОстОбор](#);

ДебКре - переменная, которая будет содержать одну из предопределенных [констант](#), задающих режим
вывода показателей, разделенных по дебету|кредиту или в свернутом виде;

Знач - необязательный параметр типа [Вариант](#)|Variant, который возвращает последнее значение-объект
в цепочке разыменования содержимого разбиения по строке, выводимого отдельным столбцом, при
синхронизации по данному столбцу.

Назначение

Процедура работает аналогично процедуре [Синхронизировать / SyncReport](#), но, при этом возвращает
дополнительную информацию о соответствующей клетке отчета:

- номер показателя, выводимого в клетке;
- вид вывода показателя в клетке (оборот, начальный или конечный остаток);
- режим вывода показателей в клетке (по дебету, кредиту или свернутый).

Замечание. В случае невозможности определить эти значения, они устанавливаются = -1.

Описание

СформироватьШаблон;
FormTemplate;

Назначение

Процедура применяется в классах отчетов, которые предназначены по замыслу разработчика проекта для построения программным способом.

Обычно форма отчета создается автоматически в ответ на запрос пользователя (например, в ходе исполнения команды **Отчеты**) и заполняется результатами построения отчета. При этом результаты фактически вносятся в специальный, динамически генерируемый шаблон, структура и оформление которого задается исходным шаблоном формы отчета.

В некоторых случаях в проекте необходимо построить отчет программным образом. Подобные отчеты требуют ручного кодирования всех операций по их созданию и визуализации, что, однако, придает гибкость в вопросах их настройки и встраивания в пользовательский интерфейс проекта. Для этой цели, например, вызывают метод [СоздатьОтчет](#) класса **Консоль** и для созданного таким образом объекта вызывают метод [Отчет.Построить](#). Такой объект на данной стадии имеет внутреннюю структуру с результатами отчета, но не имеет того самого динамически формируемого шаблона, который эти результаты мог бы отобразить.

Процедура **СформироватьШаблон** позволяет сформировать шаблон, в который заносятся результаты построения отчета. Впоследствии такой шаблон может быть подгружен (см. [LoadTemplate](#) класса **Form**) в любой бланк для демонстрации пользователю.

Файл создаваемого шаблона располагается в папке Reports рабочего каталога информационной базы (его путь можно получить с помощью свойства [ИнфСессии.КаталогЛокальный](#)) и имеет имя, совпадающее с именем класса формы отчета и расширение ".tpl".

Пример

```
ФормаДинОтчета :ReportForm;  
ДатаНачалаПериода :Дата;  
ДатаОкончанияПериода :Дата;  
  
--...  
  
прос СформироватьТотИлиИнойОтчет(Имя :Строка);  
    ФормаДинОтчета = Console.CreateReport(self.ClassProject + '.' + Имя);  
    ФормаДинОтчета.Report.BegDate = ДатаНачалаПериода;  
    ФормаДинОтчета.Report.EndDate = ДатаОкончанияПериода;  
    ФормаДинОтчета.Report.Parameters = "Сумма=руб";  
    ФормаДинОтчета.Report.PreciseBaseClass =  
        self.ClassProject + '.' + Имя + "Уточняющий";  
    ФормаДинОтчета.Report.Build;  
    ФормаДинОтчета. FormTemplate;  
    LoadTemplate(Имя + '.tpl', -1);  
end;  
  
func КлеткаОтчета_ПриНажатии(Cell :TemplateCell; Action :Template.ClickTypes) :Logical;  
    if (Action <> Template.SingleClick) then  
        ФормаДинОтчета.PreciseReport(Cell);  
    end;  
    Result = False;  
end;
```

Описание

```
Уточнить(Клетка : КлеткаШаблона);  
Precise(Cell : TemplateCell);
```

Аргументы

Клетка - клетка шаблона, значение в которой требуется уточнить.

Назначение

Процедура позволяет программным способом инициировать построение уточняющего отчета для показателя в переданной клетке.

Класс уточняющего отчета задается с помощью свойства [БазовыйУточняющего](#) класса **Отчет**.

Если отчет строился по проводкам, то при вызове уточняющего отчета происходит открытие соответствующего журнала на соответствующем месте.

Пример

```
ФормаДинОтчета :ReportForm;  
-- ...  
func КлеткаОтчета_ПриНажатии(Cell :TemplateCell;  
    Action :Template.ClickTypes) :Logical;  
    if (Action <> Template.SingleClick) then  
        ФормаДинОтчета.Precise(Cell);  
    end;  
    Result = False;  
end;
```

Описание

`СоздатьВидимым` : [ФормаОтчета](#);
`CreateVisible` : [ReportForm](#);

Назначение

Функция создает новый объект класса **ФормаОтчета**. Основное назначение функции – обеспечить реализацию конструктора объектов **ФормаОтчета**. Данный класс выступает в качестве базового для производных классов, где этот метод перекрывается одноименной функцией для создания прикладных объектов.

Создаваемая форма производных классов сразу после создания отображается на экране в немодальном окне.

Пример

```
var F: БланкОтчетаПоОборотам; -- производный от ФормаОтчета класс  
F = БланкОтчетаПоОборотам.СоздатьВидимым;
```

Основное назначение класса *ДинамическийОбъект* / *DynamicObject*, унаследованного от класса [UserObject](#) заключается в предоставлении возможности динамического добавления/удаления методов, т.е. во время исполнения программы.

Для манипулирования динамическими методами в классе заведены следующие InClass-методы:



[Функция ДобавитьМетод / AddMethod](#)



[Процедура УдалитьМетод / DeleteMethod](#)

Следует помнить, что все вызовы динамических методов работают через динамическую диспетчеризацию.

Так как динамическое добавление/удаление методов поддерживают только объекты класса *DynamicObject* и его наследников, предусмотрена возможность быстрого (без полного указания разыменований) обращения из динамических методов к полям/методам любого другого заданного программистом объекта.

Для обеспечения этой возможности в классе *DynamicObject* дополнительно введены свойства:



[Функция ДружественныйКласс / FriendlyClass](#)



[Поле ДружественныйОбъект / FriendlyObject](#)

Обращаем Ваше внимание на тот факт, что при компиляции *динамических методов* в общую последовательность поиска идентификатора, встреченного в коде метода, добавляется дополнительный контекст - контекст так называемого "дружественного класса" (см. пункт 5).

1. Текущие операторы With в порядке, обратном порядку появления;
2. Локальные переменные;
3. Список локальных процедур/функций;
4. Список полей и методов текущего класса (для InClass-методов) или объекта (для InObject-методов), т.е. список полей/методов класса или объекта, в котором описан компилируемый метод;
5. Список полей и методов дружественного класса и объекта (если последний задан);
6. Список импортированных классов (директива import);
7. Список общих классов из встроенного проекта Kernel (такие классы, как System, Console).

Описание

ДружественныйОбъект :Объект;
FriendlyObject :Object;

Назначение

InObject-поле **FriendlyObject** определяет непосредственно дружественный объект. Это поле может меняться на протяжении жизни конкретного объекта **DynamicObject**, Единственная проверка, которая производится при установке значения, заключается в том, что устанавливаемый дружественный объект должен быть экземпляром класса, возвращаемого [функцией FriendlyClass](#), или экземпляром наследника дружественного класса.

Внимание. При изменении значения поля **FriendlyObject** перекомпиляции уже скомпилированных динамических методов не происходит.

Поле доступно на чтение и на запись.

Пример

```
--- Класс Friend.cod: ---
Class "";

InClass

    var fFriendlyClassField :Integer := 123;

    proc HelloFriendClass(const aMessage :String = '');
        var vMessage :String;
        if aMessage = '' then
            vMessage = 'Hello, Friend Class!';
        else
            vMessage = aMessage;
        end;
        Message(vMessage);
    end;

InObject
    var fHelloFriendObject :String := 'Hello, Friend Object!';

    func getHelloFriendObject :String;
        Result = fHelloFriendObject;
    end;
End

--- Класс MyFriend.cod: ---

Class inherited Friend "";

InClass
    proc HelloFriendClass1;
        Message('Hello, MyFriend Class!');
    end;

InObject
    func getHelloFriendObject1 :String;
        Result = 'Hello, MyFriend Object!';
    end;
End

--- Класс MyDynamic.cod: ---

Class inherited DynamicObject "";
InClass
```



```

func FriendlyClass :Class;
    Result = Friend;
end;

InObject
proc Test;
    var vSource :String[];

    FriendlyObject = MyFriend.Create;

    vSource[1] = "func MyDynamicTest :Boolean;";
    -- Правильные вызовы полей/методов дружественного объекта
    vSource[2] = "    Message(fFriendlyClassField)";
    vSource[2] = "    HelloFriendClass";
    vSource[2] = "    Message(fHelloFriendObject)";
    vSource[2] = "    Message(getHelloFriendObject)";
    -- Неправильные вызовы полей/методов дружественного объекта -
    -- хотя объект и является экземпляром MyFriend, возможны обращения
    -- только к полям/методам, описанным в классе, который возвращает
    -- FriendlyClass, т.е. в классе Friend:
    vSource[2] = "    HelloFriendClass1";
    vSource[2] = "    Message(getHelloFriendObject1)";
    vSource[3] = "    Result = True";
    vSource[4] = "end;";

    aDynamicObject.Evaluate("MyDynamicTest");
end;
End

```

Описание

```
УдалитьМетод(ИмяМетода :Строка);  
DeleteMethod(MethodName :String);
```

Аргументы

ИмяМетода - имя метода, который требуется удалить.

Назначение

Процедура DeleteMethod удаляет динамический метод по его имени, которое передаётся в первом параметре.

Если метод с указанным именем отсутствует или не является динамическим, генерируется исключение.

Пример

```
proc AddMyDynamicMethod(aDynamicObject :DynamicObject);  
  var vSource :String[];  
  var vMethodInfo :DynamicMethodInfo;  
  
  vSource[1] = "func MyDynamicTest :Boolean;";  
  vSource[2] = "  Message('Вызов MyDynamicTest')";  
  vSource[3] = "  Result = True";  
  vSource[4] = "end;";  
  
  -- Добавление  
  vMethodInfo = aDynamicObject.AddMethod(vSource);  
  
  -- Вызов  
  aDynamicObject.Evaluate("MyDynamicTest");  
  
  -- Удаление  
  aDynamicObject.Delete(vMethodInfo.Name);  
end;
```

Функция **ДобавитьМетод** / **AddMethod**

Описание

ДобавитьМетод(ИсходныйКодМетода :Строка[]) : [ИнфДинамическогоМетода](#);
AddMethod(MethodSourceCode :String[]): [DynamicMethodInfo](#);

Аргументы

ИсходныйКодМетода - список строк, содержащий полный исходный код метода в том виде, в котором методы (т.е. процедуры и функции) описываются в классах языка ТБ.Скрипт.

Назначение

Функция добавляет метод, получая список строк, который передаётся в первый параметр.

Внимание. Добавление полей не допускается.

В момент добавления метода производится только проверка синтаксической корректности кода. Непосредственно генерация исполняемого кода происходит при [первом вызове](#) конкретного динамического метода, поэтому следует быть аккуратным при написании и проверке кода динамических методов, т.к. такие ошибки, как неправильно записанный идентификатор, несовместимые типы в выражении будут обнаружены *только при первом вызове* динамического метода.

Пример

```
proc AddMyDynamicMethod(aDynamicObject :DynamicObject);
    var vSource :String[];
    var vMethodInfo :DynamicMethodInfo;

    vSource[1] = "func MyDynamicTest :Boolean;";
    vSource[2] = "    Message('Вызов MyDynamicTest')";
    vSource[3] = "    Result = True";
    vSource[4] = "end;";

    -- Добавление
    vMethodInfo = aDynamicObject.AddMethod(vSource);

    -- Вызов
    aDynamicObject.Evaluate("MyDynamicTest");

    -- Удаление
    aDynamicObject.Delete(vMethodInfo.Name);
end;
```

Описание

```
ДружественныйКласс :Класс;  
FriendlyClass :Class;
```

Назначение

InClass-функция **FriendlyClass** возвращает класс допустимых дружественных объектов. Возвращаемое ею значение используется при компиляции динамических методов, и именно это значение определяет набор полей/методов, которые будут доступны как "свои собственные" из динамических методов *данного наследника DynamicObject*.

Следует отметить, что обращение к статическим полям/методам дружественного класса идёт через статическую диспетчеризацию. Если же дружественным классом является наследник *DynamicObject* (что допустимо), то обращение к его *динамическим методам* идёт через *динамическую диспетчеризацию*, а обращение к его статическим членам идёт как обычно - статически.

Чтобы определить конкретный дружественный класс, необходимо определить наследника *DynamicObject* и перекрыть функцию FriendlyClass.

Пример

```
--- Класс Friend.cod: ---  
Class "";  
  
InClass  
  
    var fFriendlyClassField :Integer := 123;  
  
    proc HelloFriendClass(const aMessage :String = '');  
        var vMessage :String;  
        if aMessage = '' then  
            vMessage = 'Hello, Friend Class!';  
        else  
            vMessage = aMessage;  
        end;  
        Message(vMessage);  
    end;  
  
InObject  
    var fHelloFriendObject :String := 'Hello, Friend Object!';  
  
    func getHelloFriendObject :String;  
        Result = fHelloFriendObject;  
    end;  
End  
  
--- Класс MyFriend.cod: ---  
  
Class inherited Friend "";  
  
InClass  
    proc HelloFriendClass1;  
        Message('Hello, MyFriend Class!');  
    end;  
  
InObject  
    func getHelloFriendObject1 :String;  
        Result = 'Hello, MyFriend Object!';  
    end;  
End  
  
--- Класс MyDynamic.cod: ---  
  
Class inherited DynamicObject "";
```

```

InClass
    func FriendlyClass :Class;
        Result = Friend;
    end;

InObject
    proc Test;
        var vSource :String[];

        FriendlyObject = MyFriend.Create;

        vSource[1] = "func MyDynamicTest :Boolean;";
        -- Правильные вызовы полей/методов дружественного объекта
        vSource[2] = "    Message(fFriendlyClassField)";
        vSource[2] = "    HelloFriendClass";
        vSource[2] = "    Message(fHelloFriendObject)";
        vSource[2] = "    Message(getHelloFriendObject)";
        -- Неправильные вызовы полей/методов дружественного объекта -
        -- хотя объект и является экземпляром MyFriend, возможны обращения
        -- только к полям/методам, описанным в классе, который возвращает
        -- FriendlyClass, т.е. в классе Friend:
        vSource[2] = "    HelloFriendClass1";
        vSource[2] = "    Message(getHelloFriendObject1)";
        vSource[3] = "    Result = True";
        vSource[4] = "end;";

        aDynamicObject.Evaluate("MyDynamicTest");
    end;
End

```

Класс *Команда*|*Command* позволяют программным способом создать новый класс пользовательских команд, не встроенных в программу.












Класс *Команда* используется в качестве базового для вызова команд и изменения их свойств. Например, это класс можно использовать как базовый для класса команд, которые будут использоваться для открытия пользовательских отчетов. Все команды нужно описать в *.cod файле, а алгоритмы можно разработать таким образом, чтобы при открытии сессии для вновь созданных отчетов добавлялась новая команда или удалялась, если соответствующий отчет был удален. Причем, для наглядности целесообразно сделать имя команды, совпадающим с именем отчета, а также настроить пункт меню и перечислить в нем все команды, вызывающие отчеты.

Класс *Команда* наследует все свойства и методы от родительских классов [Объект](#) и [ПользовательскийОбъект](#).

Доступ к корневой группе команд можно получить через свойство [Команды](#) класса *Консоль/Console*.

Полный список групп и вложенных в них команд можно посмотреть на странице ["Команды"](#) диалога ["Настройка интерфейса"](#).

Непосредственно в классе *Команда* определены следующие свойства и методы:


-  [Поле ТипКласса / ClassType](#)
-  [Поле РодГруппа / ParentGroup](#)
-  [Поле РодИндекс / ParentIndex](#)
-  [Поле ПолноеИмя / FullName](#)
-  [Поле Имя / Name](#)
-  [Поле Название / Caption](#)
-  [Поле Описание / Description](#)
-  [Поле Разрешена / Enabled](#)
-  [Поле Доступна / Accessible](#)
-  [Поле Изображение / Image](#)
-  [Процедура ПриОбработке / OnDispatch](#)

Описание

Доступна : Логическое;

Accessible : Logical;

Назначение

Свойство логического типа позволяет программным способом объявить команду "недоступной", присвоив полю значение False. В этом случае пользователь не сможет ее вызвать, т.е. она запрещается и на ее иконке слева от названия команды в меню рисуется символ "замок" .

Поле доступно на чтение и запись, что позволяет сделать команду как доступной, так и недоступной, не используя права доступа. Данное свойство позволяет заменить многочисленные права доступа к различным инструментальным возможностям.

Причем, если какая-либо команда недоступна, то она всегда запрещена, т.е. свойство [Enabled](#) не имеет смысла.

Пример

```
-- сделать недоступной команду Blanks  
Commands.CommandByName["Kernel.Account.Blanks"].Accessible = False;
```

Класс Объект : [ПользовательскийОбъект](#) : [Команда](#)

Поле Изображение / Image

Описание

Изображение : [Изображение](#);

Image : [Image](#);

Назначение

Свойство позволяет получить или установить картинку для команд, добавленных программно.

Для встроенных команд картинку можно только получить.

Пример

```
ButtonHelp.Image.Assign( Commands.CommandByName[ "Kernel.Help.Help" ].Image );
```


Поле Имя / Name

Описание

Имя : Строка;
Name : String;

Назначение

Свойство позволяет установить или прочитать короткое имя команды, например, Открыть, Копировать. Имя команды - идентификатор, по которому команда идентифицируется в программе. Полное иерархическое имя (Имя.Группы.ИмяКоманды) возвращает свойство [FullName](#).

Поле доступно на чтение и запись.

Пример

```
proc Pl(Comm : Command);  
  var aName : String;  
  aName=Comm.Name;  
  Message("Имя команды = " + aName);  
end;
```

Поле Название / Caption

Описание

Название : Строка;
Caption : String;

Назначение

Свойство позволяет программным способом установить или прочитать название команды, в общем случае оно может совпадать с [именем команды](#). По своему назначению действие этого свойства аналогично полю **Название** на странице ["Команды"](#) диалога ["Настройка интерфейса"](#).

Поле доступно на чтение и запись.

Пример

```
proc Pl(Comm : Command);  
  var aName : String;  
  aName=Comm.Caption;  
  Message("Название команды = " + aName);  
end;
```

Поле Описание / Description

Описание

Описание : Строка;
Description : String;

Назначение

Свойство позволяет установить или прочитать описание к команде, в котором можно, например, указать назначение и способ использования команды.

Поле доступно на чтение и запись.

Пример

```
proc P1(Comm : Command);  
  var aName : String;  
  aName=Comm.Description;  
  Message("Описание команды = " + aName);  
end;
```

Поле ПолноеИмя / FullName

Описание

ПолноеИмя : Строка;
FullName : String;

Назначение

Свойство возвращает полное имя команды (синтаксис обращения к команде: ИмяГруппы.ИмяКоманды), например, Kernel.Text.Edit.InsMode. Сначала через точку перечисляются строковые имена групп, в которые входит команда, начиная с корневой группы. Самой последней в этой цепочке указывается [короткое имя](#) команды (в приведенном примере - это InsMode).

Полный список групп и вложенных в них команд можно посмотреть, например, на странице ["Команды"](#) диалога ["Настройка интерфейса"](#).

Поле доступно только на чтение.

Пример

```
proc P1(Comm : Command);  
  var FName : String;  
  FName=Comm.FullName;  
  Message("Полное имя команды = " + FName);  
  ExecuteCommand(FName);  
end;
```

Поле Разрешена / Enabled

Описание

Разрешена : Логическое;
Enabled : Logical;

Назначение

Свойство позволяет программным способом изменить статус команды, т.е. разрешить (True) или запретить (False) ее использование. Статус команды указан в схемах доступа. Если команда не разрешена, то она отображается серым цветом. Причем, если какая-либо команда [недоступна](#), то данное свойство не имеет смысла.

Поле доступно на чтение и запись

Пример

```
-- запрещено открывать список бланков  
Commands.CommandByName["Kernel.Account.Blanks"].Enabled = False;
```

Поле РодГруппа / ParentGroup

Описание

РодГруппа : ГруппаКоманд;
ParentGroup : CommandGroup;

Назначение

Возвращает ссылку на группу, к которой принадлежит команда.

Поле доступно только на чтение.

Пример

```
proc Pl(Comm : Command);  
  var aName : String;  
  var N : Integer;  
  --определяем имя группы, в которую  
  -- вложена команда, заданная ссылкой Comm  
  aName=Comm.ParentGroup.Name;  
  Message("Имя группы = " + aName);  
  -- определяем количество команд в группе  
  N=Comm.ParentGroup.CommandsCount;  
end;
```

Поле РодИндекс / ParentIndex

Описание

РодИндекс : Целое;
ParentIndex : Integer;

Назначение

Поле предназначено для чтения порядкового номера текущей команды. Номера команд изменяются от 1 до количества команд, вложенных в родительскую группу, доступ к которой определяется свойством [РодГруппа / ParentGroup](#).

Поле доступно только на чтение.

Пример

```
var N : Integer;  
N=Commands.CommandByName["Kernel.Account.Cardfiles"].ParentIndex;  
....  
proc Pl(Comm : Command);  
  var aName : String;  
  aName=Comm.ParentGroup.Name;  
  Message("Имя группы = " + aName);  
  Message("Индекс = " + Стр(Comm.ParentIndex));  
  -- определяем количество команд в группе  
  N=Comm.ParentGroup.CommandsCount;  
end;
```

Поле ТипКласса / ClassType

Описание

ТипКласса : Класс [Команда](#);
ClassType : Class [Command](#);

Назначение

Возвращает указатель на класс **Команда** (именно на класс, а не на объект класса). Указатель позволяет создавать объекты на базе существующих (клонировать, т.е. создавать их точную копию), проверять, принадлежат ли объекты одному классу или разным, а также определять наследственные связи в иерархии классов.

Поле доступно только на чтение.

Пример

```
func F1(Ob1:Command; Ob2:Command):Logical;  
  var L : Logical;  
  if Ob1.ClassType = Ob2.ClassType then  
    Message("Объекты принадлежат одному классу");  
    L = TRUE;  
  else  
    Message("Объекты принадлежат разным классам");  
    L = FALSE;  
  fi;  
  return L;  
end;
```


Описание

ПриОбработке;
OnDispatch;

Назначение

Процедура вызывается в момент срабатывания команды. Ее можно "[перекрыть](#)", вставив требуемый код, который будет выполняться в момент срабатывания команды.

Класс *ОбщийДоступ / CommonAccess* позволяет реализовать системные права доступа через программный интерфейс. Все общие права зарегистрированы в нем как простые Inclass поля (описывающие свойства класса в целом), доступные как на чтение, так и на запись. Класс *ОбщийДоступ* позволяет устанавливать и проверять права доступа для объектов, которые в данный момент существуют среди экземпляров этого класса.

Свойства данного класса определяют полномочия пользователей при доступе к общим ресурсам, например, к записям, бланкам, картотекам, отчетам и др. Кроме этого, установку прав доступа ([ролей пользователя](#)) можно выполнять в интерактивном режиме в поле **Right** записи **Kernel.Settings.Role** на закладке "Данные" системного [диалога "Записи"](#). При наличии нескольких прав у одного пользователя начальное значение права устанавливается при запуске сессии в диалоге ["Права пользователя"](#). Каждому пользователю или группе назначается определенная роль, которая связана с конкретной схемой доступа и набором прав доступа.

Класс *ОбщийДоступ* является наследником родительских классов [Объект](#), [ПользовательскийОбъект](#) и [ПраваДоступа](#) с сохранением всех их свойств и методов.

Непосредственно в классе *ОбщийДоступ / CommonAccess* определены следующие новые свойства и переопределены наследуемые свойства:

- [Поле Объекты / Objects](#)
- [Поле AccountFilter](#)
- [Поле CloseJurs](#)
- [Поле CorrectRefs](#)
- [Поле DisignTemplate](#)
- [Поле FastDelete](#)
- [Поле FindReferences](#)
- [Поле FullRefresh](#)
- [Поле HelpContext](#)
- [Поле Interface](#)
- [Поле LoginSchema](#)
- [Поле OpenBlankList](#)
- [Поле OpenCardList](#)
- [Поле OpenRecordsView](#)
- [Поле OpenReports](#)
- [Поле SetupInterface](#)
- [Поле ShowHiddenParams](#)
- [Поле ViewConstraints](#)
- [Поле ViewRecHistory](#)
- [Поле ТипКласса / ClassType](#)

Поле AccountFilter

Описание

AccountFilter :Строка;

Назначение

Поле предназначено для запроса или установки системных прав доступа к счетам. Поле доступно на чтение и запись. В свойстве задается условие для задания неиспользуемых счетов, которые по умолчанию не отображаются в диалоге выбора счетов. Условие автоматически объединяется по "&" с любым условием, переданным в диалог [выбора счетов](#). Поэтому условие не должно содержать план счетов. Само условие будет иметь вид типа:

~(01|02|03!)

Начальное значение права доступа Kernel.AccountFilter задается в режиме исполнения проекта в поле [Right](#) записи **Kernel.Settings.Role** на закладке "Данные" системного [диалога "Записи"](#). При наличии нескольких прав у одного пользователя начальное значение права устанавливается при запуске сессии в диалоге ["Права пользователя"](#).

Пример

```
proc УстановитьМаскуИсключенныхСчетов(aMask :String);
  if (Trim(aMask) = '') then
    CommonAccess.AccountFilter = '';
  elseif ПравильноеУсловиеОтбораСчетов(aMask) then
    CommonAccess.AccountFilter = '~(' + aMask + ')';
  end;
end;
```

Поле CloseJurs

Описание

CloseJurs : Логическое;

Назначение

Пользователю, обладающему данным правом, разрешается закрывать период. Поле доступно на чтение и запись.

Начальное значение права Kernel.CloseJurs задается в режиме исполнения проекта в поле [Right](#) записи **Kernel.Settings.Role** на закладке "Данные" системного [диалога "Записи"](#). При наличии нескольких прав у одного пользователя начальное значение права устанавливается при запуске сессии в диалоге ["Права пользователя"](#).

Пример

```
proc F1(Sender:Button);
  if CommonAccess.CloseJurs then
    Message("Разрешается закрывать период.");
  end;
end;
```

Поле **CorrectRefs**

Описание

CorrectRefs :Логическое;

Назначение

Данное свойство разрешает корректировать ссылки на запись. Поле доступно на чтение и запись.

Начальное значение права доступа Kernel.CorrectRefs задается в интерактивном режиме в поле [Right](#) записи **Kernel.Settings.Role** на закладке "Данные" системного [диалога "Записи"](#). При наличии нескольких прав у одного пользователя начальное значение права устанавливается при запуске сессии в диалоге ["Права пользователя"](#).

Пример

```
proc F1(Sender:Button);
  if CommonAccess.CorrectRefs then
    Message("Разрешается корректировать ссылки на запись.");
  end;
end;
```

Поле **DesignTemplate**

Описание

DesignTemplate : Логическое;

Назначение

Пользователю, обладающему данным правом, разрешено в режиме сессии переходить в дизайн режим.

Поле доступно на чтение и запись.

Начальное значение права Kernel.DesignTemplate задается в интерактивном режиме в поле [Right](#) записи **Kernel.Settings.Role** на закладке "Данные" системного [диалога "Записи"](#). При наличии нескольких прав у одного пользователя начальное значение права устанавливается при запуске сессии в диалоге ["Права пользователя"](#).

Пример

```
proc F1(Sender:Button);
  if CommonAccess.DesignTemplate then
    Message("Разрешен переход в дизайн режим.");
  fi;
end;
```

Поле FastDelete

Описание

FastDelete : Логическое;

Назначение

Данное свойство разрешает пользователю удалять записи без проверки ссылочной целостности (значение поля равно True) или запрещает обычному пользователю отключать контроль целостности (значение поля равно False).

Поле доступно на чтение и запись.

Начальное значение права Kernel.FastDelete задается в интерактивном режиме в поле [Right](#) записи **Kernel.Settings.Role** на закладке "Данные" системного [диалога "Записи"](#). При наличии нескольких прав у одного пользователя начальное значение права устанавливается при запуске сессии в диалоге ["Права пользователя"](#).

Пример

```
proc F1(Sender:Button);
  if CommonAccess.FastDelete then
    Message("Разрешается удалять записи без "+
      "проверки ссылочной целостности.");
  end;
end;
```

Описание

FindReferences : Логическое;

Назначение

Данное свойство доступно на чтение и запись и отвечает за доступность сервиса поиска ссылок. По умолчанию значение поле равно True, т.е. сервис доступен.

Начальное значение права доступа Kernel.FindReferences задается в интерактивном режиме в поле [Right](#) записи **Kernel.Settings.Role** на закладке "Данные" системного [диалога "Записи"](#). При наличии нескольких прав у одного пользователя начальное значение права устанавливается при запуске сессии в диалоге ["Права пользователя"](#).

Пример

```
proc F1(Sender:Button);
  if CommonAccess.FindReferences then
    Message("Разрешен поиск ссылок.");
  end;
end;
```


Описание

FullRefresh : Логическое;

Назначение

Пользователю, обладающему данным правом, разрешена полная обработка расчетной базы, т.е. он может воспользоваться командой **Обработать все** (Alt+F9) для вызова диалога ["Настройка параметров обработки"](#).

Поле доступно на чтение и запись.

Начальное значение права **Kernel.FullRefresh** задается в интерактивном режиме в поле **Right** записи [Kernel.Settings.Role](#) на закладке "Данные" системного диалога ["Записи"](#). При наличии нескольких прав у одного пользователя начальное значение права устанавливается при запуске сессии в диалоге ["Права пользователя"](#).

Пример

```
proc F1(Sender:Button);  
  if CommonAccess.FullRefresh then  
    Message("Разрешена полная обработка расчетной базы.");  
  end;  
end;
```

Поле HelpContext

Описание

HelpContext :Строка;

Назначение

Данное свойство позволяет открыть тему помощи в том случае, когда она отсутствует у текущего окна. При этом по нажатию клавиши **F1** открывается содержание помощи или общая тема для бланков, заданная по умолчанию в [схеме доступа](#).

Пример

```
proc F1(Sender:Button);  
  if CommonAccess.HelpContext="" then  
    CommonAccess.HelpContext=Управление.Содержание;  
  end;  
end;
```

Поле Interface

Описание

Interface :Строка;

Назначение

Данное свойство позволяет запросить или установить имя интерфейсной схемы. Если программным способом это свойство не установлено, то загружается интерфейсная схемы, указанная [в схеме доступа](#).

Причем, если [свойство](#) CommonAccess.SetupInterface=true, то пользователю разрешается выполнять настройку пользовательского интерфейса.

Пример

```
proc F1(Sender:Button);
  if CommonAccess.Interface= " " then
    CommonAccess.Interface="MyTbCorp";
  end;
end;
```

Поле LoginSchema

Описание

LoginSchema :Строка;

Назначение

Данное свойство позволяет запросить имя схемы доступа, заданное по умолчанию или установить другое имя. Если программным способом это свойство не установлено, то загружается схема доступа, заданная на закладке "Данные" системного [диалога "Записи"](#).

Поле доступно на чтение и запись.

Пример

```
proc Fl(Sender:Button);
  if CommonAccess.LoginSchema="" then
    CommonAccess.LoginSchema="Менеджер";
  end;
  Message("Схема доступа = " + CommonAccess.LoginSchema);
end;
```

Описание

`OpenBlankList` : Логическое;

Назначение

Пользователю, обладающему данным правом, разрешается открывать диалоговое окно, содержащее [список бланков](#) для выбора нужного бланка.

Поле доступно на чтение и запись.

Начальное значение права `Kernel.OpenBlankList` задается в интерактивном режиме в поле [Right](#) записи **Kernel.Settings.Role** на закладке "Данные" системного [диалога "Записи"](#). При наличии нескольких прав у одного пользователя начальное значение права устанавливается при запуске сессии в диалоге ["Права пользователя"](#).

Пример

```
proc F1(Sender:Button);
  if CommonAccess.OpenBlankList then
    -- Разрешен выбор бланков
    ExecuteCommand("Kernel.Account.Blanks");
  end;
end;
```

Поле OpenCardList

Описание

OpenCardList : Логическое;

Назначение

Свойство позволяет открыть [список картотек](#) и выбрать нужную картотеку, если задано значение true.

Поле доступно на чтение и запись.

Начальное значение права Kernel.OpenCardList задается в интерактивном режиме в поле [Right](#) записи **Kernel.Settings.Role** на закладке "Данные" системного [диалога "Записи"](#). При наличии нескольких прав у одного пользователя начальное значение права устанавливается при запуске сессии в диалоге ["Права пользователя"](#).

Пример

```
proc F1(Sender:Button);
  if CommonAccess.OpenCardList then
    Message("Разрешен доступ к картотекам.");
  fi;
end;
```

Поле **OpenRecordsView**

Описание

OpenRecordsView : Логическое;

Назначение

Пользователю, обладающему данным правом, разрешается открывать системное окно для [просмотра записей](#).

Поле доступно на чтение и запись.

Начальное значение права **Kernel.OpenRecordsView** задается в интерактивном режиме в поле [Right](#) записи **Kernel.Settings.Role** на закладке "Данные" системного [диалога "Записи"](#). При наличии нескольких прав у одного пользователя начальное значение права устанавливается при запуске сессии в диалоге ["Права пользователя"](#).

Пример

```
proc F1(Sender:Button);
  if CommonAccess.OpenRecordsView then
    Message("Разрешен просмотр записей.");
  end;
end;
```

Поле OpenReports

Описание

OpenReports : Логическое;

Назначение

Пользователь, обладающий данным правом, может открыть диалог ["Внутренние отчеты"](#).

Поле доступно на чтение и запись.

Начальное значение права Kernel.OpenReports задается в интерактивном режиме в поле [Right](#) записи **Kernel.Settings.Role** на закладке "Данные" системного [диалога "Записи"](#). При наличии нескольких прав у одного пользователя начальное значение права устанавливается при запуске сессии в диалоге ["Права пользователя"](#).

Пример

```
proc F1(Sender:Button);
  if CommonAccess.OpenReports then
    Message("Разрешен вызов диалога внутренних отчетов.");
  fi;
end;
```


Поле SetupInterface

Описание

SetupInterface : Логическое;

Назначение

Право **SetupInterface** (true) позволяет пользователю настраивать меню, команды горячие клавиши и инструментальную панель программы с помощью диалога ["Настройка интерфейса"](#).

Поле доступно на чтение и запись.

Начальное значение права Kernel.SetupInterface можно задать в режиме исполнения проекта в поле [Right](#) записи **Kernel.Settings.Role** на закладке "Данные" системного [диалога "Записи"](#). При наличии нескольких прав у одного пользователя начальное значение права устанавливается при запуске сессии в диалоге ["Права пользователя"](#).

Пример

```
proc F1(Sender:Button);
  if CommonAccess.SetupInterface then
    Message("Разрешена настройка интерфейса.");
  end;
end;
```

Описание

ShowHiddenParams : Логическое;

Назначение

Данное свойство (право доступа) позволяет динамически включать/выключать показ [скрытых параметров](#) во всех диалогах, в которых выдается список параметров счетов, а именно, в диалогах отчетов, в ввода проводки, а также в диалогах выбора параметров счетов, ввода фильтра проводок, вызываемых через программный интерфейс. Если значение поля равно True, то разрешен показ скрытых параметров. По умолчанию показ разрешен.

Пример

```
proc F1(Sender:Button);  
  if CommonAccess.ShowHiddenParams then  
    Message("Разрешен показ скрытых параметров.");  
  end;  
end;
```

Поле ViewConstraints

Описание

ViewConstraints : Логическое;

Назначение

Свойство используется при отладке проекта, когда настройщику требуется знать, какие фильтры ограничивают его права. Данное свойство позволяет (True) или запрещает (False) настройщику просмотреть [постоянный фильтр](#), установленный на класс записей.

Поле доступно на чтение и запись.

Начальное значение права Kernel.ViewConstraints можно задать в режиме исполнения проекта в поле [Right](#) записи **Kernel.Settings.Role** на закладке "Данные" системного [диалога "Записи"](#). При наличии нескольких прав у одного пользователя начальное значение права устанавливается при запуске сессии в диалоге ["Права пользователя"](#).

Пример

```
proc Fl(Sender:Button);
  if CommonAccess.ViewConstraints then
    Message("Разрешен просмотр ограничений.");
  end;
end;
```

Описание

ViewRecHistory :Логическое;

Назначение

Данное свойство доступно на чтение и запись и позволяет узнать или установить право пользователя на просмотр истории изменения записей. Если значение свойства равно True, разрешен показ истории изменений записи, иначе - запрещен (False).

Начальное значение права Kernel.ViewRecHistory можно задать в режиме исполнения проекта в поле [Right](#) записи **Kernel.Settings.Role** на закладке "Данные" системного [диалога "Записи"](#). При наличии нескольких прав у одного пользователя начальное значение права устанавливается при запуске сессии в диалоге ["Права пользователя"](#).

Пример

```
proc F1(Sender:Button);  
  if CommonAccess.ViewRecHistory then  
    Message("Разрешен просмотр истории изменения записей.");  
  end;  
end;
```

Поле Объекты / Objects

Описание

Объекты [Индекс: Целое] : [ОбщийДоступ](#);
Objects[Index: Integer] : [CommonAccess](#);

Аргументы

Индекс - номер права доступа, который может изменяться в пределах от 1 до общего прав доступа.

Назначение

Данное свойство позволяет получить по заданному номеру доступ к любому объекту класса ОбщийДоступ, созданному к данному моменту времени.

Поле доступно только на чтение.

Описание

ТипКласса : Класс [ОбщийДоступ](#);
ClassType: Class [CommonAccess](#);

Назначение

Возвращает указатель на класс **ОбщийДоступ** (именно на класс, а не на объект класса). Указатель позволяет создавать объекты на базе существующих (клонировать, т.е. создавать их точную копию), проверять, принадлежат ли объекты одному классу или разным, а также определять наследственные связи в иерархии классов.










Поле доступно только на чтение.

Пример

```
func F1(Ob1:CommonAccess; Ob2:CommonAccess):Logical;  
  var L : Logical;  
  if Ob1.ClassType = Ob2.ClassType then  
    Message("Объекты принадлежат одному классу");  
    L = TRUE;  
  else  
    Message("Объекты принадлежат разным классам");  
    L = FALSE;  
  fi;  
  return L;  
end;
```

Класс *ПользовательскийОбъект / UserObject*, производный от класса [Объект](#), используется в качестве базового класса для производных классов, которые обеспечивают работу с пользовательскими объектами: классами и электронными формами (бланками и картотеками) Студии.

Непосредственно в классе *ПользовательскийОбъект* определены следующие свойства:

-  [Поле ЧислоОбъектов / ObjectsCount](#)
-  [Поле Объекты / Objects](#)
-  [Функция Создать / Create](#)
-  [Процедура СохранитьКласс / SaveClass](#)
-  [Процедура ЗагрузитьКласс / LoadClass](#)
-  [Поле ТипКласса / ClassType](#)
-  [Процедура ПриУничтожении / OnDestroy](#)
-  [Процедура СохранитьОбъект / SaveObject](#)
-  [Процедура ЗагрузитьОбъект / LoadObject](#)

Вышеперечисленные методы и поля наследуются и частично перекрываются производными встроенными классами Студии - [ФормаБланка](#), [ФормаКартотеки](#) и [ФормаОтчета](#) - а от них, в свою очередь, - пользовательскими классами форм бланков, картотек и отчетов, написанных с помощью ТБ.Скрипт. Непосредственно от класса *ПользовательскийОбъект* наследуются "простые" пользовательские классы, имеющие лишь алгоритмическую часть (cod-файлы).

Описание

Объекты[Индекс: Целое] : [ПользовательскийОбъект](#);
Objects [Index: Integer] : [UserObject](#);

Аргументы

Индекс - номер объекта, который может изменяться в пределах от 1 до [числа объектов](#).

Назначение

Данное свойство позволяет по заданному номеру получить доступ к любому из существующих объектов соответствующего класса ([ПользовательскийОбъект](#) или производного). Поле доступно только на чтение.

Пример

```
См. пример работы с пользовательскими объектами.  
-- Эта функция проверяет не открыт ли уже этот документ  
-- в другом окне, и если да, то возвращает указатель на него  
func ПроверитьЧтоУжеОткрыт(SDoc:Document):Движение.ОстДен;  
var j,k,m:Integer;  
var B,BRet:Движение.ОстДен;  
m=0;  
k=Движение.ОстДен.ЧислоОбъектов;  
for j=1..k do  
  B=Движение.ОстДен.Объекты[j];  
  if B.Document=SDoc:  
    m=m+1;  
    if m=1:  
      BRet=B;  
    else  
      Break;  
    fi;  
  fi;  
end;  
return if(m>1,BRet,NIL);  
end;
```


Описание

ТипКласса : Класс [ПользовательскийОбъект](#);
ClassType: Class [UserObject](#);

Назначение

Возвращает указатель на класс [ПользовательскийОбъект](#) (именно на класс, а не на объект класса). Указатель позволяет создавать объекты на базе существующих (клонировать), проверять, принадлежат ли объекты одному классу или разным, а также определять наследственные связи в иерархии классов. Поле доступно только на чтение.

Пример

```
func F1(Init:UserObject; Base:UserObject):Logical;  
  var L : Logical;  
  if Base.ClassType = Init.ClassType then  
    Message(Init.ClassName+" = "+Base.ClassName);  
    L = TRUE;  
  else  
    Message(Init.ClassName+" <> "+Base.ClassName);  
    L = FALSE;  
  fi;  
  return L;  
end;
```

Описание

ЧислоОбъектов : Целое;
ObjectsCount : Integer;

Назначение

Поле содержит количество объектов соответствующего класса (например, количество форм определенного бланка-редактора, открытых пользователем). Поле доступно только на чтение.

Пример

```
-- Эта функция проверяет не открыт ли уже этот документ
-- в другом окне, и если да, то возвращает указатель на него
func ПроверитьЧтоУжеОткрыт(SDoc:Document):Движение.ОстДен;
var j,k,m:Integer;
var B,BRet:Движение.ОстДен;
m=0;
k=Движение.ОстДен.ЧислоОбъектов;
for j=1..k do
  B=Движение.ОстДен.Объекты[j];
  if B.Document=SDoc:
    m=m+1;
    if m=1:
      BRet=B;
    else
      Break;
    fi;
  fi;
end;
return if(m>1,BRet,NIL);
end;
```

Описание

ПриУничтожении;
OnDestroy;

Назначение

Данный метод автоматически вызывается ядром при уничтожении объекта. Перекрыв его в любом пользовательском классе, можно вставить туда требуемый код, который будет выполняться в момент уничтожения объекта данного класса. Непосредственный вызов данного метода не уничтожает объект.

См. также функцию [Создать / Create](#).

Пример

```
Class "ЭтотКласс";

InClass Public

-- некий запрос, общий для всех объектов данного класса
var Q:Query;

-- перекрываем конструктор класса
func Создать :ЭтотКласс;
    -- создаём объект с помощью ядра
    Result = inherited Создать;
    -- если это первый объект этого класса,
    -- создаём требуемый запрос
    if ЧислоОбъектов = 1 then
        Q = Query.Create([Документы]);
        Q.Select;
    end;
end;

InObject

-- отслеживаем удаление объектов
proc ПриУничтожении;
    -- если удаляется последний объект
    -- закрываем запрос
    if ЧислоОбъектов = 1 then
        Q = nil;
    end;
end;

End
```

Описание

```
ЗагрузитьКласс(ИмяФайла : Строка);  
LoadClass(FileName : String);
```

Аргументы

ИмяФайла - название файла, из которого требуется прочитать состояние свойств класса. Если расширение не указано, используется расширение *.dbt. Имя может включать полный путь или же только имя файла. В последнем случае файл ищется в папке:

Здесь:

Имя сервера - сетевое имя компьютера, на котором размещается сервер. Имя сервера отображается в иерархии серверов в окне ["Администрирование"](#). Если информационная система работает в локальном, однопользовательском режиме, то в иерархии серверов присутствует только один сервер - "Мой компьютер";
Имя ИБ - имя информационной базы.

Назначение

Загружает все свойства класса из указанного файла. Данный метод может быть полезен для восстановления общих настроек конкретных классов бланков. По умолчанию система загружает перед началом сессии все пользовательские классы (аналог серии вызовов **ЗагрузитьКласс**), автоматически сохраняемые в момент завершения сессии (аналог серии вызовов **СохранитьКласс**).

Пример

```
ФормаНакладной.ЗагрузитьКласс( "НастройкиДляНакладных.dbt" );
```

Описание

```
ЗагрузитьОбъект(ИмяФайла: Строка);  
LoadObject(FileName: String);
```

Аргументы

ИмяФайла - полное или краткое файла, из которого считываются свойства объекта. Если расширение не указано, используется расширение *.dbt. Полное имя включает полный путь к папке и собственно имя файла, краткое - только имя файла. В последнем случае файл ищется в папке:

Здесь:

Имя сервера - сетевое имя компьютера, на котором размещается сервер. Имя сервера отображается в иерархии серверов в окне ["Администрирование"](#). Если информационная система работает в локальном, однопользовательском режиме, то в иерархии серверов присутствует только один сервер - "Мой компьютер";

Имя ИБ - имя информационной базы.

Назначение

Загружает все свойства объекта из указанного файла. Данный метод может быть полезен для восстановления настроек конкретных объектов (экземпляров классов), например форм бланков и картотек. По умолчанию система загружает перед началом сессии все пользовательские объекты (аналог серии вызовов **ЗагрузитьОбъект**), автоматически сохраненные в момент завершения сессии.

Пример

```
ФормаНакладной.ЗагрузитьОбъект( "ТекущаяНакладная.dbt " );
```

Описание

```
СохранитьКласс(ИмяФайла: Строка);  
SaveClass(FileName: String);
```

Аргументы

ИмяФайла - название файла, в котором необходимо сохранить состояние свойств класса. Если расширение не указано, используется расширение *.dbt. Имя может включать полный путь или же только имя файла. В последнем случае файл записывается в папке:

Здесь:

Имя сервера - сетевое имя компьютера, на котором размещается сервер. Имя сервера отображается в иерархии серверов в окне ["Администрирование"](#). Если информационная система работает в локальном, однопользовательском режиме, то в иерархии серверов присутствует только один сервер - "Мой компьютер";
Имя ИБ - имя информационной базы.

Назначение

Записывает все свойства класса в указанный файл. Данный метод может быть полезен для сохранения общих настроек конкретных классов бланков. По умолчанию система сохраняет все пользовательские классы, активные на момент завершения сессии (аналог множественного вызова **СохранитьКласс**), то есть, например, все бланки, открытые в программе в момент ее закрытия, и загружает эти классы перед началом сессии (аналог множественного вызова **ЗагрузитьКласс**).

Пример

```
ФормаНакладной.СохранитьКласс( "НастройкиДляНакладных.dbt" );
```

Описание

```
СохранитьОбъект(ИмяФайла: Строка);  
SaveObject(FileName: String);
```

Аргументы

ИмяФайла - название файла, в котором необходимо сохранить состояние свойств конкретного объекта (экземпляра класса). Если расширение не указано, используется расширение DBT. Имя может включать полный путь или же только имя файла. В последнем случае файл записывается в папке:

Здесь:

Имя сервера - сетевое имя компьютера, на котором размещается сервер. Имя сервера отображается в иерархии серверов в окне ["Администрирование"](#). Если информационная система работает в локальном, однопользовательском режиме, то в иерархии серверов присутствует только один сервер - "Мой компьютер";
Имя ИБ - имя информационной базы.

Назначение

Записывает все свойства объекта в указанный файл. Данный метод может быть полезен для сохранения настроек конкретных объектов (форм бланков, картотек). По умолчанию система сохраняет все пользовательские объекты, активные на момент завершения сессии, например, открытые формы бланков и картотек, что аналогично серии вызовов процедуры **СохранитьОбъект**.

Пример

```
ФормаНакладной.СохранитьОбъект( "ТекущаяНакладная.dbt " );
```

Описание

Создать : [ПользовательскийОбъект](#);

Create : [UserObject](#);

Назначение

Функция создает новый объект класса **ПользовательскийОбъект**. Основное назначение функции – обеспечить реализацию конструктора объектов в базовом классе **UserObject**. В производных классах этот метод будет перекрываться одноименной функцией для создания прикладных объектов, обеспечивая тем самым полиморфизм.

См. также процедуру [ПриУничтожении / OnDestroy](#).

Пример

```
Class "ЭтотКласс";

InClass Public

-- некий запрос, общий для всех объектов данного класса
var Q:Query;

-- перекрываем конструктор класса
func Создать :ЭтотКласс;
-- создаём объект с помощью ядра
Result = inherited Создать;
-- если это первый объект этого класса,
-- создаём требуемый запрос
if ЧислоОбъектов = 1 then
    Q = Query.Create([Документы]);
    Q.Select;
end;
end;

InObject

-- отслеживаем удаление объектов
proc ПриУничтожении;
-- если удаляется последний объект
-- закрываем запрос
if ЧислоОбъектов = 1 then
    Q = nil;
end;
end;

End
```


Класс *ПраваДоступа* / *AccessRights*, производный от класса *ПользовательскийОбъект*, используется в качестве базового класса для производного класса [ОбщийДоступ / CommonAccess](#), который позволяет реализовать системные права доступа через программный интерфейс.

Класс *ПраваДоступа* является наследником родительских классов [Объект](#) и [ПользовательскийОбъект](#) с сохранением всех их свойств и методов.

Непосредственно в классе *ПраваДоступа* определены следующие свойства:

- [Поле Объекты / Objects](#)
- [Поле ТипКласса / ClassType](#)

Описание

Объекты [Индекс: Целое] : [ПраваДоступа](#);
Objects[Index: Integer] : [AccessRights](#);

Аргументы

Индекс - номер права доступа, который может изменяться в пределах от 1 до общего количества объектов класса ПраваДоступа.

Назначение

Данное свойство позволяет получить по заданному номеру доступ к любому объекту класса ПраваДоступа, созданному к данному моменту времени.

Поле доступно только на чтение.

Поле ТипКласса / **ClassType**

Описание

ТипКласса : Класс [ПраваДоступа](#);
ClassType: Class [AccessRights](#);

Назначение

Возвращает указатель на класс **ПраваДоступа** (именно на класс, а не на объект класса). Указатель позволяет создавать объекты на базе существующих (клонировать, т.е. создавать их точную копию), проверять, принадлежат ли объекты одному классу или разным, а также определять наследственные связи в иерархии классов.

Поле доступно только на чтение.

Пример

```
func F1(Ob1:AccessRights; Ob2:AccessRights):Logical;  
  var L : Logical;  
  if Ob1.ClassType = Ob2.ClassType then  
    Message("Объекты принадлежат одному классу");  
    L = TRUE;  
  else  
    Message("Объекты принадлежат разным классам");  
    L = FALSE;  
  end;  
  return L;  
end;
```

Функция "RGB"

Описание функции:

```
func RGB(Red :Integer; Green :Integer; Blue :Integer) :Integer;
```

Функция позволяет получить необходимый цвет на основании заданных уровней красного, синего и зеленого цвета. Возвращает целое число, определяющее цвет в 24-битном формате.

Через параметры Red, Green и Blue передаются уровни (целые числа) красного, зеленого и синего цвета соответственно. Так как любой цвет можно определить как сочетание красного, зеленого и синего цвета.

Функция "Загл"

Описание функции:

```
func Загл synonym СБольшойБуквы(aИсходнаяСтрока :String) :String;
```

Функция возвращает строку aИсходнаяСтрока, в которой первая буква заменена на заглавную.

Функция "Падеж"

Описание функции:

```
func Падеж synonym ВернутьСловоформу(  
    aОписаниеСловоформ :String;  
    aПадежЧисло :Integer = фсИменительныйЕдин;  
    aСБольшойБуквы :Logical = True;  
    aОписаниеСловоформПоУмолчанию :String = ''  
) :String;
```

Функция выделяет из строки в формате описания словоформ (строка с разделителями, в которой перечислены различные падежные формы одного существительного), передаваемой в параметре aОписаниеСловоформ, конкретную [словоформу](#), заданную падежом и числом (единственным или множественным). Для указания требуемой словоформы в качестве параметра aПадежЧисло можно использовать следующие константы:

```
фсИменительныйЕдин  
фсИменительныйМнож  
фсРодительныйЕдин  
фсРодительныйМнож  
фсДательныйЕдин  
фсДательныйМнож  
фсВинительныйЕдин  
фсВинительныйМнож  
фсТворительныйЕдин  
фсТворительныйМнож  
фсПредложныйЕдин  
фсПредложныйМнож
```

Параметр aСБольшойБуквы указывает, следует ли делать первую букву возвращаемой словоформы заглавной (по умолчанию - следует). А в параметре aОписаниеСловоформПоУмолчанию в функцию можно передать дополнительное, "страховочное" описание словоформ, которое будет использовано, если в основном описании нет словоформы для требуемых падежа и числа.

Пример использования функции "Падеж" в формулах реквизитов:

```
Падеж(%1.Менеджер.СловоФормаДолжность, фсДательныйЕдин)
```

где %1 - ссылка на конкретный реквизит (например, "ПроцессОткуда").

Функция "ПересчитатьВ" (ConvertTo)

Описание функции:

```
func ПересчитатьВ synonym ConvertTo(  
    const Сумма :Numeric;  
    const Откуда, Куда :Справочники.ЕдиницаИзмерения;  
    const ДатаПересчета :Date = Today;  
    const ОкруглятьДоТочностиЕдИзм :Logical = False;  
    const НольЕслиНеСовместимы :Logical = False;  
    const СообщениеЕслиНесовместимыЕдИзм, СообщениеЕслиКоефНоль :String = '';  
    const ФиксированныйКурс :Numeric = 0  
): Numeric
```

Функция пересчитывает сумму, переданную в параметре Сумма, из одной единицы измерения или валюты (параметр Откуда) в другую (параметр Куда).

В параметре ДатаПересчета указывается дата, на которую берется курс. Если же дата не указана, то берется текущая дата.

Параметр ОкруглятьДоТочностиЕдИзм указывает, округлять ли результат до точности единицы измерения. По умолчанию округление не производится.

Если параметр НольЕслиНеСовместимы равен True, то при несовместимости валют или единиц измерения (например, при попытке пересчета из рублей в штуки) функция вернет 0, а не выдаст ошибку. Если же (как и по умолчанию) данный параметр равен False, то при несовместимости единиц измерений будет возбуждена ошибка с текстом, передаваемом в параметре "СообщениеЕслиНесовместимыЕдИзм".

Аналогично, в параметре "СообщениеЕслиКоефНоль" передается текст сообщения об ошибке, выдаваемой, если курс между заданными валютами или единицами измерения - нулевой.

В параметре ФиксированныйКурс можно указать фиксированный курс пересчета. Если же данный параметр опущен, то курс, по которому будет идти пересчет, берется из значения курсов валют или отношений единиц измерения на заданную дату.

Функция "ПоследнееЗначение / LastValue"

Описание функции:

```
ПоследнееЗначение synonym LastValue(  
    аИмяПараметра      :String;  
    аКонечнаяДата      :Date;  
    аУсловиеНаПараметры :String;  
    аУсловиеНаСчета     :String = 'УправленческийПлан:Об'  
) :Variant
```

Функция возвращает значение указанного параметра из дебетовой полупроводки последней проводки на указанную дату окончания и удовлетворяющую заданным условиям.

Параметры функции:

аИмяПараметра - имя параметра проводки, для которого требуется найти значение.

аКонечнаяДата - дата, на которое требуется найти последнее значение.

аУсловиеНаПараметры - условие отбора проводок по параметрам.

аУсловиеНаСчета - условие отбора проводок по счетам. Для управленческих проводок данный параметр можно опустить, для бухгалтерских проводок - следует указать.

Результатом функции является произвольное значение.

Функция Право

Функция предназначена для получения значения права доступа с именем, переданном через единственный параметр.

Описание функции:

```
func Право(аНаименованиеПрав :String) :Variant;
```

В функции Право в качестве параметра передается одна из нижеприведенных строк, рядом указан тип возвращаемого значения и его смысл.

системные права

'Kernel.LoginSchema'	строка	название действующей схемы доступа
'Kernel.SetupInterface'	логическое	разрешено настраивать пользовательский интерфейс
'Kernel.DesignTemplate'	логическое	разрешено переключать бланк в дизайн-режим
'Kernel.CorrectRefs'	логическое	разрешено проводить коррекцию ссылок
'Kernel.FullRefresh'	логическое	разрешено проводить полную обработку журналов и справоников
'Kernel.OpenBlankList'	логическое	разрешено открывать список бланков
'Kernel.OpenCardList'	логическое	разрешено открывать список картотек
'Kernel.OpenReports'	логическое	разрешено открывать диалог встроенных отчетов
'Kernel.OpenRecordsView'	логическое	разрешено открывать окно просмотра всех записей
'Kernel.OpenAccountView'	логическое	разрешено открывать список счетов
'Kernel.AccountFilter'	строка	фильтр на видимые счета
'Kernel.CloseJurs'	логическое	разрешено закрывать период

Функция возвращает значение указанного права. Тип значения определяется сущностью запрашиваемого права.

Функция "РабочийДеньПоНомеру"

Описание функции:

```
func РабочийДеньПоНомеру(  
    НаДату :Date;  
    Номер  :Integer;  
    Шестидневка :Logical = False  
) :Date;
```

Функция возвращает дату рабочего дня по номеру указанному в параметре Номер (если указано отрицательное число, то поиск происходит от конца месяца). Первый параметр (НаДату) определяет год и месяц в котором определяется дата рабочего дня. Параметр Шестидневка определяет количество рабочих дней в неделе, если параметр равен True то количество рабочих дней шесть, иначе пять;

Функция "РежимУстановлен" / Функция "РежимыУстановлены"

Описание функций:

```
РежимУстановлен(аРежим :Integer) :Logical  
РежимыУстановлены(аРежимы :Integer[]) :Logical
```

Функции проверяют, установлен ли определенный *режим*. Режим - это предопределенная в системе целочисленная константа, указывающая на то, что в данный момент на клиентском месте запущен и работает один из стандартных сервисов. Стандартные константы режимов, предусмотренные в проекте ТБ.Управление, перечислены ниже.

В функцию РежимУстановлен в качестве параметра передается константа режима, который требуется проверить, и она возвращает True, если этот режим в настоящий момент установлен. Аналогично, в функцию РежимыУстановлены - список констант в виде массива, и она возвращает True, если в настоящий момент установлен хотя бы один режим из переданных в списке.

Константы стандартных режимов

Константы стандартных режимов, относящиеся к базовым сервисам, реализованным в проекте ТБ.Управление:

- *рсРасстановкаПартий* = 1. Данный режим установлен, пока работает алгоритм локальной или глобальной расстановки партий.
- *рсСозданиеЗаполнениеНаОсновании* = 2. Данный режим установлен, пока работает [сервис заполнения документа "на основании"](#).
- *рсСверткаСтрок* = 3. Данный режим установлен, пока работает [сервис "Свернуть позиции"](#).
- *рсРасстановкаПартийЛок* = 4. Данный режим установлен, пока работает локальный сервис расстановки партий (одновременно с режимом *рсРасстановкаПартий*).
- *рсРасстановкаПартийГлоб* = 5. Данный режим установлен, пока работает глобальный сервис расстановки партий (одновременно с режимом *рсРасстановкаПартий*).
- *рсИмпортер* = 6. Данный режим установлен, пока работает [сервис "Импорт документа"](#).

Использование режимов в условиях пересчетов

Режим можно использовать при пересчетах в условиях или в формулах, заполняя редактор формул, как показано на рисунке. В примере производится проверка стандартного режима сервиса "Заполнить на основании". Если функция возвращает истину, то условие выполнимо, иначе - нет.

Редактор формул реквизита

Редактор формул реквизита

OK Отмена Помощь

РежимУстановлен(Управление.Константы.РсСозданиеЗаполнениеНаОсновании)

Реквизит

Запись

Проверка

Записи

Поле...

Комментарий:

Функция "ТекущийСубъект"

Описание функции:

func ТекущийСубъект :Управление.Данные.Субъект;

Функция возвращает запись типа Управление.Данные.Субъект, соответствующую текущему пользователю системы. Результат функции может использоваться, например, для того, чтобы записать ссылку на текущего субъекта в поля документа "Сдал" или "Принял";

Функции параметрических отчетов

Эти функции доступны в формулах, используемых при настройке параметрических отчетов.

В ряде этих функций используются [специальные типы и константы](#), определенные для [параметрических отчетов](#).

Перечень функций:

- [Диап](#)
- [Доля](#)
- [Зап](#)
- [Изм](#)
- [ИзОтчета](#)
- [КолКол, НомКол](#)
- [КолСтр, НомСтр](#)
- [НайтиЗапись](#)
- [Огр](#)
- [ОстатокОбязательств](#)
- [Парам, ПарамРасш](#)
- [Пок, ПокИзм, ПокЧаст, Чис, Ед](#)
- [ПокОтчет](#)
- [Поле](#)
- [Распр](#)
- [Сум, Разн, Част](#)
- [ТекущийДебКре](#)
- [ТекущийОстОбор](#)

Функция "Диап"

Описание функции:

func Диап :Integer;

Функция возвращает разность в днях между конечной и начальной датами отчета. Если по какому-то измерению отчета есть разбиение по времени - разность между соседними значениями этого разбиения.

Примеры:

1. Пусть отчет строится для 2005 год, разбит на строки по счетам и не имеет других разбиений.

Функция Диап вернет число дней в 2005 году, т.е. 365.

2. Пусть тот же отчет дополнительно разбит на таблицы по времени с шагом 1 месяц. В таблице, соответствующей январю, функция вернет 31, в следующей - 28 и т.д. В итоговой таблице функция вернет "длину" всего года, т.е. 365. Как правило, функция Диап используется в формулах для расчета оборачиваемости.

Функция Диап не может быть использована в [формуле для удаления строк в отчете](#).

Функция "Доля в процентах"

Эта функция вычисляет отношение измерителя параметрического отчета к его итоговому значению. Итог можно считать по колонке таблицы/по таблице, по колонке отчета/по отчету.

Описание функции:

```
func Доля(  
    const аИмяПоказателя: String;  
    const аОстОбор :      ОстОбор = ОстОборПоУмолчанию;  
    const аДебКре  :      ДебКре  = ДебКреПоУмолчанию;  
    const аВидДоли :      ВидыДоли = ВКолонкеТаблицы  
) :Numeric;
```

Обязательный параметр аИмяПоказателя указывает имя показателя, по которому производится вычисление.

Через параметр аОстОбор передается признак того, вычисляется данная функция в остатках или в оборотах (допустимые значения - константы [НОст](#), [Об](#), [КОст](#)). Если параметр не задан, то функция использует константу [ОстОборПоУмолчанию](#).

Параметр аДебКре определяет, вычисляется ли функция в дебете или кредите (допустимые значения - константы [Деб](#), [Кре](#), [Сверн](#)). Параметр является не обязательным, если он не указан, то передается [ДебКреПоУмолчанию](#)

Параметр аВидДоли определяет относительно, какого итога будет рассчитываться процент (допустимые значения - константы [ВКолонкеТаблицы](#), [ВКолонкеОтчета](#), [ВТаблице](#), [ВОтчете](#)). Если параметр не указан, то по умолчанию используется константа ВКолонкеТаблицы - доля в процентах будет рассчитываться, используя итоговое значение измерителя в колонке таблицы.

ВНИМАНИЕ: Функция доступна только в отчетах по оборотам.

Функция "Зап"

Описание функции:

```
func Зап(const aПризнак :Sign) :Record;
```

Функция возвращает запись, соответствующую признаку, переданному ей в качестве параметра.

Если в качестве параметра функции Зап передать пустое значение, она также вернет пустое значение.

Как правило, эта функция используется в комбинации с функцией [Ед](#), чтобы получить единицу измерения показателя не как значение типа Признак, а как запись.

Пример

Пусть в отчете есть простой показатель Количество типа Измеритель.

Формула

Зап([Ед](#) ("Количество"))

вернет единицу измерения для показателя Количество.

Функция "Изм"

Описание функции:

```
func Изм(  
    const аЗначение :Numeric;  
    const аЕдИзм :Базовый.Справочники.ЕдиницаИзмерения)  
):Unit;
```

Функция создает значение типа Измеритель из двух его составляющих (пары "число - единица измерения"). Это упрощает формулу, когда единица измерения имеется не в виде элемента справочника, а в виде записи. Функция сама находит признак, соответствующей этой записи, и возвращает искомый измеритель.

Функция "ИзОтчета"

Описание функции:

```
func ИзОтчета(  
    const аОтчет :String;  
    const аИмяПоказателя :String;  
    const аВидРазбиенияОсновногоОтчета :RepDimensions;  
    const аНомерРазбиенияОсновногоОтчета :Integer;  
    const аОстОбор :ОстОбор = ОстОборПоУмолчанию;  
    const аДебКре :ДебКре = ДебКреПоУмолчанию;  
    const аНачСмещения, аКонСмещения :Integer[] = [];  
    const аУсловиеНаСчета :String = Отчет.AccountFilter;  
    const аУсловиеНаПараметры :String = Отчет.ParameterFilter;  
    const аИгнорироватьУсловиеОсновногоОтчета :Logical = False  
) :Variant;
```

Данная функция предназначена для вывода в отчет информации, полученной построением вспомогательного отчета. По своим возможностям она похожа на функцию [ПокОтчет](#), однако в отличие от нее, она строит в качестве вспомогательного не тот же самый отчет, но за другой период времени и с другими условиями, а другой отчет, имя которого передается в параметре аОтчет.

В параметре аИмяПоказателя передается имя показателя из вспомогательного отчета. Показатель с таким именем должен существовать.

Вспомогательным отчетом должен быть отчет по оборотам без разбиения на таблицы и колонки, без иерархии и с единственным разбиением на строки. Причем данное разбиение должно встречаться и в основном отчете - в качестве одного из разбиений на строки, колонки или таблицы. Вид разбиения основного отчета, по которому ищется аналогичная строка во вспомогательном отчете, передается в параметре аВидРазбиенияОсновногоОтчета - это должна быть одна из констант Стр, Кол или Таб. Если разбиений данного вида несколько (например, отчет разбит на строки по нескольким параметрам), то номер нужного разбиения в основном отчете передается в параметре аНомерРазбиенияОсновногоОтчета.

Прочие параметры аналогичны таким же параметрам в функциях [Пок](#) и [ПокОтчет](#) и являются необязательными.

Кроме того, с помощью необязательного параметра аИгнорироватьУсловиеОсновногоОтчета указывается, добавляется ли условие отбора параметров, переданное в функцию в параметре аУсловиеНаПараметры, к условию отбора основного отчета с помощью логического оператора AND (т.е. условие отбора вспомогательного отчета "сужает" условие отбора основного отчета). Если данный параметр равен True, то условия отбора параметров основного и вспомогательного отчета являются независимыми.

Пример: Если в отчет с разбиением на строки по паре "Процесс, Ресурс" нужно вывести свернутый конечный остаток ресурса (без учета процессов), то настройщик может создать отчет, например, с именем "ВторойОтчет". В него он добавляет "простой" показатель с уникальным именем "Количество". Тогда формула для обращения к этому показателю из основного отчета будет такой:

ИзОтчета('ВторойОтчет', 'Количество', 2)

где число 2 означает "сопоставлять значение второго разбиения основного отчета единственному разбиению вспомогательного отчета".

Функции "КолКол", "НомКол"

Функция "КолКол"

Описание функции:

Функция возвращает число колонок в текущей таблице отчета. Если в отчете включено разбиение на колонки, функция вернет 1 - в таком отчете всегда одна колонка. Наличие или отсутствие в отчете итоговой колонки не влияет на результат функции.

Функция "НомКол"

Описание функции:

Функция возвращает номер текущей заполняемой колонки в отчете. Если заполняется итоговая колонка в таблице, функция вернет 0.

Как правило, значение, возвращаемое этой функцией, используется как один из параметров функции [Пок и ее аналогов](#).

Функции "КолСтр", "НомСтр"

Функция "КолСтр"

Описание функции:

func КолСтр :Integer;

Функция возвращает число строк в текущей группе отчета. Если в отчете выключена иерархия по строкам, возвращается число строк в текущей таблице отчета. Наличие или отсутствие в отчете итоговой строки не влияет на результат функции.

Пример:

1. Пусть отчет строится за 2005 год, разбит на строки по времени и каждый из месяцев этого года попал в отчет. Тогда функция КолСтр вернет количество месяцев в году, т.е. 12.

Функция "НомСтр"

Описание функции:

func НомСтр :Integer;

Функция возвращает номер текущей заполняемой строки в отчете. Если заполняется итоговая строка в группе или таблице, функция вернет 0. Как правило, значение, возвращаемое этой функцией, используется как один из параметров функции [Пок и ее аналогов](#)

Функция "НайтиЗапись"

Описание функции:

```
func НайтиЗапись(  
    const aКлассЗаписи :class Record;  
    const aФильтр :String  
) :Record;
```

Функция возвращает запись типа aКлассЗаписи, удовлетворяющую фильтру, переданному во втором параметре. Если таких записей несколько, возвращается первая удовлетворяющая условию запись. Как правило, эта функция используется, если необходимо найти запись, зная значение какого-то уникального поля этой записи.

Функция "Огр"

Описание функции:

```
func Огр(  
    const аИмяПоля :String;  
    const аЗначение :Variant;  
    const аВключатьПоддерево, аИгнорироватьПустоеЗначение :Logical = True  
) :String;
```

Функция возвращает строку фильтра на записи (например, на записи картотеки). Эта функцию рекомендуется использовать при записи [формулы для строкового фильтра](#) ссылочного параметра отчета.

Параметр аИмяПоля задает имя поля, на которое накладывается ограничение, параметр аЗначение задает значение (или значения), которые накладывают ограничение. При этом параметр аЗначение может быть либо скалярным значением, либо массивом. В качестве этого параметра можно, например, использовать результат, возвращенный функциями [Парам](#) или [ПарамРасш](#). Тип значения (или значений), переданных в этом параметре, должен быть совместим с типом поля аИмяПоля. В случае, если в качестве имени поля передана пустая строка, значение (или значения) должно быть записью того же типа, что и та, на которую накладывается ограничение.

Если ограничение накладывается на ссылочное поле, параметр аВключатьПоддерево задает, какое будет наложено ограничение - на строгое равенство или на вхождение в группу.

Параметр аИгнорироватьПустоеЗначение определяет, что вернет функция в случае, если параметр аЗначение равен nil. Если аИгнорироватьПустоеЗначение - истина, то функция вернет пустую строку. В противном случае функция вернет ограничение вида 'аИмяПоля = nil'.

Функция "ОстатокОбязательств"

Данная функция предназначена для использования в [формулах параметрических отчетов](#).

Описание функции:

```
ОстатокОбязательств(  
    НазваниеОтчета :String;  
    НомерПоказателяОткудаВычитать :Integer;  
    НомерПоказателяЧтоВычитать :Integer  
) :Unit;
```

Параметры:

- НазваниеОтчета - строка с именем существующего отчета, который будет строиться при вызове данной функции; к настройкам этого отчета предъявляются специальные требования (*см. ниже*); если отчет размещен в папке, а не в корне иерархии отчетов, необходимо указывать полное имя вместе с элементами всего пути, разделенными точками, например, "Уточняющие.Остаток_доп";
- НомерПоказателяОткудаВычитать - порядковый номер показателя (из числа показателей указанного отчета), из которого будет браться значение для вычитания; это - уменьшаемое;
- НомерПоказателяЧтоВычитать - порядковый номер показателя (из числа показателей указанного отчета), который будет вычитаться из вышеуказанного показателя; это - вычитаемое.

Возвращаемое значение - измеритель с разностью конечных остатков заданных показателей. Если уменьшаемое меньше вычитаемого, возвращается ноль.

В целях терминологической ясности будем называть отчет, заданный с помощью параметра НазваниеОтчета, *дополняющим*, поскольку этот отчет предназначен для получения дополнительных данных, попадающих во внешний (по отношению к нему) отчет. Дополняющий отчет является *вызываемым*, опосредованно - с помощью описываемой функции, из клеток внешнего или *вызывающего* отчета.

Принцип работы функции следующий. Формула, содержащая функцию, вызывается в контексте некоторого вызывающего отчета. В каждой клетке результирующей таблицы этого отчета, в которой вычисляется формула, известны конкретные значения элементов разбиения. Например, отчет может быть разбит на строки по товарам, а на колонки по субъектам. Эти значения разбиения используются в вызываемом отчете для отбора лишь тех элементов разбиения, которые имеют аналогичные значения. Так, для строки вызывающего отчета с конкретной ТМЦ функция отбирает только те данные вызываемого отчета, которые также относятся к той же ТМЦ. Если в вызываемом отчете находится несколько элементов разбиения (например, строк) с искомым значением (например, с конкретной ТМЦ), то результаты вычитания в каждой строке двух заданных показателей затем построчно суммируются (*см. пример*). Учитывая тот факт, что из результата вычитания фактически берется лишь дебетовая составляющая (т.е. результат вычитания либо положительный, если из большего вычиталось меньшее значение, либо нулевой, если из меньшего вычиталось большее), сумма таких результатов по всем строкам в общем случае не равна значению, которое можно было бы получить, сперва рассчитав суммы первого и второго показателей и лишь затем вычтя из первой суммы вторую.

Следует иметь в виду, что вычисление формулы для разных клеток отчета происходит при использовании данной функции оптимизированным образом, т.е. построение вызываемого отчета осуществляется только один раз, и затем его результаты используются при последующих вызовах функции ОстатокОбязательств.

Название ОстатокОбязательств отражает лишь один частный прикладной смысл применения функции. В общем случае функция позволяет получить с помощью указанного отчета разницу между значениями показателей, например, между количеством выпущенного в счетах (заказанного покупателями) товара и остатком имеющегося на складе.

При использовании функции следует обратить внимание на следующие нюансы:

1. Вызываемый отчет должен как минимум иметь те же разрезы, что и вызывающий отчет. Например, если в

вызывающем отчете есть разбиение по процессу, то в вызываемом оно обязательно должно быть, причем не важно в каком виде (по строкам, по столбцам или по таблицам). Иными словами, функция ищет соответствие в отчетах по типу разбиения (по параметру, корр.параметру, дате, счету и т.д.) и параметру разбиения в случае разбиения по параметру или корр.параметру. Например, если в одном отчете есть разбиение на строки по параметру "Процесс", то и в другом отчете в любом из измерений должно быть разбиение по параметру "Процесс" (или по сочетанию нескольких параметров, куда "Процесс" также входит, например, "Субъект,Процесс"), в то время как разбиение по корр.параметру "@Процесс" не будет применимым. Оптимально, если разрезы отчетов совпадают.

2. Оба отчета должны содержать разбиение по ресурсу или корр.ресурсу.
3. В отчетах нельзя задавать один и тот же разрез по разным разбиениями (например, на строки и на столбцы по ресурсу).
4. Отчеты не должны быть иерархическими.
5. Показатели, индексы которых переданы через параметры НомерПоказателяОткудаВычитать и НомерПоказателяЧтоВычитать, не должны быть пользовательскими (т.е. должны заполняться машиной проводок, а не по формулам параметрических настроек).

Пример:

Пусть для бизнес-процесса склада имеется отчет "Остаток и резерв" с разбиением на строки по ресурсам. Наравне с простым показателем "Остаток" (остатки по измерителю "Количество") в нем имеется пользовательский показатель "Резерв" с формулой:

ОстатокОбязательств("Остаток_доп", 2, 1)

Имеется также отчет "Остаток_доп" с разбиением на строки по ресурсам и заказам, причем первый показатель считается по измерителю "Количество" в статусе "факт" (признак управленческой проводки по фактическому движению ресурсов), а второй - по тому же измерителю в статусе "Резерв" (признак управленческой проводки по движению резервируемых ресурсов). Таким образом, разность между вторым и первым значениями представляет собой индикатор наличия неотгруженного резерва. Разбивка на строки не только по ресурсам, но и по заказам, приводит к тому, что одна и та же ТМЦ может фигурировать в нескольких строках отчета "Остаток_доп", если она была оприходована, отгружена или зарезервирована в контексте разных заказов (сделок). При этом те строки, в которых фактические остатки больше резерва, дают в результате вычитания ноль (все зарезервированные товары, если они были, уже отгружены), а в противном случае - количество зарезервированного товара. Все резервы одной и той же ТМЦ из разных строк суммируются и попадают в строку вызывающего отчета (в колонку "Резерв") для той же ТМЦ, и там же в соседней колонке "Остаток" выводятся её фактические остатки.

Функции "Парам", "ПарамРасш"

Функция "Парам"

Описание функции:

```
func Парам(const aИмяПараметра :String) :Record;
```

Функция возвращает значение параметра в шапке отчета с уникальным именем aИмяПараметра. Параметр с таким именем должен быть настроен на [странице "Параметры"](#) диалога настроек отчета и должен иметь ссылочный тип.

Если для этого параметра допустим ввод нескольких значений, функция вернет только первое введенное значение.

Если параметра с таким именем нет в отчете, функция генерирует исключение.

Функция "ПарамРасш"

Описание функции:

```
func ПарамРасш(const aИмяПараметра :String) :Variant;
```

Эта функция может вызываться для параметров любых типов. Если для параметра допустим ввод нескольких значений, функция вернет массив, элементами которого будут значения параметра. В противном случае функция вернет скалярную величину - одиночное значение параметра. В этом случае тип возвращаемого значения совпадает с типом параметра, указанным в его настройках.

Как правило, значение, возвращаемое функциями Парам и ПарамРасш, используется как входной параметр для другой функции.

Например, таким образом можно задавать

- Значения ограничений для функции [Orp](#).

Функции "Пок", "ПокИзм", "ПокЧаст", "Чис", "Ед"

Функция "Пок"

Описание функции:

```
func Пок(  
    const аИмяПоказателя :String;  
    const аОстОбор :ОстОбор = ОстОборПоУмолчанию;  
    const аДебКре :ДебКре = ДебКреПоУмолчанию;  
    const аНомерКолонки :Integer = НомКол;  
    const аНомерСтроки :Integer = НомСтр  
) :Variant;
```

Функция возвращает значение показателя отчета с уникальным именем аИмяПоказателя. Смысл параметров аОстОбор и аДебКре см. в [соответствующей теме помощи](#).

Параметр аНомерКолонки имеет смысл задавать в случае, если в отчете есть разбиение на колонки, и задает колонку отчета, из которой берется значение показателя. Значение этого параметра может быть любым в пределах от 0 до [КолКол](#). По умолчанию этот параметр равен [НомКол](#), т.е. значение показателя берется из текущей заполняемой колонки отчета. Нулевое значение параметра указывает на то, что значение показателя берется из итоговой колонки отчета.

С параметром аНомерСтроки все аналогично. Он задает колонку отчета, из которой берется значение показателя. Его значение может лежать в пределах от 0 до [КолСтр](#), по умолчанию оно равно [НомСтр](#). Нулевое значение параметра указывает на то, что значение показателя берется из итоговой строки группы или таблицы.

Т.к. "пользовательские" показатели отчета заполняются после простых и вычисляемых показателей, с помощью функции Пок можно получить значение любого из таких показателей

"Пользовательские" показатели в отчете заполняются в том порядке, в котором для них указаны формулы на [странице "Показатели"](#) диалога настроек отчета. Поэтому для таких показателей в функции Пок можно указать имя любого "пользовательского" показателя, чья формула описана выше в [таблице формул](#).

Пусть в этой таблице заданы формулы для трех показателей отчета, и имена этих показателей (в порядке задания формул для них) - Показатель1, Показатель2 и Показатель3. Если в формуле для Показателя2 встречается функция Пок, с ее помощью можно получить значение Показателя1 в любой клетке в пределах текущей группы отчета. Значения Показателя3 в момент вычисления формулы для Показателя2 еще не заполнены, и если в функцию Пок передать имя этого показателя, она вернет пустое значение.

Кроме того, при вычислении значений Показателя2 можно использовать значения самого этого показателя в других клетках текущей группы отчета. При этом надо учитывать порядок заполнения значений показателя в группе. Сначала заполняется значение в первой колонке первой строки, затем - значение во второй колонке первой строки и т.д., пока не заполнится вся первая строка. Последней заполняется итоговая колонка первой строки. Затем в таком же порядке заполняются вторая, третья и т.д. строки, пока не заполнится вся группа. Наконец, заполняется итоговая строка в группе.

Таким образом, в функции Пок для Показателя2 можно использовать значения самого этого показателя:

- Из любой строки с номером, меньшим [НомСтр](#)
- Из строки с номером [НомСтр](#), если номер колонки не превосходит [НомКол](#)
- В итоговой колонке можно использовать любые значения показателя из строк с меньшими номерами, плюс любое неитоговое значение из текущей строки отчета.
- В итоговой строке можно использовать любые значения из неитоговых строк отчета, плюс значения из

- колонок итоговой строки, чей номер не превышает [НомКол](#).
- В итоговой строке итоговой колонки можно использовать любые значения показателя в пределах текущей группы отчета.

Есть особенности использования функции Пок для разных типов формул.

- Если функция Пок встречается в [формуле, выполняющейся в контексте строки](#), кроме параметров аОстОбор и аДебКре, необходимо явно указать номер колонки.

Кроме того, при явном задании параметра аОстОбор существуют следующее ограничение:

- В отчете, в котором не выводятся остатки, нельзя явно обращаться к начальному или конечному остатку показателя. В таком отчете недопустимы формулы вида Пок('Количество', НОст) или Пок('Количество', КОст). Формулы Пок('Количество', Об) или Пок('Количество') - допустимы.

Если указанное правило не выполнено, при вычислении значения функции Пок произойдет ошибка.

Примеры:

Пусть в отчете есть простой показатель Количество и "пользовательские" показатели ПокКолич и ИтогКолич.

1. Пусть для показателя ПокКолич задана формула

Пок("Количество").

Эта формула вернет значение показателя "Количество" в текущей колонке текущей строки, при этом при вычислении дебетового начального остатка показателя ПокКолич будет использован дебетовый начальный остаток показателя Количество, при вычислении кредитового оборота - кредитовый оборот - и т.д.

По сути дела, эта формула "копирует" значения показателя Количество в показатель ПокКолич.

2. Пусть для показателя ПокКолич задана формула

Пок("Количество", Об, Кре).

Эта формула всегда вернет кредитовый оборот показателя Количество в текущей колонке текущей строки отчета.

3. Пусть для показателя ИтогКолич задана формула:

If([НомСтр](#) <= 1, Пок("Количество"), [Сум](#) (Пок("ИтогКолич", , , , [НомСтр](#) - 1), Пок("Количество"))) .

Эта формула вернет нарастающий итог по строкам для показателя Количество

Функция "ПокИзм"

Эта функция имеет те же параметры, что и функция Пок, и отличается только тем, что возвращает значение, приведенное к типу Измеритель. Поэтому показатель, имя которого передается в эту функцию, должен быть типа Измеритель.

Функция "ПокЧаст"

Эта функция возвращает значение, приведенное к типу Частное. Показатель, значение которого возвращает эта функция, должен быть типа Частное.

Функция "Чис"

Эта функция допустима для показателей типа Целое, Число, Измеритель и Частное. Для первых двух типов она возвращает само значение показателя, для типов Измеритель и Частное - числовую составляющую его значения. Возвращаемое значение имеет тип Число.

Функция "Ед"

Эта функция допустима для показателей типа Измеритель и возвращает единицу измерения для значения показателя. Возвращаемое значение имеет тип Признак. Чтобы из него получить соответствующую запись, необходимо воспользоваться функцией [Зап.](#)

Функция "ПокОтчет"

Описание функции:

```
func ПокОтчет(  
    const аИмяПоказателя :String;  
    const аНачСмещения, аКонСмещения :Integer[] = [];  
    const аУсловиеНаСчета :String = Отчет.AccountFilter;  
    const аУсловиеНаПараметры :String = Отчет.ParameterFilter;  
    const аБезИерархии :Logical = False  
): Variant;
```

Эта функция строит дополнительный отчет с теми же настройками, что и основной, за исключением дат и условий на счета и параметры. В этот отчет добавляется показатель с теми же настройками, что у показателя основного отчета с уникальным именем аИмяПоказателя, который должен быть простым или вычислимым.

Содержимое дополнительного отчета определяется остальными параметрами функции ПокОтчет:

- аНачСмещения - массив из трех элементов. Он определяет смещение начальной даты дополнительного отчета относительно начальной даты основного отчета. Первый элемент этого массива задает число месяцев, на которое должна отличаться нач. дата дополнительного отчета от нач. даты основного отчета, второй элемент - число дней, третий элемент - число секунд. Если в массиве аНачСмещения нет элементов, нач. дата дополнительного отчета будет считаться равной нач. дате основного отчета.
- аКонСмещения - задает смещение конечной даты дополнительного отчета относительно конечной даты основного отчета. Этот массив заполняется по тем же правилам, что и массив аНачСмещения. Если этот параметр не задан, смещения для конечной даты отчета будут считаться равными смещениям для начальной даты. Если заданы смещения для начальной даты, а конечная дата дополнительного отчета должна совпадать с конечной датой основного отчета, в параметре аКонСмещения необходимо передать нулевые смещения.
- аУсловиеНаСчета - если ограничение на счета дополнительного отчета должно отличаться от ограничения основного отчета, в параметре аУсловиеНаСчета надо передать нужное ограничение. Если этот параметр опущен, ограничение на счета дополнительного отчета берется из основного отчета. Если необходимо отменить ограничение на счета основного отчета, в этом параметре надо передать пустую строку.
- аУсловиеНаПараметры - задает ограничение на параметры в дополнительном отчете. Так же, как и для условия на счета, если этот параметр опущен, ограничение на параметры дополнительного отчета берется из основного отчета. Если необходимо отменить ограничение на параметры основного отчета, в этом параметре надо передать пустую строку.

Параметр аБезИерархии важен, если функция ПокОтчет используется в иерархическом отчете, и управляет тем, строится ли дополнительный отчет так же иерархическим или же в нем иерархия отключается. Учет или неучет иерархии в дополнительном отчете, очевидно, дадут разные результаты. Если данный параметр опущен, он принимается равным **False** (иерархия не отключается).

Замечания:

1. Функция ПокОтчет не добавляет таблиц, строк и колонок в основной отчет. Если, например, в основном отчете есть строки А и С, а в дополнительном - строки А и В, функция ПокОтчет выведет значение показателя в строке А, но строка В не будет добавлена в основной отчет.
2. В иерархических отчетах для показателя, в формуле для вычисления которого встречается функция ПокОтчет, в [диалоге расширенной настройки показателя](#), рекомендуется задать вычисление итогов суммированием. Это связано с тем, что в ряде случаев невозможно правильно сопоставить итоги по группам в дополнительном и

основном отчетах. Особенно это важно, если параметр **аБезИерархии** установлен в **True** - в этом случае итогов по группам как таковых в иерархическом отчете, скорее всего, не будет.

Примеры:

1. Пусть в отчете выводится оборот за текущий год, и в нем есть "простой" показатель с уникальным именем "Количество".

Формула

ПокОтчет("Количество", [-12])

вернет оборот показателя Количество за прошлый год.

Формула

ПокОтчет("Количество", [-12], [0])

вернет суммарный оборот этого показателя прошлый и текущий годы.

2. Пусть в отчете из п.1 есть ограничения на счета и параметры.

Формула

ПокОтчет("Количество", [], [], ")")

вернет значение показателя с такими же настройками, как у показателя Количество, но без учета ограничения отчета на счета.

Формула

ПокОтчет("Количество", [], [], ", ")")

учтет ограничение на счета, но не учтет ограничения на параметры.

Формула

ПокОтчет("Количество", [], [], ", ", ")")

не учтет ограничения ни на счета, ни на параметры.

Функция "Поле"

Описание функции:

```
func Поле(  
    const аИмяПоля :String;  
    const аИзмерение :ИзмерениеОтчета = Стр;  
    const аНомерЗначения :Integer = 1  
) :Variant;
```

Функция может возвращать:

- Значение разбиения отчета по измерению аИзмерение. В этом случае в параметре аИмяПоля передается пустая строка.
- Если значение разбиения имеет тип Счет, Признак или Запись - можно получить его разыменование. В этом случае в параметре аИмяПоля передается имя поля, которое надо разыменовать.

Значение разбиения имеет тип Счет, если разбиение отчета по измерению аИзмерение - по счетам или корр. счетам; тип Признак, если разбиение - по параметру или корр. параметру; тип Запись - если отчет разбит по документу.

- Значение [дополнительного поля](#). В этом случае, параметр аИмяПоля должен начинаться с символа "\$", что будет означать, что необходимо получить именно дополнительное поле. Стоит заметить, что при помощи данной функции можно получить дополнительные поля только шапочной части поцесса, а также, только если разбиение отчета по параметру, корр.параметру или документу.

Если разбиение отчета по измерению аИзмерение - составное (например, одновременно по параметру и документу), каждому элементу отчета может соответствовать несколько значений разбиения. В этом случае в параметре аНомерЗначения передается номер значения разбиения, которое необходимо получить.

Для значений разбиения типа Признак параметр аИмяПоля может начинаться с символа "#". Это означает, что разыменовать надо не сам признак, а соответствующую ему запись.

Если параметр разбиения имеет тип Признак, и аИмяПоля равно "#", функция вернет запись, соответствующую этому признаку, не разыменывая эту запись.

Есть особенности использования функции Поле в [формулах, выполняемых в контексте строки отчета](#). В этом случае параметр аИзмерение не может принимать значение [Кол](#), т.е. можно получать значения разбиения только из таблиц и строк отчета.

Функция "Распр"

Описание функции:

```
func Распр(  
    const аИмяПоказателя1, аИмяПоказателя2 :String;  
    const аПротивоположныеЗначения :Logical = False;  
    const аАналитика :String = '';  
    const аПоКоррЗначениям :Logical = False  
) :Unit;
```

Вычисляет "распределение" итогового значения 2-го показателя по значениям в строках 1-го показателя. Что такое распределение и как оно вычисляется - см. [ниже](#).

Если параметр ПротивоположныеЗначения установлен в True, для вычисления распределения в дебетовом столбце берется кредитовое значение распределяемого показателя, и наоборот.

Значение распределяющего показателя всегда берется в соответствии с тем, для какого столбца вычисляется распределение (т.е. для дебетового столбца - дебетовое для кредитового - кредитовое).

Если выводится свернутое значение распределения, берутся свернутые значения распределяемого и распределяющего показателей.

Параметр Аналитика служит для [распределения с учетом аналитики](#), а параметр ПоКоррЗначениям - [распределение по корреспондирующим значениям](#). Эти параметры могут использоваться как по отдельности, так и в [комбинации](#).

Что такое распределение?

Простое распределение

Допустим, что в отчете с разбиением на строки необходимо итоговое значение одного показателя распределить по значениям в строках таблицы другого показателя. Такая задача возникает, например, при распределении общей суммы оплаты по выписанным счетам.

Реализуется это так: делается "проход" по строкам таблицы, и итоговое значение распределяемого показателя (Б) распределяется по значениям распределяющего показателя (А): итоговое (распределяемое) значение показателя Б сравнивается со значением показателя А в 1-й строке отчета; если итоговое значение больше значения показателя А, распределение считается равным значению показателя А, а распределяемое значение уменьшается на значение показателя А.

Далее уменьшенное распределяемое значение сравнивается со значением показателя А во 2й строке, и операция повторяется до тех пор, пока распределяемое значение не станет меньше значения показателя А в текущей строке отчета. Значение распределения в этой строке считается равным распределяемому значению, значение распределения во всех последующих строках отчета считается равным нулю.

Пример. Пусть итоговое значение показателя Б = 150^{руб}. Приведем таблицу со значениями показателя А и значениями распределения:

Показатель А	Распределение
20 ^{руб}	20 ^{руб} (осталось 130 ^{руб})
15 ^{руб}	15 ^{руб} (осталось 115 ^{руб})
60 ^{руб}	60 ^{руб} (осталось 55 ^{руб})
25 ^{руб}	25 ^{руб} (осталось 30 ^{руб})
70 ^{руб}	30 ^{руб} (осталось 0 ^{руб})
30 ^{руб}	0 ^{руб}
45 ^{руб}	0 ^{руб}

Распределение по корреспондирующим значениям

Вычисляется аналогично простому распределению, но сравниваются корреспондирующие значения распределяющего показателя с итогом по корреспондирующему же значению распределяемого показателя. Если итог больше корр. значения распределяемого показателя, распределение считается равным ПРЯМОМУ значению распределяемого показателя, итог уменьшается на КОРРЕСПОНДИРУЮЩЕЕ значение распределяемого показателя. Если итог меньше корр. значения распределяемого показателя, распределение вычисляется по формуле: (значение показателя Б/корр. значение показателя Б) * Итог.

Пример. Пусть итоговое корр. значение показателя Б = 30^шт. Приведем таблицу с прямыми и корр. значениями показателя А и значениями распределения:

Показатель А	Показатель А(корр.)	Распределение
20^руб	3^шт	20^руб (осталось 27^шт)
15^руб	8^шт	15^руб (осталось 19^шт)
60^руб	15^шт	60^руб (осталось 4^шт)
25^руб	1^шт	25^руб (осталось 3^шт)
70^руб	7^шт	30^руб (70^руб/7^шт * 3^шт; осталось 0^шт)
30^руб	2^шт	0^руб
45^руб	9^шт	0^руб

Распределение с учетом аналитики

Отличается от простого распределения тем, что распределяющее и распределяемое значение могут сравниваться только в том случае, если они имеют одинаковую аналитику, указанную в параметре Аналитика. В этом случае строится 2 дополнительных отчета - такой же, как исходный, но с доп. разбиением на колонки по корр. параметру Аналитика и отчет с разбиением на строки по корр. параметру Аналитика.

Значения в строках второго отчета распределяются отдельно по соотв. колонкам 1-го отчета. Затем распределенные значения в строках 1-го отчета суммируются, и просуммированные значения представляют собой искомые значения распределения.

Пример. Пусть распределение вычисляется по параметру Ресурс, и отчет 1 имеет вид:

	Товар1	Товар2	Товар3	Товар4
Счет1	3^руб	5^руб	8^руб	2^руб
Счет2	6^руб	4^руб	0^руб	5^руб

Отчет 2 имеет вид:

Товар1	7^руб
Товар2	8^руб
Товар3	5^руб
Товар4	10^руб

Получаем частичные распределения:

	Товар1	Товар2	Товар3	Товар4
Счет1	3^руб (осталось 4^руб)	5^руб (ост. 3^руб)	5^руб (ост. 0^руб)	2^руб (ост. 8^руб)
Счет2	4^руб (осталось 0^руб)	3^руб (ост. 0^руб)	0^руб	5^руб (ост. 3^руб)

Суммируя частичные распределения, получаем искомые значения:

Счет1	3^руб + 5^руб + 5^руб + 2^руб = 15^руб
Счет2	4^руб + 3^руб + 0^руб + 5^руб = 12^руб

Распределение с учетом аналитики по корр. значениям

Такое распределение вычисляется аналогично предыдущему, но частичные распределения вычисляются с учетом корр. значений.

Требования к отчету для использования функции Распр

1. Не должно быть разбиения на колонки.
2. Должна быть выключена иерархия по строкам.
3. Должна быть выключена иерархия по таблицам (само разбиение на таблицы допустимо).

Функции "Сум", "Разн", "Част"

Функция "Сум"

Описание функции:

```
func Сум(const aИзм1, aИзм2 :Variant) :Unit;
```

Функция возвращает сумму двух значений типа Измеритель. Возвращаемое значение тоже имеет тип Измеритель. Если единицы измерения в слагаемых - разные, но совместимые, функция производит пересчет значения второго измерителя в единицу измерения первого. Если единицы измерения несовместимы - сумма вычисляется без пересчета, а вместо единицы измерения подставляется пустое значение (в отчете такая единица измерения выводится как "***")

Функция "Разн"

Описание функции:

```
func Разн(const aИзм1, aИзм2 :Variant) :Unit;
```

Функция возвращает разность двух значений типа Измеритель. Тип возвращаемого значения и правила пересчета единиц измерения - такие же, как для функции Сум.

Функция "Част"

Описание функции:

```
func Част(const aИзм1, aИзм2 :Variant) :Numeric;
```

Функция возвращает отношение значений типа Измеритель. Возвращаемое значение имеет тип Число. Правила пересчета единиц измерения - такие же, как для функции Сум.

Функция "ТекущийДебКре" в параметрических отчетах

Описание функции:

func ТекущийДебКре :SumDebCre

Данная функция используется в формулах вычисляемых показателей параметрических отчетов.

Функция возвращает признак того, вычисляется ли формула в дебете или кредите (допустимые значения - константы [Деб](#), [Кре](#), [Сверн](#)).

Функция "ТекущийОстОбор" в параметрических отчетах

Описание функции:

func ТекущийОстОбор :SumKinds

Данная функция используется в формулах вычисляемых показателей параметрических отчетов.

Функция возвращает признак того, вычисляется ли данная формула в остатках или оборотах (допустимые значения - константы [НОст, Об, КОст](#))

Функция "MakeIs" в реквизитах

Данная функция предназначена для решения проблемы двух операторов IS, существующих в ТБ.Студии.

Дело в том, что в языке фильтров, применяемом для отбора записей из таблиц информационной базы, оператор IS означает "входит в группу" по иерархии записей, а в языке ТБ.Скрипт IS - это оператор приведения типов. Выражения в формулах реквизитов записываются на языке ТБ.Скрипт, поэтому, если требуется, скажем, написать в каком-либо условии формулу типа "менеджер принадлежит группе "отдел продаж", нельзя написать просто

Менеджер IS Процесс('Отдел продаж')

т.к. это вызовет ошибку, т.к. оператор IS будет интерпретирован как оператор приведения типов. Вместо этого надо формировать логическое выражение с помощью функции MakeIs: MakeIs('Менеджер', Процесс('Отдел продаж')).

Описание функции:

```
func MakeIs (  
    aПараметр :String;  
    aГруппа    :Record  
) :String;
```

Функция формирует строку с выражением, интерпретация которого дает True, если значение параметра aПараметр, в котором содержится имя ссылочного поля, равно ссылке на запись, переданной в параметре aГруппа, или является ссылкой на запись, по иерархии входящей в группу aГруппа или одну из ее подгрупп.

Примеры

Редактор формул реквизита

MakeIs('GroupDoc', %1)

MakeIs('GroupDoc', Процесс('ООО "Комбинат ЖБИ"'))

Реквизит

Запись

Проверка

Поле

Записи	
1	ООО "Комбинат ЖБИ"

Поле...

Комментарий:

С помощью данной формулы на реквизит накладывается ограничение "только записи, входящие в группу процессов "ООО "Комбинат ЖБИ"".

Функция "Value/Значение/ТекущееЗначение" в реквизитах

Описание функции

func Value synonym Значение, ТекущееЗначение :Variant;

Функция возвращает значение реквизита, для которого идет пересчет. Данную функцию имеет смысл употреблять в пересчетах по событиям.

Функция "ВидПериодаКарточкиСоздателя" в реквизитах

Функция определяет значение периода карточки объекта, в контексте которого был создан текущий объект.

Описание функции:

```
func ВидПериодаКарточкиСоздателя :ПериодыВремени
```

Функция возвращает период времени карточки-создателя, из которой был создан текущий объект. Период времени - это предопределенная в системе целочисленная константа. Возможные константы периода времени, возвращаемые данной функцией, перечислены ниже.

Константы периода времени

Стандартные константы периода времени, предусмотренные в проекте ТБ.Управление.

```
0 = Интервал времени между определенными датами.  
1 = День.  
2 = Неделя.  
3 = Месяц.  
4 = Квартал.  
5 = Год.  
6 = Все время.  
7 = Последний месяц.  
8 = Полугодие.  
9 = период в 9 месяцев.
```

Таким образом, если результат вычисления функции - одно из перечисленных целых чисел, то значением периода объекта-создателя будет являться период времени, соответствующий целому числу.

Пример

Пример использования функций ВидПериодаКарточкиСоздателя и [ДатаКарточкиСоздателя](#) в пересчетах по событиям объекта.

При создании нового документа, если период карточки создателя - день, то выбирается дата из карточки объекта-создателя, а если период какой-то другой - то текущая дата.

Редактор формул реквизита

Редактор формул реквизита

if(ВидПериодаКарточкиСоздателя<>1,Now,ДатаКарточкиСоздателя)

if(ВидПериодаКарточкиСоздателя<>1,Now,ДатаКарточкиСоздателя)

Реквизит

Запись

Проверка

Записи

Поле...

Комментарий:

Функция "ВыбратьЗапись" в реквизитах

Описание функции:

```
func ВыбратьЗапись (  
    аПредварительноОчищать :Logical = False;  
    аОднозначныйВыбор      :Logical = True;  
    аФильтр                 :String  = '';  
    аУпорядочивание        :String  = '';  
    аКлассЗаписи            :Class Record = nil;  
    аПодзагружаемыеПоля    :String  = ''  
) :Record;
```

Функция возвращает запись, удовлетворяющую ограничениям реквизита, для которого производится вычисление.

Параметр аОднозначныйВыбор отвечает за то, нужно ли выбирать только в случае, если под ограничение попадает единственная запись (True) или нет (False).

Параметр аПредварительноОчищать отвечает за то, что нужно вернуть, если под фильтр ничего не попадает или, в случае однозначного выбора, попадает более одной записи: True - вернуть nil, False - оставить предыдущее значение.

С помощью параметра аФильтр передается фильтр, в соответствии с которым выбирается запись. Если параметр опущен, то для выбора записи используется фильтр, заданный в ограничениях реквизита, который заполняется с использованием данной функции.

С помощью параметра аУпорядочивание можно задать приоритет выбора записи при неоднозначном выборе.

Параметр аКлассЗаписи является необязательным и определяет класс записи, который необходимо выбрать. Если параметр не указан, то класс определяется как класс записи, с которым работает реквизит, для которого производится вычисления.

Необязательный параметр аПодзагружаемыеПоля определяет поля записи, которые будут загружены с сервера на клиент в момент поиска записи. Данный параметр можно использовать для оптимизации работы функции. подзагрузок.

Функция имеет смысл только для реквизитов ссылочного типа.

Примеры

ВыбратьЗапись

Если под ограничения реквизита попадает ровно одна запись, то возвращается она, иначе - предыдущее значение реквизита.

ВыбратьЗапись(True)

Если под ограничения реквизита попадает ровно одна запись, то возвращается она, иначе - nil.

ВыбратьЗапись(False, False)

Если под ограничения реквизита попадает хоть одна запись, то возвращается первая попавшаяся, иначе -

предыдущее значение реквизита.

ВыбратьЗапись(True, False)

Если под ограничения реквизита попадает хоть одна запись, то возвращается первая попавшаяся, иначе - nil.

Функция "Группы" в реквизитах

Данная функция устанавливает фильтр на поле класса записи определенных групп. Как правило, она используется в выражениях фильтров прав доступа. В момент вызова функции Группы (например, при наложении фильтра прав доступа при старте сессии) выполняется запрос записей, входящих в переданные нее в качестве параметров группы иерархии, и затем формируется фильтр типа "DocID in [...]", который выполняется сервером данных быстрее и эффективнее, чем фильтр с оператором IS или другим видом отбора по иерархии.

Используя функцию Группы, имеет смысл накладывать фильтр на постоянные параметры Структуры бизнеса. К примеру, если при работе функции в определенном списке групп, на который наложен фильтр, был добавлен новый элемент, то этот элемент уже не войдет в фильтр.

Описание функции:

```
func Группы(  
    aТипЗаписи      :Class;  
    aПараметр       :String;  
    aСписокГрупп    :Record[ ]  
) :String;
```

Через параметр aТипЗаписи передается класс записи списка групп, на который необходимо установить фильтр.

Через строковый параметр aПараметр передается поле списка групп, по которому формируется фильтр.

Параметр aСписокГрупп указывает массив групп класса записи, поля которых будут учитываться при формировании фильтра.

Пример

Пример установки фильтра на поле Заказ групп "Денежные средства ЖБИ" и "Договора закупки ЖБИ" класса записи Процесс.

Редактор формул

Группы(Данные.Процесс, 'Заказ', [%1,%2])

Реквизит

Запись

Проверка

Записи	
1	ДЕНЕЖНЫЕ СРЕДСТВА ЖБ
2	Договора закупки ЖБИ

Комментарий:

Функция "ДатаКарточкиСоздателя" в реквизитах

Функция определяет дату карточки объекта, в контексте которого был создан текущий объект.

Описание функции:

```
func ДатаКарточкиСоздателя(аПоказатьНачальнуюДатуПериода :Logical = True):Date
```

Логический параметр аПоказатьНачальнуюДатуПериода указывает, возвращать ли начальную дату из карточки объекта-создателя (значение True или по умолчанию) или конечную дату (значение False).

Функция возвращает дату карточки объекта, из которой был создан текущий объект. Ссылку на сам объект-создатель можно получить с помощью функции [Создатель](#)

Пример

Пример использования функций [ВидПериодаКарточкиСоздателя](#) и ДатаКарточкиСоздателя в пересчетах по событиям объекта.

При создании нового документа, если период карточки создателя - день, то выбирается дата из карточки объект-создателя, а если период какой-то другой - то текущая дата.

Редактор формул реквизита

Редактор формул реквизита

if(ВидПериодаКарточкиСоздателя<>1,Now,ДатаКарточкиСоздателя)

if(ВидПериодаКарточкиСоздателя<>1,Now,ДатаКарточкиСоздателя)

Реквизит

Запись

Проверка

Записи

Поле...

Комментарий:

Функция "ЗначениеПоляПодтаблицы" в реквизитах

Данная функция позволяет получить значение определенного поля строки, в которой происходят пересчеты объекта, заданной подтаблицы.

Описание функции:

```
func ЗначениеПоляПодтаблицы(  
    аИмяПодтаблицы :String;  
    аИмяПоля        :String  
    ) :Variant;
```

Обязательный параметр аИмяПодтаблицы указывает на подтаблицу объекта, из которой необходимо получить значение поля.

Через параметр аИмяПоля передается поле подтаблицы, значение которого нужно определить.

Метод "ИндексПозиции" в реквизитах

Описание метода

ИндексПозиции :Integer;

Метод ИндексПозиции позволяет получить индекс позиции подтаблицы, в рамках которой идут пересчеты.

Метод "Интерфейс" в реквизитах

Описание метода

Интерфейс :Базовые.интЗаписьСРеквизитом;

Метод Интерфейс позволяет получить интерфейс объекта, в рамках которого идут пересчеты, а также обратиться к методам интерфейса.

Функция "КоличествоПозиций/PositionsCount" в реквизитах

Функция вычисляет количество позиций (строк) в подтаблице объекта.

Описание функции:

```
func КоличествоПозиций synonym PositionsCount(аФильтр :String = '') :Integer;
```

Через необязательный строковый параметр аФильтр задается фильтр для подтаблицы. Если параметр не указан, то функция возвращает количество строк во всей подтаблице.

Пример

Пример установки фильтра на тип движения "Обычное движение" в функции КоличествоПозиций. Функция вычислит количество позиций в подтаблице учетного движения "Обычное движение" процесса.

Редактор формул

КоличествоПозиций("ТипДвижения = " + Str (%1))

КоличествоПозиций("ТипДвижения = " + Str (ТипДвижения("Обычное движение")))

Комментарий:

Реквизит

Запись

Проверка

Записи	
1	Обычное движение

Поле...

Функция "КурсДокумента" в реквизитах

Функция предназначена для вычисления нестандартного курса валют в документе.

Описание функции:

```
func КурсДокумента(  
    аДата          :Date;  
    аПроцесс       :Данные.Процесс = nil;  
    аИсходнаяВалюта :Справочники.ЕдиницаИзмерения = nil;  
    аЦелеваяВалюта  :Справочники.ЕдиницаИзмерения = nil  
) :Numeric;
```

Возвращает количество единиц необходимой валюты за единицу исходной с учетом нестандартного курса на указанную дату в заданном [типе объекта](#).

Обязательный параметр аДата - дата, на которую необходимо вычислить курс.

аПроцесс является необязательным параметром. Указывается объект, в котором необходимо пересчитать курс указанных валют. Если данный параметр не указан, то функция вычисляет в текущем объекте.

Необязательный параметр аИсходнаяВалюта - валюта, относительно которой необходимо вычислить количество единиц необходимой (целевой) валюты. Если данный параметр не указан, то валюта берется из валюты объекта.

Необязательный параметр аЦелеваяВалюта - валюта, в которую необходимо произвести вычисления с учетом нестандартного курса. Если параметр не задан, то за целевую валюту берется рубль.

Примеры

Запрос на получение курса на сегодняшнюю дату в текущем типе объекта. Так как никакие параметры не указаны, кроме аДаты = Now, то функция вычисляет количество рублей в текущем объекте относительно валюты указанной в этом объекте на сегодняшнюю дату.

Запрос на получение курса рубля к евро на сегодняшнюю дату в текущем типе объекта.

Функция "НастройкиСервера"

Функция возвращает ссылку на объект, описывающий параметры текущего сервера.

Описание функции:

func НастройкиСервера :Настройки.ПараметрыСервера

Передаваемых параметров в функции нет. Функция возвращает ссылку на объект, содержащий параметры текущего сервера. Возможные параметры сервера описаны в разделе [Параметры сервера](#).

Пример

Запрос на получение параметра "Наше предприятие" текущего сервера.

Редактор формул реквизита

Редактор формул реквизита

НастройкиСервера.НашеПредприятие

НастройкиСервера.НашеПредприятие

Комментарий:

Реквизит

Запись

Проверка

Записи

Поле...

Функция "НачалоУчета" в реквизитах

Описание функции:

```
func НачалоУчета(аОбластьУчета :String) :Date
```

В строковом параметре аОбластьУчета в функцию передается имя области учета.

Функция возвращает дату, ранее которой период по данной области учета закрыт. Если в качестве параметра передана пустая строка, то функция возвращает минимальную принятую в системе дату проводки (01.01.1900).

Пример

Запрос на получение даты начала управленческого учета.

Редактор формул

НачалоУчета("УправленческийУчет")

Реквизит

Запись

Проверка

Комментарий:

Записи

Поле...

Функция "Позиция" в реквизитах

Описание функции:

```
func Позиция :Structure;
```

Функция возвращает ссылку на позицию (строку) подтаблицы объекта, в котором происходят пересчеты.

Функция "ПолеУникально" в реквизитах

Функция проверяет уникальность поля записи.

Описание функции:

```
func ПолеУникально synonym FieldUnique(  
    aПоле :String;  
    aДопПоле :Logical = False  
) :Logical;
```

В качестве параметра aПоле передается имя поля, значение которого нужно проверить на уникальность.

Необязательный логический параметр aДопПоле указывает на [дополнительное поле](#). Если этот параметр имеет значение True, то будет проверяться уникальность дополнительного поля. Иначе - нет.

Результат функции - логическое значение. Функция возвращает True, если поле имеет уникальное значение, иначе False.

Примеры

ПолеУникально("Имя")

Второй параметр опущен, значит проверяется уникальность значения поля Имя.

ПолеУникально("ДопИмя", True)

В данном случае проверяется уникальность дополнительного поля ДопИмя.

Функция "ПолучитьДопПолеПроцесса" в реквизитах

При помощи данной функции можно получить значение [дополнительного поля](#) из указанной записи любого класса, унаследованного от класса записи БазоваяСРеквизитом.

Описание функции:

```
func ПолучитьДопПолеПроцесса(  
    const aПроцесс :Базовые.БазоваяСРеквизитом;  
    aДопПоле :String;  
    aIndex :Integer = 0  
) :Variant;
```

Параметр aПроцесс указывает на запись, из которой необходимо получить значение дополнительного поля. Запись может быть указана из любого класса записи, унаследованного от БазоваяСРеквизитом, в которой используются дополнительные поля.

Через параметр aДопПоле указывается имя дополнительного поля, значение которого необходимо получить.

Необязательный параметр aIndex указывает на строку подтаблицы, из которой необходимо получить значение дополнительного поля, если оно используется в подтаблице записи.

Пример

Запрос на получение значения логического дополнительного поля "лДопПолеЗ" из 2 строки подтаблицы документа "Отгрузка 09".

Редактор формул

Редактор формул

ПолучитьДопПолеПроцесса(%1,'лДопПолеЗ',2)

ПолучитьДопПолеПроцесса(Процесс('Отгрузка 09'),'лДопПолеЗ',2)

Комментарий:

Записи	
1	Отгрузка 09

Функция "ПолучитьРеквизитПроцесса" в реквизитах

При помощи данной функции можно получить значение конкретного [реквизита](#) из указанной записи любого класса, унаследованного от класса записи БазоваяСРеквизитом.

Описание функции:

```
func ПолучитьРеквизитПроцесса(  
    const aПроцесс :Базовые.БазоваяСРеквизитом;  
    aКод :String;  
    aIndex :Integer = 0  
) :Variant;
```

Параметр aПроцесс указывает на запись, из которой необходимо получить значение реквизита. Запись может быть указана из любого класса записи, унаследованного от БазоваяСРеквизитом, в которой используются реквизиты.

Через параметр aКод указывается код реквизита, значение которого необходимо получить.

Необязательный параметр aIndex указывает на строку подтаблицы, из которой необходимо получить значение реквизита, если он является [табличным](#).

Пример

Запрос на получение значения реквизита "ОбщПроцессОткуда" из процесса "Отгрузка 09".

Редактор формул

ПолучитьРеквизитПроцесса(%1, "ОбщПроцессОткуда")

Реквизит

Запись

Проверка

Комментарий:

Записи	
1	Отгрузка 09

OK Отмена Помощь

Запрос на получение значения табличного реквизита "РесурсОткуда" из 4 строки подтаблицы процесса "Отгрузка 09".

Редактор формул

ПолучитьРеквизитПроцесса(%1, 'РесурсОткуда', 4)

Реквизит

Запись

Проверка

Комментарий:

Записи	
1	Отгрузка 09

OK Отмена Помощь

Метод "ПроверяемоеЗначение" в реквизитах

Описание метода

ПроверяемоеЗначение :Variant;

Метод ПроверяемоеЗначение используется при проверке [реквизитов-лимитов](#). Служит для получения нового, еще не установленного значения реквизита.

Функция "Процесс" в реквизитах

Описание функции

func Процесс :Record;

Функция возвращает ссылку на объект, для которого идут пересчеты.

Функция "Создатель" в реквизитах

Функция возвращает ссылку на объект, в контексте которого было создан данный объект.

Описание функции:

```
func Создатель :Record;
```

Значение функции определено только в случае создания нового объекта, если объект открыт на редактирование, то функция возвращает ссылку на пустой объект.

Функция "ТекущееПредставление" в реквизитах

Описание функции:

```
func ТекущееПредставление :Представления.интПредставление;
```

Данная функция возвращает интерфейс активного [представления](#) объекта.

Функция "УчСуммаКуда" в реквизитах

Функция получает учетную сумму со стороны "Куда" в строке документа на основании настроек заданного варианта стоимостного учета.

Описание функции

func УчСуммаКуда(парВариантУчета :Настройки.ВариантСтоимостногоУчета) :Numeric

Функция имеет один параметр парВариантУчета - вариант стоимостного учета, для которого требуется найти учетную сумму со стороны "Куда".

Результат функции - число.

Пример

- Запрос на получение учетной суммы со стороны "Куда" в текущей позиции документа для варианта стоимостного учета "Управленческий учет".

Редактор формул реквизита

УчСуммаКуда(%1)

Реквизит

Запись

Проверка

Записи

№	Наименование	Действие
1	Управленческий учет	Поле...

Комментарий:

Функция "УчСуммаОткуда" в реквизитах

Функция получает учетную сумму со стороны "Откуда" в строке документа на основании настроек заданного варианта стоимостного учета.

Описание функции

func УчСуммаОткуда(парВариантУчета :Настройки.ВариантСтоимостногоУчета) :Numeric

Функция имеет один параметр парВариантУчета - вариант стоимостного учета, для которого требуется найти учетную сумму со стороны "Откуда".

Результат функции - число.

Примеры

- Запрос на получение учетной суммы со стороны "Откуда" в текущей позиции документа для варианта стоимостного учета "Управленческий учет":

Редактор формул реквизита

УчСуммаОткуда(%1)

Реквизит

УчСуммаОткуда(ВариантСтоимостногоУчета("Управленческий учет"))

Запись

Проверка

Комментарий:

Записи	
1	Управленческий учет

ОК Отмена Помощь

Функция "УчЦенаКуда" в реквизитах

Функция отображает учетную цену со стороны "Куда" в строке документа на основании уже рассчитанных цен по заданному варианту стоимостного учета.

Описание функции

func УчЦенаКуда(парВариантУчета :Настройки.ВариантСтоимостногоУчета) :Numeric

Функция имеет один единственный параметр парВариантУчета - вариант стоимостного учета, для которого требуется найти учетную цену со стороны "Куда".

Результат функции - число.

Пример

- Запрос на получение учетной цены "Куда" в текущей позиции документа для варианта стоимостного учета "Управленческий учет":

Редактор формул реквизита

УчЦенаКуда(%1)

УчЦенаКуда(ВариантСтоимостногоУчета("Управленческий учет"))

Комментарий:

Реквизит

Запись

Проверка

Записи

1	Управленческий учет	Поле...
---	---------------------	---------

OK Отмена Помощь

Функция "УчЦенаОткуда" в реквизитах

Функция отображает учетную цену со стороны "Откуда" в строке документа на основании уже рассчитанных цен по заданному варианту стоимостного учета.

Описание функции

func УчЦенаОткуда(парВариантУчета :Настройки.ВариантСтоимостногоУчета) :Numeric

Функция имеет один единственный параметр парВариантУчета - вариант стоимостного учета, для которого требуется найти учетную цену со стороны "Откуда".

Результат функции - число.

Примеры

- Запрос на получение учетной цены "Откуда" в текущей позиции документа для варианта стоимостного учета "Управленческий учет":

Редактор формул реквизита

УчЦенаОткуда(%1)

Реквизит

УчЦенаОткуда(ВариантСтоимостногоУчета("Управленческий учет"))

Запись

Проверка

Записи

1	Управленческий учет
---	---------------------

Поле...

Помощь

Комментарий:

Функция "Цена" в реквизитах

Описание функции:

```
func Цена(  
    аПроцесс          :Данные.Процесс = nil;  
    аПоля              :String[] = nil;  
    аЗначения          :Variant[] = nil;  
    var аЕдИзм         :Справочники.ЕдиницаИзмерения[] = nil;  
    аБратьБазовуюЕдИзм :Integer = 0;  
    аДата              :Date = nil;  
const аИспользоватьСвободныеПрайсы :Logical = False  
) :Numeric;
```

Функция выполняет поиск цены в действующих [прайс-листах](#) для процесса, чье значение указано в параметре аПроцесс или для текущего документа, если этот параметр опущен.

С помощью параметров аПоля и аЗначения можно указать условие, по которому будут отбираться цены в прайс-листах. Оба эти параметра представляют собой массивы.

В параметре аПоля указываются имена полей прайс-листов, на которые нужно наложить ограничение. Если это поле шапки, то перед именем поля нужно указать символ "^" (например, "^Замок"), таким образом можно управлять из какого прайс-листа брать цену. К тому же существует возможность задавать цену в прайс-листах не за сам ресурс, а за группу ресурсов. Чтобы включить иерархию по какому - либо ресурсу (или по обоим), надо перед именем поля ресурса ("РесурсОткуда", "РесурсКуда") поставить знак "#" (например, Цена(['#РесурсОткуда', '#РесурсКуда'], [% 1, % 2]) - иерархия по обоим ресурсам). Порядок следования полей ресурса тоже имеет важность, если первым указано поле #РесурсОткуда, то поиск будет проводиться до более точного подходящего РесурсОткуда, а потом уточняется РесурсКуда.

Во втором массиве параметра аЗначения (в том же порядке) указываются значения ограничений. Ограничения задаются по равенству, т.е. окончательное условие получается следующего вида: "(Поле1 = Значение1) and (Поле2 = Значение2) and ..."

Параметр аЕдИзм указывает единицы измерения, в которых надо вернуть единицу измерения и/или валюту цены. Если параметр не задан, то в качестве единиц измерения будут неявно подставлены значения полей ЕдИзмОткуда и ЕдИзмКуда из позиции документа, для которой происходит вычисление функции Цена. Если опущено какое-либо одно значение этого массива, то вместо пропущенной единицы измерения будет возвращено значение единицы измерения из прайс-листа.

Параметр аБратьБазовуюЕдИзм указывает, надо ли производить поиск в базовых единицах измерения. Если цена не будет найдена в тех единицах измерения, которые заданы с помощью параметра аЕдИзм, то поиск будет проводиться в базовых для них единицах измерения. Если параметр равен -1, то поиск цены в базовых единицах измерения будет проходить для первой единицы измерения, указанной в массиве аЕдИзм, если значение равно 1 - то для второй единицы измерения, указанной в массиве аЕдИзм. Если параметр аБратьБазовуюЕдИзм равен 0, то поиск не производится.

Параметр аДата указывает дату, на которую необходимо искать цену в [прайс-листах](#). Если этот параметр не задан, дата берется равной начальной дате процесса, для которого происходит поиск цены.

При помощи логического параметра аИспользоватьСвободныеПрайсы можно указать, использовать свободные прайс-листы (значение True) или не использовать (значение False по умолчанию). Если установлено значение True и алгоритм не нашел цену в обычных прайс-листах, то поиск цены будет производиться в свободных прайсах ("Свободный прайс" - прайс-лист, который не привязан ни к какому процессу). Не желательно включать данный параметр без кэш-памяти.

Примеры

1. Запрос на получение цены из первого найденного прайс-листа, у которого совпадают значения полей РесурсОткуда и РесурсКуда с указанными в документе значениями, в единицах измерения ЕдИзмОткуда и ЕдИзмКуда.

Редактор формул

Редактор формул

Цена

Цена

Комментарий:

Реквизит

Запись

Проверка

Записи

Поле...

OK Отмена Помощь

- Запрос на получение цены из первого найденного прайс-листа, у которого совпадают значения полей РесурсОткуда и РесурсКуда с указанными в документе значениями, в единицах измерения, заданных реквизитами ЕдИзмКуда и ЕдИзмОткуда.

Редактор формул

Редактор формул

Цена(, , [%1, %2])

Цена(, , [ЕдИзмКуда, ЕдИзмОткуда])

Комментарий:

Реквизит

Запись

Проверка

Записи

1 ЕдИзмКуда

2 ЕдИзмОткуда

Поле...

Поле...

OK Отмена Помощь

- Запрос на получение цены из первого найденного прайс-листа, у которого значение поля РесурсОткуда совпадает со значением реквизита РесурсКуда и РесурсКуда со значением реквизита РесурсОткуда, в единицах измерения, заданных реквизитом ЕдИзмКуда, или в базовой единице измерения для значения этого реквизита и ЕдИзмОткуда.

Редактор формул

Редактор формул

Цена(, ['РесурсОткуда', 'РесурсКуда'], [%1, %2], [%3, %4], -1)

Цена(, ['РесурсОткуда', 'РесурсКуда'], [РесурсКуда, РесурсОткуда], [ЕдИзмКуда, ЕдИзмОткуда], -1)

Комментарий:

Реквизит

Запись

Проверка

Записи

1 РесурсКуда

2 РесурсОткуда

3 ЕдИзмКуда

4 ЕдИзмОткуда

Поле...

Поле...

Поле...

Поле...

OK Отмена Помощь