

Встроенный язык

- [Типы данных и операторы](#)
- [Общая информация](#)
- [Циклы и метки](#)
- [Модули, директивы препроцессора и компиляции](#)
- [Подпрограммы](#)
- [Исключения, внешний код](#)
- [Управление отображением и событиями платформы](#)
- [Механизм блокировок](#)
- [Механизм транзакций](#)

Типы данных и операторы

Переменные

Динамическая типизация.

```
a = -1;
a = "Один";
Сообщить(a);
//"Один"
```

ТипЗнч(элмас) - вывести тип переменной.

Можно явно определить имя переменной, ключевое слово Экспорт позволяет обращаться к переменной через контекст модуля. Без экспорта только для эстетики кода.

```
Перем <Имя переменной 1> [Экспорт]
```

Типы данных и преобразования типов

Примитивные типы данных

Тип	Описание
Null	Исключительно для определения отсутствующего значения при работе с базой данных.
Неопределено (Undefined)	<div>Пустое значение, не принадлежащее ни к одному другому типу.</div> <div>a = Неопределено; б = ?(а = Неопределено, 0, 1);//б = 0</div> <div>Прямое преобразование в булево: нельзя При сравнении любой тип не равен Неопределено</div> <div>a = 0; б = ?(а = Неопределено, 0, 1);//б = 1</div>

Тип	Описание
Число	Определены основные арифметические операции. Максимальная разрядность 38 знаков. Разделитель точка. 32 знака. Прямое преобразование в булево: любое ненулевое Истина, 0 Ложь Преобразование в строку: Строка()
Строка	Формат Unicode произвольной длины. В двойных кавычках. Многостроковый режим через Прямое преобразование в булево: нельзя <div>a = ""; b = ?(a, 0, 1); //вызовет ошибку</div> Преобразование в число: <div>Ч1 = Число(СтрокаЧ1); //вызовет исключение при невозможности</div> Преобразование в дату: <div>Дата("20211231123456"); //YYYYMMDDHHMMSS Дата("2021", "12", "24", "12", "34", "56");</div>
Дата	Строка цифр, заключенная в одинарные кавычки вида: 'ГГГГММДдччммсс' Прямое преобразование в булево: нельзя Контроль заполненности: Перем.ЗначениеЗаполнено() Пустая дата - Дата(1,1,1)
Булево	Значения данного типа имеют два значения Истина и Ложь
Тип	Используются для идентификации типов значений. Это необходимо для определения и сравнения типов.

Универсальные коллекции значений

Тип	Описание
-----	----------

<p>Массив</p>	<p>Аналог списка. Конструктор:</p> <div data-bbox="815 147 1485 409"> <pre> м1 = Новый Массив; //массив нулевой длины м2 = Новый Массив(<ФиксированныйМассив>); //создание массива из фиксированного массива м3 = Новый Массив(10, 3); //мас. из 10 эл-тов, каждый из которых - мас. из 3 эл-тов </pre> </div> <p>Методы:</p> <p>ВГраница - индекс верхней границы</p> <p>Вставить(Индекс, Значение) - Добавляет Значение в указанный индекс со смещением. Если указать больше границы, то будут созданы несуществующие значения с типом Неопределено и последним - Значение.</p> <p>Добавить - Добавляет элемент в конец массива.</p> <p>Количество</p> <p>Найти - Если найден - индекс, иначе неопределено.</p> <p>Очистить - Удаляет все значения из массива.</p> <p>Удалить - удаляет указанный индекс</p>
<p>Структура</p>	<p>Аналог словаря. Ключи только строковые.</p> <p>Конструктор:</p> <div data-bbox="815 925 1485 1088"> <pre> ст1 = Новый Структура; //пустая ст1 = Новый Структура("к1, к2", 1, 2) //структура с ключами к1, к2 и знач. 1 и 2 </pre> </div> <p>Доступ к элементу:</p> <div data-bbox="815 1167 1485 1281"> <pre> а["к1"]; а.к1; </pre> </div> <p>Перебор значений:</p> <div data-bbox="815 1359 1485 1621"> <pre> а = Новый Структура("к1, к2", 1, 2); Для каждого элмас из а Цикл Сообщить(элмас.Ключ + " " + элмас.Значение); КонецЦикла; </pre> </div> <p>Вставить(ключ, значение)</p> <p>Количество()</p> <p>Очистить() - удаляет все элементы</p> <p>Свойство(Ключ, [НайденноеЗначение]) - безопасный поиск по ключу</p> <p>Удалить(Ключ)</p>
<p>Соответствие</p>	<p>Как структура, только ключ может быть любого типа.</p>

Список значений	Как массив + строковое описание (Представление), Пометка и Картинка
-----------------	--

Таблица значений

Двумерная таблица.

```
тз = Новый ТаблицаЗначений;//Создание
таблицы
тз.Колонки.Добавить("Наименование", Новый
ОписаниеТипов("Строка"));
//Имя — идентификатор колонки
//Заголовок — представление колонки в
диалогах
//ТипЗначения — тип, может быть
произвольного типа;
//Ширина — ширина колонки в диалогах;
```

```
найдКолонка =
тз.Колонки.Найти("Наименование");//поиск
колонки
Если найдКолонка = Неопределено Тогда
    Сообщить("Колонка не найдена!");
КонецЕсли;
```

```
//перебор колонок
Для каждого Колонка Из тз.Колонки Цикл
    Сообщить(Колонка.Имя);
КонецЦикла;
```

```
Вставить() — Вставляет новую колонку в
указанную позицию коллекции
Добавить() — Добавляет новую колонку в конец
коллекции
Количество() — Возвращает количество колонок в
коллекции
Найти() — Ищет колонку в коллекции по имени
Очистить() — Удаляет все колонки из коллекции
Сдвинуть() — Сдвигает колонку влево или вправо
Удалить() — Удаляет колонку из коллекции
```

```
СтрокаТЧ = тз.Добавить();
СтрокаТЧ.Наименование = "Стул деревянный";
тз = СтрокаТЧ.Владелец();
тз.Удалить(тз.Индекс(СтрокаТЧ));
```

```
Для Каждого СтрокаТЧ Из тз Цикл
    ИндСтроки = тз.Индекс(СтрокаТЧ);
КонецЦикла;
```

Фиксированный (массив, соответствие, структура)	Аналогичный объект только для чтения.
Хранилище значений	Недоступен напрямую в управляемой форме. Нужно городить временное хранилище.

Объекты

Респект и уважуха автору AlexO за разъяснение сущности взаимодействия в 1С [ссылка на форум](#)

В 1С нет настоящих "объектов" ООП. Эти товарищи обозвали словом "объект" ссылку на словарь значений. В 1С введено внутреннее понятие КоллекцияЗначений - и это не аналог контейнера объектов из ООП (который и сам определяет поведение входящих объектов, и дает доступ напрямую к ним - к их свойствам, методам, данным, событиям и т.д.), а набор ссылок на другие объекты, и из коллекции, если не получен "вложенный" элемент-объект (например, через метод НабораЗаписей НаборЗаписей.Прочитать()), нельзя напрямую получить свойства и методы элементов коллекции, а только - получить "объекты" коллекции, и уже обходом или обращением к элементу коллекции - работать со свойствами и методами "вложенных" объектов. Объект РегистрСведений не содержит объект РегистрСведенийНаборЗаписей, а НаборЗаписей не содержит объекты РегистрСведенийЗапись. Для работы с каждым вложенным уровнем нужно заново получать объекты этого нового уровня вложенности. Собственно, вся канитель "не могу получить данные объекта там-то", "не могу получить доступ к процедуре тут-то", "не видна переменная экспортная такая-то" и прочие невообразимые и множественные ограничения платформы - именно из-за наборов не связанных напрямую друг с другом "объектов", которых нужно каждый раз "получать", извлекать данные, и которым нужно каждый раз указывать - что мы от них хотим.

Тип конфигурации ...Объект (например СправочникОбъект) -

Поскольку переписывать лень, далее используется слово Объект в понимании 1С.

Внешних модулей нет. Использование процедур в разных проектах - только копирование/вставка. Придется смириться.

Обращение к свойствам через точку или <Объект>["имя свойства"]

```
Менеджер = Справочники["Менеджеры"];
```

```
Менеджер = Справочники.Менеджеры;
```

Обращение к методам - через точку. Дополнение:

Категорический запрет на использование в запросах "двойного разыменования"
(Объект.Свойство<содержащее СсылкуНаДругойОбъект>.СвойствоДругогоОбъекта) -
только явное соединение таблиц через СОЕДИНЕНИЕ...

Условные операторы

?(<Логическое выражение>, <Выражение 1>, <Выражение 2>)

```
а = Истина;
```

```
б = ?(а = Ложь, 0, 1); // б = 1
```

Если <Логическое выражение> Тогда

// Операторы

[ИначеЕсли <Логическое выражение> Тогда]

// Операторы

[Иначе]

// Операторы

КонецЕсли;

Разное

```
Выполнить(<Строка>);
```

```
Выполнить("Сообщить(а)");
```

Исполняет строковое представление команды.

Общая информация

В данном разделе предоставлена информация, максимально связанная только со встроенным языком 1С. Информация об остальных объектах языка представлена по необходимости.

Предварительная подготовка конфигурации.

- Создать пустую конфигурацию [Создание или добавление информационной базы](#)
- Открыть конфигуратор, открыть конфигурацию
- Создать справочник Менеджеры, без дополнительных данных.

Проверка встроенного языка 1С будет проводиться по следующему алгоритму: [Создание обработки](#)

Циклы и метки

В циклах могут использоваться операторы Прервать; и Продолжить;

Цикл for:

```
Для <Имя переменной> = <Выражение 1> По <Выражение 2> Цикл
// Операторы
КонецЦикла;
```

Обход коллекции:

```
Для каждого <Имя переменной 1> Из <Имя переменной 2> Цикл
// Операторы
КонецЦикла;
```

Цикл While

```
Пока <Логическое выражение> Цикл
// Операторы
КонецЦикла;
```

В языке есть метки.

```
Перейти ~ВыходИзДвойногоЦикла;

~ВыходИзДвойногоЦикла:
//продолжение
```


Модули, директивы препроцессора и компиляции

В 1C ебанутые правила, связанные с модульностью. Понять и простить.

Модули служат для хранения кода. Несколько регламентированных точек размещения модулей, однако точка размещения кода для решения конкретной задачи регламентирована на уровне "можно и здесь, а можно и где-то там", поэтому финальная точка размещения зависит от фазы луны в момент создания кода разработчиком. То есть правила вроде есть, но их можно не исполнять. Однако важный нюанс: в каждой точке размещения доступны разные глобальные переменные, поэтому при написании кода набор доступных переменных нужно обязательно уточнять. Часто в интернетах приводят код без указания размещения модуля, потом при вопросе "А у меня не работает" сначала пара страниц троллинга типа они пиздец какие охуевшие спецы, потом кто-то снисходит до лошары. Хотя данное разделение не несет практической цели для разработчика конфигурации кроме искусственного усложнения и скорее артефакт изначально упрощенной архитектуры. Как деление подпрограмм на процедуры и функции.

Интересная [статья](#) по модулям, однако есть ряд недосказанных моментов.

Для подпрограмм всех модулей актуальны директивы препроцессора и компиляции. Модули можно сгруппировать по следующим правилам (группировка субъективная на основании изучения различных источников):

Группа	Размещение модулей	Базовое назначение	Подробное описание
Общие	Конфигурация - Общие - Общие модули	Хранение общего кода, доступного из остальных модулей. Настроить обработку событий нельзя.	В этой статье.

Группа	Размещение модулей	Базовое назначение	Подробное описание
Модули конфигурации	Конфигурация - Свойства. Модуль приложения Модуль внешнего соединения Модуль сеанса	Обработка общих событий платформы. Размещение подпрограмм обработки событий платформы. Например запуска приложения или внешнего вызова.	Управление отображением и событиями платформы
Модули классов	Конкретный класс. Модуль формы Модуль объекта Модуль менеджера	События, связанные с конкретным классом. Например нажатие кнопки "Показать группы" в окне списка справочника Номенклатура.	Объекты конфигурации

Структура модуля:

Раздел определения переменных - от начала текста модуля до первого оператора Процедуры, Функции, или любого исполняемого оператора. Только операторы объявления переменных Переменных.

Раздел процедур - от первого Процедура / Функция до любого исполняемого оператора вне тела описания процедур или функций.

Раздел основной программы - от первого исполняемого оператора вне тела последней процедуры или функции до конца модуля. Могут находиться только исполняемые операторы. Исполняется в момент инициализации модуля. Обычно в разделе основной программы имеет смысл размещать операторы инициализации переменных какими-либо конкретными значениями, которые необходимо провести до первого вызова любой из процедур или функций модуля.

Свойства модуля:

- Способ обращения к подпрограммам
 - «Глобальный». Возможность вызова подпрограмм без указания имени модуля. Имена должны быть уникальными в разрезе всего глобального контекста. Компилируются при старте системы. Чем больше таких модулей, тем медленнее программа будет стартовать. Если не указан, то компиляция будет выполняться в момент первого обращения к нему.
- Область видимости
 - «Клиент». Возможность исполнения подпрограмм модуля на клиенте;
 - «Сервер». Возможность исполнения подпрограмм модуля на сервере;
 - «Внешнее соединение». Возможность исполнения подпрограмм модуля через подключение внешнего источника;
- Возможности со стороны подпрограмм модуля
 - «Вызов сервера». Возможность подпрограмм модуля вызывать сервер, выполняясь на клиенте;
 - «Привилегированный». При работе кода процедур модуля не проверяет права доступа. Вызвать общий модуль с такой настройкой можно только на сервере.

Настройки «Клиент» и «Внешнее соединение» будут сброшены.
Это необходимо, если требуется массовая обработка данных. Контроль прав доступа увеличивает время обращения к базе данных .

- Кэширование
 - Для не глобальных Общих модулей. Варианты: «Повторное использование». Варианты: «Не использовать», «На время сеанса», «На время вызова». При многократном вызове одной процедуры система может использовать рассчитанные ранее данные в рамках процедуры (вызов) или жизни всего сеанса (запуска 1С).
- Цель – ускорить повторные вызовы. Имеет смысл только для тех функций, результат которых зависит исключительно от входных параметров. Если выбрано значение соответствующего параметра На время вызова, то кэш будет действовать до тех пор, пока работает та процедура, откуда был сделан вызов метода Общего модуля. Если выбрано значение На время сеанса, то условно считается, что кэш будет действовать, пока пользователь работает. Существуют временные ограничения, очистка кэша происходит автоматически через 20 минут после попадания значения в кэш. Использовать осознанно.

Директивы препроцессора и компиляции

Есть два режима, определяются при создании конфигурации: на локальном ПК (файловый) и на сервере 1С (клиент-серверный вариант). Файловый вариант объединяет в себе и код клиента, и код сервера, поэтому смысла от деления кода при помощи директив препроцессора нет.

При запуске конфигурации на выполнение производится загрузка и компиляция конфигурации. Алгоритм компиляции:

- Экземпляры всех общих модулей создаются как на серверной, так и на клиентской стороне. Препроцессор используя директивы препроцессора и области видимостей модулей создает несколько слегка отличающихся копий (в соответствии с количеством платформ) кода. Директивы компиляции игнорируются. По умолчанию (без директив) все подпрограммы попадают во все копии. При наличии, код физически вырезается, и другая платформа не видит чужой код. Нужно для уменьшения объема кода и ускорения работы каждого из блоков.
- Код компилируется для каждой платформы. Используя директивы компиляции, формируются алгоритмы взаимодействия. Например, при вызове из подпрограммы &НаКлиенте подпрограммы &НаСервере форма упаковывается в контейнер (!), переправляется по каналу tcp\ip на сервер, распаковывается сервером, выполняется код целевой процедуры, форма запаковывается обратно в контейнер, передается на клиент, распаковывается клиентом, отображается на экране. В случае с "..БезКонтекста" платформа выполняет код "налегке", т.е. на сервер передаются только параметры функции, а обратно прилетает только результат выполнения. Директивой по умолчанию является &НаСервере.

Директивы препроцессора

При компиляции блоки распределяются в соответствии с директивами. Для указания разрешения использования процедур и функций общих модулей и модулей объектов используют инструкции препроцессора.

Синтаксис:

```
#Если <Логическое выражение> Тогда
#ИначеЕсли <Логическое выражение> Тогда
#Иначе
#КонецЕсли
```

<Логическое выражение> = [НЕ] <Символ препроцессора> [<Булева операция> [НЕ]
<Символ препроцессора> [<Булева операция> [НЕ] <Символ препроцессора>...]
<Символ препроцессора> = {НаКлиенте | НаСервере |
ТолстыйКлиентОбычноеПриложение | ТолстыйКлиентУправляемоеПриложение | Клиент |
Сервер | ВнешнееСоединение }
<Булева операция> = {И | ИЛИ}

Можно использовать И (AND), ИЛИ (OR), НЕ (NOT) Регистр букв не имеет значения.

Области

#Область, #КонецОбласти Нужны только для визуального сворачивания блоков кода. Могут быть вложенными.

```
#Область [<Имя области>]
#КонецОбласти
```

Условия

#Если (#If), #Тогда (#Then), #ИначеЕсли (#Elseif), #Иначе (#Else), #КонецЕсли (#EndIf)
Можно организовывать выполнение различных процедур и функций на сервере приложения или на клиентском месте. Для того, чтобы процедура присутствовала и была вызвана на стороне сервера, фрагмент кода должен выглядеть следующим образом:

```
#Если Сервер Тогда
Процедура Проц1() Экспорт
КонецПроцедуры
#КонецЕсли
```

Если блок #Если Сервер Тогда ... #КонецЕсли включает только часть процедуры, то процедура будет присутствовать как на стороне клиента, так и на стороне сервера. Только на клиентской стороне она будет без той части, которая заключена в блок, поэтому

результат выполнения процедуры может зависеть от того, где обрабатывается вызов этой процедуры.

Для выполнения на клиентском месте в обычном и управляемом режиме:

```
#Если НаКлиенте Тогда
```

```
#КонецЕсли
```

Точки исполнения

Инструкция препроцессора НаКлиенте определена для всех клиентских приложений. Для тонкой подстройки модуля под конкретное клиентское приложение дополнительно введены инструкции ТолстыйКлиентОбычноеПриложение, ТолстыйКлиентУправляемоеПриложение, ТонкийКлиент и ВебКлиент, которые определены в соответствующих приложениях.

В обычном клиенте в обычном и управляемом режиме доступны НаКлиенте, Клиент, ТолстыйКлиентОбычноеПриложение, ТолстыйКлиентУправляемоеПриложение, НаСервере, Сервер.

В файловом варианте инструкции препроцессора #Если Сервер..., #Если Клиент..., #Если ТолстыйКлиентОбычноеПриложение или #Если ТолстыйКлиентУправляемоеПриложение... определены всегда, поэтому экземпляр кода будет присутствовать всегда.

В тонком клиенте доступны – ТонкийКлиент, НаКлиенте, Клиент.

На серверной части тонкого клиента – Сервер, НаСервере.

Во внешнем соединении – ВнешнееСоединение, НаСервере, Сервер.

Директивы компиляции

Каждая процедура и функция модуля формы, модуля команды и общего модуля управляемого приложения предваряется директивой компиляции, определяющей среду исполнения данной процедуры. Директива предваряется символом "&".

Запросы между клиентом и сервером за счет директив компиляции корректно формируются только внутри модуля обработчика. Во внешнем модуле нельзя вызвать подпрограммы разных контекстов. Нужно сначала перейти в соответствующий контекст в модуле, затем вызвать подпрограмму из общего модуля.

&НаКлиенте — определяет клиентскую процедуру (функцию); выполняется в среде клиентского приложения. В такой процедуре доступен клиентский контекст формы и вызовы любых процедур модуля.

&НаСервере — определяет серверную процедуру (функцию); выполняется в среде серверного приложения. В такой процедуре доступны данные формы, доступен серверный контекст формы и вызовы серверных и серверных внеконтекстных процедур модуля. При вызове такой процедуры данные формы будут передаваться с клиента на сервер и обратно (по окончании вызова).

&НаСервереБезКонтекста — определяет серверную процедуру (функцию), исполняемую на сервере вне контекста формы. Переменные не могут быть внеконтекстными. В таких методах недоступен контекст формы (включая данные формы). Допустимыми являются вызовы только других внеконтекстных методов. При вызове этих методов не выполняется передача данных формы на сервер и обратно. Применение внеконтекстных методов позволяет существенно уменьшить объем передаваемых данных при вызове серверной процедуры из среды клиентского приложения;

&НаКлиентеНаСервереБезКонтекста — определяет процедуру (функцию), исполняемую в модуле формы на клиенте и на сервере, не имеющую доступа к контексту формы, данным формы, переменным, но имеющую доступ к процедурам и функциям общих модулей – серверных, не глобальных и серверных и клиентских одновременно. Сама процедура (функция) доступна для клиентский, серверных контекстных и неконтекстных процедур и функций модуля формы. Из серверных внеконтекстных методов формы допускается вызов серверных методов общих модулей;

&НаКлиентеНаСервере — определяет процедуру (функцию), исполняемую в модуле команды, выполняемую на клиенте и на сервере, имеющую доступ к процедурам и функциям общих модулей – серверных, не глобальных и серверных и клиентских одновременно, не имеющую доступ к переменным. Сама процедура (функция) доступна для клиентских серверных процедур и функций модуля команды.

Серверная процедура (функция), исполняемая вне контекста формы, (внеконтекстная) выполняется в среде серверного приложения. В такой процедуре (функции) недоступен контекст формы (включая данные формы). Допустимыми являются вызовы только других внеконтекстных процедур (функций). При вызове этих процедур (функций) не выполняется передача данных формы на сервер и обратно. Применение внеконтекстных процедур (функций) позволяет существенно уменьшить объем передаваемых данных при вызове серверной процедуры (функции) из среды клиентского приложения.

В модуле команды предопределенная процедура-обработчик ОбработатьКоманду должна предваряться директивой &НаКлиенте, так как выполнение команды происходит в клиентском приложении.

Модуль формы

В модуле формы доступны директивы компиляции – &НаКлиенте, &НаСервере, &НаСервереБезКонтекста, &НаКлиентеНаСервереБезКонтекста.

Модуль команды

В модуле команды доступны директивы компиляции – &НаКлиенте, &НаСервере, &НаКлиентеНаСервере.

Общий модуль

В общем модуле доступны директивы компиляции – &НаКлиенте, &НаСервере.

Общие модули.

Описание подпрограмм, вызываемых из различных мест. Для доступности подпрограмм извне необходимо ключевое слово Экспорт. То есть модули-то общие, но есть нюанс.

Даже при средней сложности конфигурации, подпрограммы нужно группировать в разные Общие модули. Для удобства имена Общих модулей должны отражать содержание описываемых в них процедур. При создании Общего модуля, правилом хорошего тона считается не указывать директивы компиляции. Т.е. доступность процедур и функций должна определяться свойствами самого модуля. При таком подходе в отдельных Общих модулях будут располагаться клиентские процедуры, и в отдельных Общих модулях – процедуры серверные. В названиях общих модулей рекомендуется это указывать. Например: РегламентныеПроцедурыСервер, РегламентныеПроцедурыКлиент.

Нельзя описывать глобальные переменные модуля и раздел основной программы. Т е все внешние переменные должны передаваться в подпрограммы извне.

Доступность модулей зависит от типа приложения и режима работы (Клиент, Сервер).

В отличие от остальных типов модулей (в них по умолчанию директива компиляции &НаСервере), если директиву компиляции для подпрограммы не указывать, то она будет скомпилирована во всех контекстах, определенных для модуля. Будет сделано несколько копий подпрограмм. Выбор конкретного скомпилированного экземпляра зависит от места вызова процедуры (по правилу ближайшего вызова). При этом следует учитывать, что код такой процедуры должен быть написан с учетом его доступности во всех определенных для модуля контекстах.

Модули с несколькими флагами компиляции на практике используются крайне редко. Это некоторые общие действия, доступные как на Клиенте, так и на Сервере. Обычно, это какие-то простейшие вычисления.

Подпрограммы

Функции возвращают одно значение, процедуры выполняют операции и могут изменять переданные параметры. Переменные, объявленные в теле подпрограммы, локальные.

Общие определения:

Знач - Следующий за ним параметр передается по значению. По умолчанию параметр процедуры передается по ссылке.

<Парам1>, ..., <ПарамN> - Список формальных параметров, разделяемых запятыми, может быть пуст.

=<ДефЗнач>- Установка значения параметра по умолчанию.

Экспорт - Данная процедура доступна из других программных модулей.

Описание процедуры:

```
Процедура <Имя_процедуры>([[Знач] <Парам1> [=<ДефЗнач>], ... ,[Знач] <ПарамN>
[=<ДефЗнач>]])[Экспорт]
// Операторы;
[Возврат;]
КонецПроцедуры
```

Параметры:

Возврат - Завершает выполнение процедуры и осуществляет возврат. Не обязательно.

КонецПроцедуры - Обязательное ключевое слово, обозначающее конец исходного текста процедуры, завершение выполнения процедуры.

Описание функции:

```
Функция <Имя_функции>([[Знач] <Парам1> [=<ДефЗнач>], ... ,[Знач] <ПарамN>
[=<ДефЗнач>]])[Экспорт]
// Операторы ;
Возврат <Возвращаемое значение>;
КонецФункции
```

Возврат <Возвращаемое значение> Завершает выполнение функции и возвращает значение.

Возврат; Завершает выполнение процедуры.

В качестве возвращаемого значения может выступать выражение или переменная, значение которого содержит результат обращения к функции.

Функции отличаются от процедур только тем, что возвращают <Возвращаемое значение>. Конец программной секции функции определяется по ключевому слову КонецФункции.

Вызов функции можно записывать как вызов процедуры, т.е. допускается не принимать от функции возвращаемое значение.

Исключения, внешний код

Попытка

ПроверяемыйПараметр = Неопределено;

Если ПроверяемыйПараметр Тогда

//код, который не будет выполнен

КонецЕсли;

Исключение

Сообщить(ОписаниеОшибки());

КонецПопытки;

Генерация исключения

ВызватьИсключение [<Выражение>]; //описание - результат вычисления Выражение.

Выполнить(""" выполнить фрагмент кода, который передается ему в качестве строкового значения. Использовать аккуратно.

Управление отображением и событиями платформы

Модули конфигурации

Модуль сеанса Выполняется при старте системы «1С:Предприятие» в момент загрузки конфигурации. Предназначен для инициализации параметров сеанса и отработки действий, связанных с сеансом работы. Исполняется в привилегированном режиме. Установка параметров сеанса выполняется в обработчике события УстановкаПараметровСеанса(). Исполнение модуля сеанса происходит до начала исполнения модуля управляемого приложения и модуля внешнего соединения. Может содержать только определения процедур и функций, может использовать процедуры из общих модулей конфигурации и не содержит экспортируемых процедур и функций.

Модуль управляемого приложения Автоматически выполняется при загрузке конфигурации, при старте системы в режимах тонкого клиента, веб-клиента, толстого клиента в режиме управляемого приложения.

Предназначен для отработки действий, связанных с сеансом работы конечного пользователя (прежде всего обработки начала и окончания сеанса работы). Недоступен для процедур, работающих на сервере. Процедуры и функции модуля управляемого приложения, а также переменные, для которых в заголовке указано ключевое слово Экспорт, являются доступными в неглобальных клиентских общих модулях, клиентских процедурах и функциях модуля команды, клиентских процедурах и функциях модуля управляемой формы.

В контексте модуля управляемого приложения доступны часть глобального контекста, которая может исполняться в управляемом приложении; экспортируемые процедуры и функции любых клиентских общих модулей; экспортируемые процедуры и функции серверных неглобальных общих модулей, у которых установлено свойство Вызов сервера.

Модуль внешнего соединения Процедуры-обработчики событий при старте и окончании работы системы в режиме внешнего соединения (COM-соединения). Возможно объявление переменных, а также объявление и описание процедур и функций, которые будут доступны для внешнего приложения. Объекты, доступные извне через COM-соединение:

- экспортируемые переменные и процедуры/функции модуля внешнего соединения;
- экспортируемые переменные и процедуры/функции общих модулей;
- включение и исключение модулей целиком выполняются с помощью установки свойств общих модулей;
- включение и исключение фрагментов общих модулей выполняются с помощью инструкций препроцессора;

- глобальный контекст «1С:Предприятия».

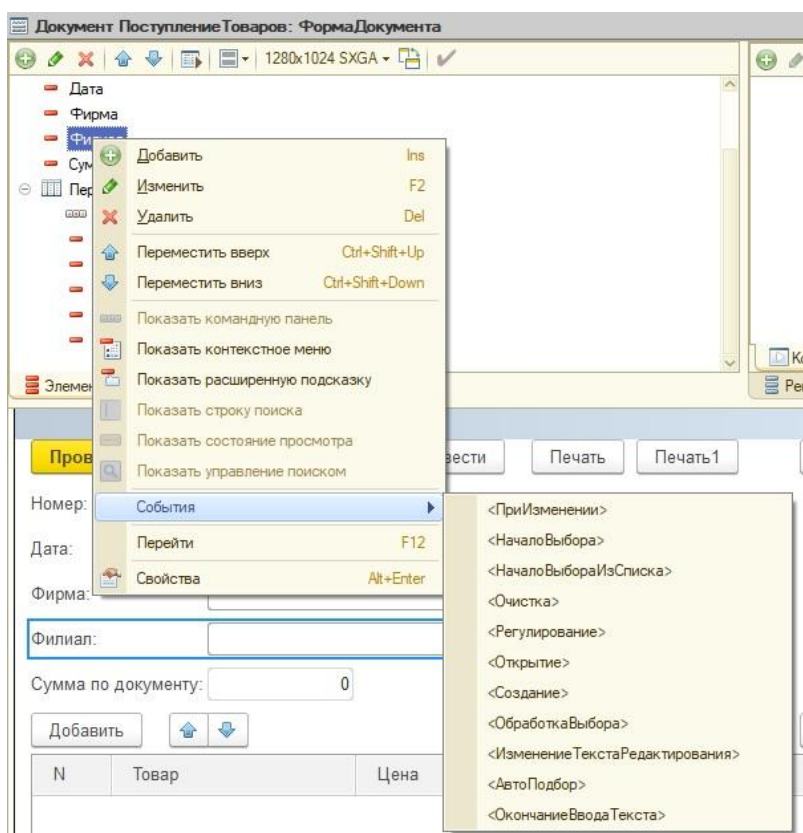
Модуль присутствует только в сессии внешнего соединения, нет пользовательского интерфейса.

События платформы

Основные направления общей настройки платформы: внешний вид и события. Переход в элементы настройки производится через окно свойств конфигурации.

События формы

Для настройки обработки событий формы необходимо сначала создать форму вместо формы по умолчанию. ПКМ на любом добавленном элементе формы -> События выведет список доступных обработчиков событий. Или в свойствах этого элемента.



При клике будет создан обработчик события. Для обращения к активному в модуле формы используется Элементы, все хранится в Объект. Пример автоматического пересчета итоговой суммы при изменении цены или количества в табличной части:

```
&НаКлиенте
```

```
Процедура ПересчитатьИтоговуюСумму()
```

```
□ТабЧасть = Объект.ПереченьТоваров;
```

```
□ФиналСумма = 0; □
```

```
□Для каждого Товар из ТабЧасть Цикл
```



```
□□ФиналСумма = ФиналСумма + Товар.Сумма;
```

```
□КонецЦикла;
```

```
    Объект.СуммаПоДокументу = ФиналСумма;
```

```
КонецПроцедуры
```

```
&НаКлиенте
```

```
Процедура ПереченьТоваровКоличествоПриИзменении(Элемент)
```

```
□СтрТабЧасти = Элементы.ПереченьТоваров.ТекущиеДанные;
```

```
□СтрТабЧасти.Сумма = СтрТабЧасти.Цена * СтрТабЧасти.Количество;
```

```
□ПересчитатьИтоговуюСумму();
```

```
КонецПроцедуры
```

```
&НаКлиенте
```

```
Процедура ПереченьТоваровЦенаПриИзменении(Элемент)
```

```
□СтрТабЧасти = Элементы.ПереченьТоваров.ТекущиеДанные;
```

```
□СтрТабЧасти.Сумма = СтрТабЧасти.Цена * СтрТабЧасти.Количество;
```

```
□ПересчитатьИтоговуюСумму();
```

```
КонецПроцедуры
```

Если будет несколько документов с необходимостью пересчитывать суммы, возможно стоит сделать процедуру НаКлиенте в общих модулях, передавая туда табличную часть, текущие данные, поле для общей суммы,

Программное создание обработчика события.

ДобавитьОбработчик <Событие>, <ОбработчикСобытия>;

Добавляет обработчик события. При добавлении обработчика события производится проверка соответствия числа параметров события числу параметров метода, назначаемого в качестве обработчика.

УдалитьОбработчик <Событие>, <ОбработчикСобытия>;

Удаляет обработчик события. При удалении обработчика события производится проверка соответствия числа параметров события числу параметров метода, назначенного в качестве обработчика.

Механизм блокировок

Хорошая [статья по блокировкам на ИТС](#), но слегка устаревшая, в 8.3.14 алгоритм пессимистичной блокировки изменен.

[Очень интересная статья](#), исследование блокировок.

[Ускорение работы при высокой нагрузке](#), для погружения.

В простых словах и с примерами [описание](#) управляемой блокировки

Механизмы и типы блокировок.

Критичная тема для понимания работы платформы.

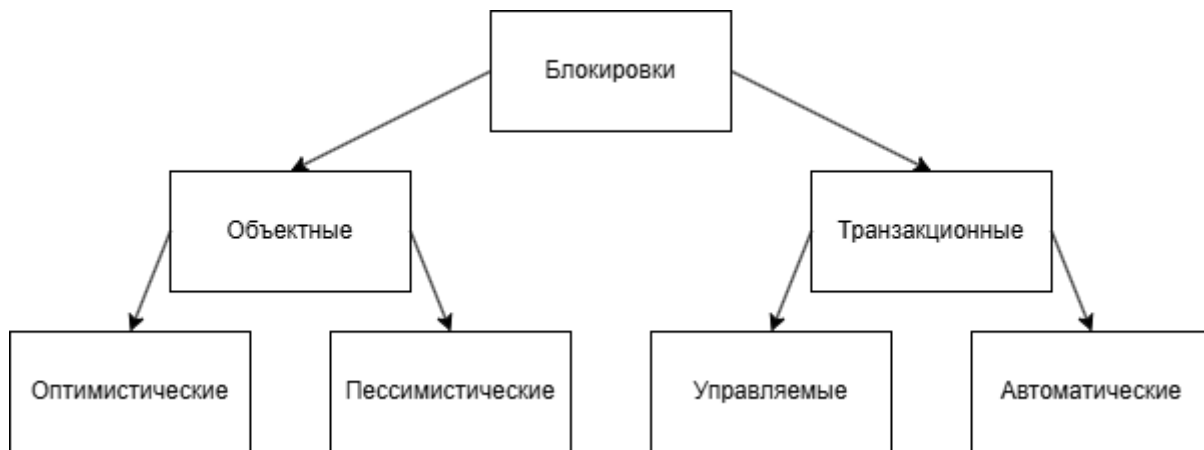
Механизм объектной блокировки - механизм обеспечения целостности объекта (элемент справочника, документ, ...) средствами платформы 1С. В большинстве случаев это связано с интерактивной работой пользователей в формах: редактирование существующих объектов, удаление, создание новых и др.

Механизм транзакционной блокировки - механизм обеспечения целостности и непротиворечивости данных средствами ядра СУБД на уровне транзакций.

Отличия заключается в элементе контроля и выбора точки блокировки. К тому же, у разных СУБД разные уровни блокировок (на уровне записей или на уровне таблиц). Например, элемент справочника состоит из реквизита Наименование и табличной части Характеристики с названием и значением. С точки зрения базы данных, в ней создается основная таблица с ключом, наименованием, и дополнительная таблица Характеристики с индексом, названием, значением и ссылкой на владельца.

Блокировка на уровне объекта подразумевает наличие в БД третьей таблицы, в которой указывается, что Пользователь1 редактирует объект1. При попытке Пользователем2 изменить объект1, в операции будет отказано и до уровня старта изменений в данных дело не дойдет. Но здесь на первый план выходит алгоритм добавления и удаления данных из третьей таблицы.

Блокировка на уровне транзакции подразумевает отсутствие третьей таблицы, и при изменении данных в соответствии с логикой СУБД (либо запись об объекте1 в таблице1 и соответствующих записей в таблице2, либо целиком обеих таблиц), после успешного завершения блокировки снимаются.



- Объектная Оптимистическая. Построена на анализе номера версии объекта, хранящейся в базе данных и номера версии, помещенной в память компьютера в момент считывания данных из информационной базы. Если при записи объекта номера версий на форме и на сервере отличаются, то будет выдано предупреждение о том, что версия объекта изменилась или он был удален. Происходит автоматически при сохранении данных через механизмы формы (не через запрос) или при чтении данных (разные типы блокировок).
- Объектная Пессимистическая неявная (в форме). В тот момент, когда пользователь начинает модификацию объекта (имеется в виду в первый раз нажал на кнопку Сохранить, а не момент открытия или момент изменения данных в поле) в форме, расширение формы устанавливает пессимистическую блокировку. Если после этого другой пользователь попытается выполнить редактирование, ему будет выдано сообщение о невозможности блокировки объекта. Снимается после закрытия формы объекта либо через минуту после сохранения данных. Для отключения пессимистической блокировки в управляемых формах в свойстве основного реквизита надо снять флаг «Сохраняемые данные». Данный флаг определяет будет ли при интерактивном редактировании блокироваться данные основного реквизита, или нет.
- Объектная пессимистическая блокировка явная (в коде)
 - Для управляемых форм. Используется если необходимо просто заблокировать данные на сервере пока выполняются преобразования данных на форме. Для работы с блокировками из управляемой формы без вызова сервера используются методы «ЗаблокироватьДанныеФормыДляРедактирования()» и «РазблокироватьДанныеФормыДляРедактирования()». Как и в случае неявной, контролируются флагом «Сохраняемые данные». Пессимистическая блокировка хранится в сервисе блокировок DataEditLockService кластера 1C. Увидеть блокировку конкретного объекта можно в консоли кластера в разделе Кластеры -> Информационные базы -> Блокировки
 - Без формы Методы «Заблокировать()», «Разблокировать()» и «Заблокирован()», используются для объектов базы данных, существуют только на сервере. Метод «Заблокирован()» не может быть использован для проверки, заблокирован ли вообще объект базы данных, например, другими пользователями. Для этого следует использовать метод «Заблокировать()» в попытке.

```
&НаСервереБезКонтекста
```

```
...
```

```
Попытка
```

```
□тестОбъект.Заблокировать();
```

```
□// что-то делаем с объектом
```

```
    тестОбъект.Разблокировать();
```

```
Исключение
```

```
□Сообщить("Не удалось заблокировать тест");
```

```
КонецПопытки;
```

- Автоматические. Ответственность за блокировки полностью лежит на СУБД. Это облегчает работу, но создание информационной системы для большого количества пользователей на автоматических блокировках нежелательно (особенно для СУБД PostgreSQL, Oracle BD, т.к. при модификации данных они полностью блокируют таблицу).
- Управляемые. В управляемом режиме есть возможность использовать менеджер транзакционных блокировок данных. В этом режиме система использует гораздо более низкий уровень изоляции транзакций для MS SQL Server и IBM DB2, и блокировку на уровне записей для PostgreSQL. Это позволяет повысить степень параллельности работы пользователей.

Длительность блокировок

Детали установки блокировок

Настройка объектов конфигурации

Режим управления блокировками данных настраивается в свойствах объекта конфигурации, раздел Режим управления блокировками данных. Может быть Управляемый или Автоматический.

Шаблоны кода

Получить версию объекта на клиенте и на сервере.

```
&НаСервереБезКонтекста
```

```
Процедура ПолучитьВерсиюОбъектаНаСервере(ТекОбъект)
```

```
□Сообщить(ТекОбъект.ВерсияДанных);
```

КонецПроцедуры

&НаКлиенте

Процедура ПолучитьВерсиюОбъекта(Команда)

□ПолучитьВерсиюОбъектаНаСервере(Объект.Ссылка);

□Сообщить(Объект.ВерсияДанных);

КонецПроцедуры

Проверка необходимости обновления.

&НаСервереБезКонтекста

Функция ВернутьВерсиюДанныхНаСервере(ТекОбъект)

□ Возврат ТекОбъект.ВерсияДанных;

КонецФункции

&НаКлиенте

Процедура ОболочкаОбработчика()

□ВерсияНаСервере = ВернутьВерсиюДанныхНаСервере(Объект.Ссылка);

□Если ВерсияНаСервере <> Объект.ВерсияДанных Тогда

□□ЭтаФорма.Прочитать();

 ПроверкаЭлементовНаВидимость();

□КонецЕсли;

КонецПроцедуры

Проверить, заблокирован ли объект

// Попытка установки блокировки

Объект = Номенклатура.ПолучитьОбъект();

Попытка

□Объект.Заблокировать();

Исключение

□// Данные объекта уже заблокированы.□

КонецПопытки;

Проверка предположений.

Проверка факта запрета блокировки другим объектом.

&НаСервереБезКонтекста

Процедура ПроверкаПериодаБлокировкиНаСервере()

□тест = Справочники.ДляТранзакций.НайтиПоНаименованию("первый");

□тест2 = Справочники.ДляТранзакций.НайтиПоНаименованию("первый");

□тестобъект = тест.ПолучитьОбъект();

□тестобъект2 = тест2.ПолучитьОбъект();

□Попытка

□□тестобъект.Заблокировать();

□□Сообщить("Объект тест заблокирован");

□Исключение

□□Сообщить("Не удалось заблокировать тест");

□КонецПопытки;

□Попытка

□□тестобъект2.Заблокировать();

□□Сообщить("Объект тест2 заблокирован");

□Исключение

□□Сообщить("Не удалось заблокировать тест2");

□КонецПопытки;

КонецПроцедуры

Объект тест2 заблокировать не удалось.

Блокировка держится до конца транзакции

Механизм транзакций

[Статья на ИТС](#)

Транзакция – это последовательность действий, переводящая базу данных из одного целостного состояния в другое целостное состояние.

Свойства транзакции:

- Атомарность (неделимость). После завершения транзакции все данные должны быть согласованы. Даже при простом добавлении записи может произойти рассогласование, и необходимо это обрабатывать.
- Изоляция. Это свойство обеспечивает параллельную работу пользователей и предотвращает порчу общих данных. Например, чтобы не вышла ситуация, когда 2 пользователя меняют один и тот же документ и тем самым перестиривают данные друг друга. Блокировки выступают как средство обеспечения изоляции транзакции.

Особенности транзакции:

- не поддерживаются вложенные транзакции. Так эта особенность официально звучит, однако интересно: далее в тексте встречается термин Вложенная транзакция. Понять и простить. Под этим подразумевается, что вместо дерева транзакций мы имеем дело с одноуровневым списком, и все должно быть выполнено. В случае возникновения и подавлении ошибки где-то внутри "вложенной" транзакции, все равно вся транзакция будет считаться битой.

Ситуация в коде	Ожидание	Реальность
Начало транзакции 1 Вызов функции А Начало транзакции 2 Конец транзакции 2 Вызов функции Б Начало транзакции 3 Ошибка, обработка исключения Конец транзакции 3 Конец транзакции 1	Начало транзакции 1 Начало транзакции 2 Конец транзакции 2 Начало транзакции 3 Ошибка, обработка исключения Конец транзакции 3 Конец транзакции 1 Транзакция 1 должна завершиться правильно, т.к. ошибка в транзакции 3 обработана	Начало транзакции 1 Начало транзакции 2 Конец транзакции 2 Начало транзакции 3 Ошибка, обработка исключения Конец транзакции 3 Конец транзакции 1 Транзакция 1 завершится возвратом в любом случае. А если транзакция 3 еще и не отправит ошибку вверх по стеку, (пример кода ниже), то при разборе полетов будет ничего не понятно.

Здесь помогает метод ТранзакцияАктивна, помогающий текущему коду понять, находится ли он внутри транзакции или его транзакция наверху.

- при возникновении исключения в общем случае транзакция не может быть зафиксирована – при этом не важно, было ли это исключение обработано или нет
- транзакция может быть инициирована явно в прикладном коде при использовании метода НачатьТранзакцию. Так же платформа неявным образом начинает транзакцию при любой записи в базу данных //Добавлено после экспериментов//

Некоторые события (например ПриЗаписиНаСервере) входят в состав транзакции, автоматически созданной платформой. Поэтому в некоторых источниках вся процедура обработки данного события входит в транзакцию (причем код, начинающий и заканчивающий транзакцию где-то посередине), что приводит к неправильным выводам (в моем случае - "то есть если в коде присутствует начало и конец транзакции, то вся процедура попадает в транзакцию", "а зачем тогда посередине начинать транзакцию?").

Из особенностей следуют правила:

- Поскольку исключение не отменяет транзакцию сразу, но запрещает успешное завершение транзакции, то все вызовы НачатьТранзакцию с одной стороны и ЗафиксироватьТранзакцию или ОтменитьТранзакцию с другой стороны должны быть парными.
- Начало транзакции и ее фиксация (отмена) должны происходить в контексте одного метода
- Обработка исключений должна придерживаться следующих правил:
 - метод НачатьТранзакцию должен быть за пределами блока Попытка-Исключение непосредственно перед оператором Попытка
 - все действия, выполняемые после вызова метода НачатьТранзакцию, должны находиться в одном блоке Попытка, в том числе чтение, блокировка и обработка данных
 - метод ЗафиксироватьТранзакцию должен идти последним в блоке Попытка перед оператором Исключение, чтобы гарантировать, что после ЗафиксироватьТранзакцию не возникнет исключение
 - необходимо предусмотреть обработку исключений – в блоке Исключение нужно сначала вызвать метод ОтменитьТранзакцию, а затем выполнять другие действия, если они требуются
 - рекомендуется в блоке Исключение делать запись в журнал регистрации
 - при использовании вложенных транзакций в конце блока Исключение рекомендуется добавить оператор ВызватьИсключение. В противном случае исключение не будет передано выше по стеку вызовов, там не сработает обработка исключения, внешняя транзакция не будет явным образом отменена и платформа вызовет исключение «В данной транзакции происходила ошибка»

```
НачатьТранзакцию();
Попытка
    БлокировкаДанных = Новый БлокировкаДанных;
    ЭлементБлокировкиДанных =
        БлокировкаДанных.Добавить("Документ.ПриходнаяНакладная");
    ЭлементБлокировкиДанных.УстановитьЗначение("Ссылка", СсылкаДляОбработки);
    ЭлементБлокировкиДанных.Режим = РежимБлокировкиДанных.Исключительный;
    БлокировкаДанных.Заблокировать();
```

```

... // чтение или запись данных
ДокументОбъект.Записать();
ЗафиксироватьТранзакцию();
Исключение
ОтменитьТранзакцию();
ЗаписьЖурналаРегистрации(НСтр("ru = 'Выполнение операции'",
    УровеньЖурналаРегистрации.Ошибка,,,
    ОбработкаОшибок.ПодробноеПредставлениеОшибки(ИнформацияОбОшибке()));
ВызватьИсключение; // есть внешняя транзакция
КонецПопытки;

```

- Использование вложенных транзакций приводит к усложнению кода. Принимая решение об использовании этой возможности, нужно очень взвешенно оценить решаемую задачу: возможно, это усложнение просто не оправдано.
 - Не стоит усложнять код, явно используя метод НачатьТранзакцию, когда кроме записи объекта другие действия с базой данных не делаются – платформа при записи сама откроет транзакцию.
 - Не нужно явно открывать транзакцию тогда, когда не требуется выполнять ответственное чтение данных. Например, обычно ответственное чтение не требуется при записи нового объекта (нового набора записей регистра).
 - При использовании методов ПолучитьОбъект (или Прочитать для наборов записей) необходимо анализировать должно ли чтение быть ответственным и в зависимости от этого принимать решение о явном использовании метода НачатьТранзакцию.
 - Если метод рассчитан на вызов только в рамках уже открытой транзакции (например, метод предназначен для вызова только из событий ПередЗаписью, ОбработкаПроведения и т.п.) в общем случае явным образом открывать в нем транзакцию не имеет никакого практического смысла.
 - При необходимости повысить качество сообщений об ошибках – на каждом уровне разработчик может предусмотреть свою обработку исключений, для чего, возможно, потребуется открыть вложенную транзакцию.
- Пример. Вызывается метод ДобавитьЭлектроннуюПодпись. Внутри, если что-то пошло не так, нужно обработать исключение и добавить текст вида: «Не удалось добавить электронную подпись к объекту %ПредставлениеОбъекта% по причине:%ОписаниеОшибки%». В противном случае исключение будет обработано выше по стеку вызовов, например, при записи файла и будет выдано сообщение вида: «Не удалось записать файл %ИмяФайла% по причине: %ОписаниеОшибки%», где в «%ОписаниеОшибки%», будет просто указание на строчку кода и пользователю будет непонятно, зачем вообще программа записывала файл, если он просто его подписывал.
- При обработке исключения, если транзакция все еще активна, например, исключение возникло во вложенной транзакции, нельзя обращаться к базе данных, так как это приведет к исключению «В этой транзакции уже происходили ошибки». При этом нужно учитывать, что обращение к базе

данных может быть неявным, например, для получения представления ссылки.

- В общем случае в рамках одной транзакции нужно выполнять только те действия, которые неделимы, исходя из бизнес-логики.

Пример. При проведении документа записывается документ и его движения в регистрах. Если не прошла запись хотя бы в один регистр вся операция проведения должна быть отменена.

- Следует избегать транзакций, которые выполняются длительное время.

Например, неправильно: для загрузки адресного классификатора записывать все данные, относящиеся к одной версии классификатора в одной транзакции, для того, чтобы в случае ошибки откатить целиком загружаемую версию классификатора. Т.к. данных по одной версии классификатора много (объем около 1 Гб), то для выполнения такой транзакции, во-первых, может не хватить оперативной памяти (особенно при использовании файловой информационной базы на 32-разрядной ОС), а, во-вторых, такая операция будет выполняться достаточно долго и ее нельзя будет оптимизировать за счет выполнения в несколько потоков. Правильно: разбить загрузку новой версии классификатора на небольшие порции так, чтобы запись порции в одной транзакции не превышала 20 секунд в условиях высоконагруженной информационной системы и реализовать функциональность по откату к предыдущей версии в случае ошибки. Максимальная продолжительность указана исходя из того, что время ожидания установки транзакционной блокировки данных в информационной базе по умолчанию равно 20 сек.

- Чем дольше выполняется транзакция, тем большее время будут заняты ресурсы сервера 1С:Предприятия и СУБД. Как правило длинные транзакции занимают следующие ресурсы:
 - в ходе выполнения транзакции все изменения в базе данных записываются в журнал транзакций, что необходимо для возможности откатить транзакцию;
 - блокировки, установленные в транзакции, остаются до конца транзакции
 - на сервере 1С:Предприятия блокировки занимают оперативную память
 - другие ресурсы, необходимые самой бизнес-логике, которая выполняется в транзакции.
- Если две транзакции пересекаются по блокируемым ресурсам, то транзакция, которая начала выполняться позже, будет ожидать возможность установления блокировки ограниченное время (по умолчанию – 20 секунд), после чего будет завершена с исключением «Превышено время ожидания установки блокировки». Поэтому длинные транзакции могут сильно снижать удобство параллельной работы пользователей.
Возникновение таких исключений – это повод провести анализ действий, которые выполняются в конфликтующих транзакциях
- В рамках транзакции нужно стремиться выполнять минимум действий – только те, которые нельзя в соответствии с бизнес-логикой выполнять вне транзакции.
- Обязательное использование транзакции в случае, если для ускорения операции записи в регистр используется отключение итогов, такую операцию вместе с отключением и включением итогов необходимо выполнять в транзакции, иначе в других сеансах может возникнуть ошибка при получении среза последних.

Проверка понимания.

Определение и граница использования транзакции. Транзакции - нечто неделимое, возвращающее все в предыдущее состояние. Это осталось в голове после заумных текстов без тестов лапками. Первым, что пришло в голову - восстановить значение реквизита :) Обработка с полем ввода ТестовоеЧисло, 0 по умолчанию.

```
&НаСервере
Процедура БезОбработкиОшибокНаСервере()
□НачатьТранзакцию();
□ТестовоеЧисло = 5;
□ВызватьИсключение "Число не то!";
□ЗафиксироватьТранзакцию();
КонецПроцедуры
```

```
&НаКлиенте
Процедура БезОбработкиОшибок(Команда)
□БезОбработкиОшибокНаСервере();
КонецПроцедуры
```

Не удалось! Ошибка сгенерировалась, но ТестовоеЧисло стало 5. Ведь транзакции актуальны при изменении данных в базе, т е SQL запрос обрамляется чем-то типа "BEGIN ... COMMIT". Мой косяк, поехали дальше.

Ошибка при создании элемента справочника. Справочник Города.

```
&НаСервереБезКонтекста
Процедура НовыйЭлементСправочникаНаСервере()
□НачатьТранзакцию();
□НовыйГород = Справочники.Города.СоздатьЭлемент();
□НовыйГород.Наименование = "Тестовый город";
□НовыйГород.Записать();
    НовыйГород2 = Справочники.Города.СоздатьЭлемент();
□НовыйГород2.Наименование = "Тестовый город 2";
□НовыйГород2.Записать();
□ВызватьИсключение "Город не правильный!";
□ЗафиксироватьТранзакцию();
КонецПроцедуры
```

Да, тестовые города не создались. Получилось!

Вложенные транзакции (или то чего нет).

```
&НаСервереБезКонтекста
Процедура ВложеннаяТранзакция()
□НачатьТранзакцию();
□Попытка
□□НовыйГород = Справочники.Города.СоздатьЭлемент();
□□НовыйГород.Наименование = "Тестовый город вложенный";
□□НовыйГород.Записать();
□□ВызватьИсключение "Город не правильный!";
□□ЗафиксироватьТранзакцию();
□Исключение
□□ОтменитьТранзакцию();
□КонецПопытки;
КонецПроцедуры
```

```
&НаСервереБезКонтекста
Процедура ВложенныеТарнзакцииНаСервере()
□НачатьТранзакцию();
□НовыйГород = Справочники.Города.СоздатьЭлемент();
□НовыйГород.Наименование = "Тестовый город внешний";
□НовыйГород.Записать();
□ВложеннаяТранзакция();
□ЗафиксироватьТранзакцию();
КонецПроцедуры
```

Во вложенной транзакции мы подавляем исключение - и вся транзакция тихо не выполняется. Ни один город не создан. Я не получил вообще никакого отклика от платформы, что и логично. Теперь попробуем добавить во внешний код обработку ошибок.

```
&НаСервереБезКонтекста
Процедура ВложеннаяТранзакция()
□НачатьТранзакцию();
□Попытка
□□НовыйГород = Справочники.Города.СоздатьЭлемент();
□□НовыйГород.Наименование = "Тестовый город вложенный";
□□НовыйГород.Записать();
□□ВызватьИсключение "Город не правильный!";
□□ЗафиксироватьТранзакцию();
□Исключение
□□ОтменитьТранзакцию();
□КонецПопытки;
```

КонецПроцедуры

&НаСервереБезКонтекста

Процедура ВложенныеТарнзакцииНаСервере()

□НачатьТранзакцию();

□Попытка

□□НовыйГород = Справочники.Города.СоздатьЭлемент();

□□НовыйГород.Наименование = "Тестовый город внешний";

□□НовыйГород.Записать();

□□ВложеннаяТранзакция();

□□ЗафиксироватьТранзакцию();

□Исключение

□□ОтменитьТранзакцию();

□□Сообщить("Исключение было поймано");

□КонецПопытки;

КонецПроцедуры

Вполне логично, что сообщение не было сформировано.

Проверка функции ТранзакцияАктивна.

&НаСервереБезКонтекста

Процедура НовыйЭлементСправочникаНаСервере()

□Сообщить(ТранзакцияАктивна());

□НачатьТранзакцию();

□НовыйГород = Справочники.Города.СоздатьЭлемент();

□НовыйГород.Наименование = "Тестовый город 3";

□НовыйГород.Записать();

□Сообщить(ТранзакцияАктивна());

□ЗафиксироватьТранзакцию();

□Сообщить(ТранзакцияАктивна());

КонецПроцедуры

Ожидаемый вывод: Нет, Да, Нет. Теперь проверим факт из статьи о блокировках о возникновении блокировки и как следствие транзакции при чтении (это так я понял текст)

&НаСервереБезКонтекста

Процедура ТранзакцияПриЧтенииНаСервере()

□МойГород = Справочники.Города.НайтиПоНаименованию("Иркутск");

□Сообщить(ТранзакцияАктивна());

□НовыйГород = Справочники.Города.СоздатьЭлемент();

```
□НовыйГород.Наименование = "Тестовый город 3";
□НовыйГород.Записать();
□Сообщить(ТранзакцияАктивна());
КонецПроцедуры
```

```
&НаКлиенте
Процедура ТранзакцияПриЧтении(Команда)
□ТранзакцияПриЧтенииНаСервере();
КонецПроцедуры
```

И получил Нет, Нет. Видимо что-то не так понял. Аналогичный результат при поэлементном чтении всего справочника. После внимательного рассмотрения, о транзакции в пределах процедуры было сказано для обработки события ПриЗаписиНаСервере (в случае сохранения данных через форму). Чтож, проверим это. Если транзакция есть, то данный код не запишет данные.

```
&НаСервере
Процедура ПриЗаписиНаСервере(Отказ, ТекущийОбъект, ПараметрыЗаписи)
□Отказ = ТранзакцияАктивна();
КонецПроцедуры
```

И да, создать объект не удалось, значит транзакция была активна!

Количество выполненных действий при исключении во вложенной транзакции.

Смоделируем ситуацию: при входе во вложенную транзакцию происходит подавленная ошибка. Причем только одна ошибка в середине процесса. Сколько раз выполнится внешняя транзакция? Ведь в одном из предыдущих примеров исключение из вложенной транзакции поймано не было.

```
&НаСервереБезКонтекста
Процедура ВложеннаяТранзакция()
□НачатьТранзакцию();
□Попытка
□□НовыйЭлемент = Справочники.ДляТранзакций.СоздатьЭлемент();
□□НовыйЭлемент.Наименование = "Тестовый элемент внутренний";
□□НовыйЭлемент.Записать();
□□ВызватьИсключение "Ошибка!";
□□ЗафиксироватьТранзакцию();
□Исключение
□□ОтменитьТранзакцию();
□□Сообщить("Ошибка во вложенной транзакции");
□КонецПопытки;
```

КонецПроцедуры

&НаСервереБезКонтекста

Процедура ТестТранзакцииНаСервере()

□НачатьТранзакцию();

□Попытка

□□Для сч = 1 По 5 Цикл

□□□НовыйЭлемент = Справочники.ДляТранзакций.СоздатьЭлемент();

□□□НовыйЭлемент.Наименование = "Тестовый элемент внешний " + Строка(сч);

□□□НовыйЭлемент.Записать();

□□□Сообщить("Попытка записи " + Строка(сч) + " внешнего элемента.");

□□□//Если сч = 3 Тогда

□□□□ВложеннаяТранзакция(); //ошибка в середине

□□□□КонецЕсли;

□□КонецЦикла;

□□ВложеннаяТранзакция();//ошибка в конце

□□ЗафиксироватьТранзакцию();

□Исключение

□□ОтменитьТранзакцию();

□□Сообщить("Исключение было поймано");

□□КонецПопытки;

КонецПроцедуры

Сначала все логично. В указанном варианте (он повторяет предыдущий эксперимент) выведено 5 сообщений и "Ошибка во вложенной транзакции". Сообщение "Исключение было поймано" не отображается.

А дальше ждал сюрприз. Если раскомментировать условие (ошибка на третьем шаге) то получается какая-то бабуйня: выведено 3 сообщения, сообщение "Ошибка во вложенной транзакции" и сообщение "Исключение было поймано"! То есть исключение транслируется вверх, если происходит между выполняемыми действиями. Однако если после выполняемых действий, но до фиксации транзакции - ошибка не транслируется вверх, а транзакция тихо откатывается.

