Программировать в 1С за 11 шагов

Низамутдинов Ильяс

1С:ТАКСИ
второе издание

Простой пошаговый курс обучения программированию в 1С для всех

НИЗАМУТДИНОВ ИЛЬЯС

ПРОГРАММИРОВАТЬ В 1С ЗА 11 ШАГОВ

ПРОСТОЙ ПОШАГОВЫЙ КУРС ОБУЧЕНИЯ ПРОГРАММИРОВАНИЮ В 1С ДЛЯ ВСЕХ

ИЖЕВСК

2018 ГОД

Оглавление

Предисловие

Приветствую, уважаемый читатель!

Книга, которую Вы сейчас читаете, предназначена для людей, которые находятся в самой начальной стадии изучения языка программирования 1С. Это может быть простой бухгалтер, который хочет делать небольшие правки в программу, чтобы не ждать специалиста. Или студент, который хочет понять, что такое 1С, как написать на языке программирования 1С и стоит ли связывать свою будущую жизнь с этой отраслью. А возможно, это системный администратор - фрилансер, который решил получить дополнительный заработок, обслуживая незначительные задачи своих клиентов в «1С:Предприятии».

Так или иначе, эта книга будет полезна всем, кто желает получить базовые знания в работе с языком программирования 1С.

Мой труд рассчитан на тех, кто в принципе никогда не изучал никакие языки программирования. В этом его уникальность. Все идет с самых основ: переменные, циклы, условия. И заканчивается более сложными вещами, такими как язык запросов, работа с регистрами сведений и накоплений, СКД. Поэтому после изучения этой книги Вы сможете писать элементарные программы на языке 1С, читать имеющийся несложный код и выполнять другие базовые и элементарные работы с программами, написанными в среде 1С. Не исключаю, что этой информации Вам хватит с избытком, а возможно, Вы захотите идти дальше и изучать программирование в 1С на более глубоком и профессиональном уровне. Так или иначе, с этой книгой у Вас есть шанс на великолепный старт в изучении языка программирования 1С.

Книга разделена на одиннадцать глав, каждая глава также разделена на несколько частей. Каждая часть это кусочек тех или иных знаний и умений по программированию (конфигурированию) в платформе 1С. В каждой части дается минимум теоретических знаний и огромное количество различных практических примеров. Рекомендую прорешать все примеры, которые приведены в этой книге, тогда Вы очень хорошо закрепите свой навык программирования. Запомните: научиться программировать можно только программируя.

Удачи в изучении языка программирования в 1С!

Введение

Дорогие друзья, я начал программировать в 1С в 2008 году, когда пришел в одну фирму франчайзи 1С. После моего принятия на работу никто не занимался как-то моим обучением, благо я уже перед этим умел программировать в Pascal и успел проработать 1,5 года в одном из предприятий моего города. Поэтому какой-то базис у меня был. В той фирме не было какого-то отдела по обучению учеников, или каких-то специально разработанных программ и методик. Шеф просто дал мне одну методичку от фирмы 1С и одну толстенную книгу, где объяснялось всё, всё, всё, вплоть до значения каждого мелкого свойства. И так я начал постигать программирование в 1С путем изучения этой литературы и методичек.

После я ушел из фирмы и начал работать на себя – фрилансить. Это был послекризисный 2009 год. Многое из программирования в 1С мне пришлось изучить самостоятельно по различным разрозненным статьям в интернете. И в один момент я осознал, что в том обучении, которое я прошел в 1С-франчайзи, не было никакой структуры, не было системы. Многих вещей я не знал в принципе, когда там работал, - а должен был! Все эти знания я получил уже тогда, когда работал на себя, методом проб и ошибок. Поэтому в один прекрасный момент у меня родилась идея создать методичку, учебник, курс, который поможет начинающим как можно быстрее въехать в профессию программиста 1С. И который собрал бы в себя все те знания, которые необходимы именно начинающим разработчикам, тем, кто только хочет прикоснуться к этой профессии.

И прежде чем начать изучать материал этой книги, рекомендую Вам ознакомиться с основными положениями, соблюдение которых поможет легче его усвоить.

Как читать эту книгу

«Программировать в 1С за 11 шагов» - это не просто книга, где дается сухая теория. В каждой главе приведены различные практические примеры. Вы должны прорешать их все. Причем не просто скопировать код из листинга и посмотреть, как он работает, а именно решить ту задачу, что была перед этим поставлена. Это не значит, что нужно тупо набирать код с книги. Нет! Вы должны осмыслить приведенный в книге код и сделать то же самое, на заглядывая в книгу. Да, поначалу это будет трудно, но в процессе таких тренировок Вы гораздо быстрее запомните весь материал, который изложен в моем труде.

Структура книги

Если Вы впервые столкнулись с программой 1С, то начать изучение нужно с самой первой главы, где даются самые базовые значения о платформе 1С. Изучив первую главу, Вы научитесь самостоятельно устанавливать платформу 1С, создавать пустую базу (для разработки и учебы), пользоваться справкой и работать в отладке. Эти значения самые базовые и без них Вы не сможете идти дальше.

Во второй и третьей главе даются основы программирования в 1С: мы познакомимся с примитивными типами, научимся создавать переменные и узнаем основные конструкции языка программирования в 1С (условия, циклы, процедуры и т.д.). Четвертая глава очень объемная, в ней мы изучим основные объекты конфигурации (метаданные), которые часто применимы при разработке. В пятой главе мы научимся работать с управляемыми формами. Сама по себе тема

управляемых форм очень объемна, мы же освоим тот минимум, который необходим для начальной работы. В шестой главе узнаем все об объектных типах. Узнаем, что такое *Объект*, *Ссылка* и многие другие интересные понятия. В седьмой и восьмой главе мы будем изучать универсальные коллекции значений: массивы, структуры, таблицы значений и многое другое. В девятой главе мы изучим различные способы получения данных. При помощи выборок и запросов. Язык запросов в этой книге дан в том минимуме, который позволит Вам двигаться успешно дальше в изучении этого интересного инструмента. В десятой главе научимся работать с регистрами сведений и накоплений. А в одиннадцатой, заключительной главе мы научимся работать с печатными формами, отчетами, динамическими списками, т.е. со всем тем, что выводит пользователю конечную информацию.

Информация во всех главах структурирована, и в дальнейшем Вы сможете эту книгу использовать в качестве справочника.

Успехов в изучении языка программирования в 1С!

Глава 1. Вводная информация

В первой главе Вы узнаете общую информацию о разработке в платформе «1С: Предприятия», которая Вам поможет более свободно ориентироваться в материале этой книги, а также подготовит Вас к дальнейшему изучению.

Часть 1. Установка платформы и создание пустой базы

Если Вы еще ни разу не имели дело с программой 1С, то Вам нужно скачать и установить платформу 1С (в противном случае сразу переходите ко второй части этой главы, изучив только создание пустой базы). Фирма 1С предоставляет возможность скачать бесплатную учебную версию платформы по этой ссылке:

http://online.1c.ru/catalog/free/18610119/

Вам необходимо будет ввести свое ФИО и e-mail. Через какое-то время Вам на почту, указанную при регистрации, придёт архив с дистрибутивом на установку платформы. Скачайте его на жесткий диск Вашего компьютера и распакуйте.

Зайдите в распакованном архиве в папку platform_8_3_8_1933 (номер может отличаться) и найдите среди множества файлов файл под названием Setup.

Вот он.

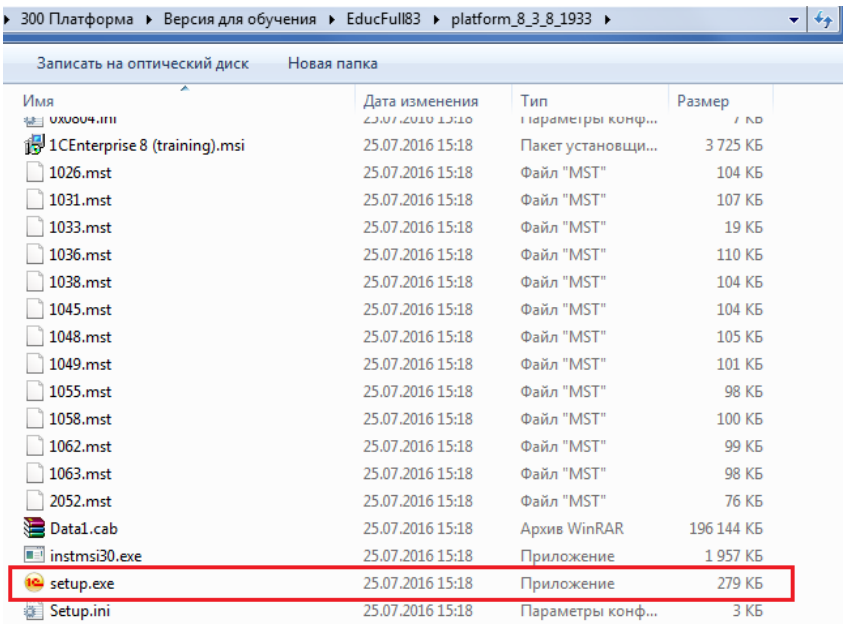


Рис. 1.1.1

Запускаем данный файл, дважды кликнув по нему левой клавишей мышки (у пользователя Windows должны быть административные права). После запуска должно открыться окно установки учебной версии платформы 1С.

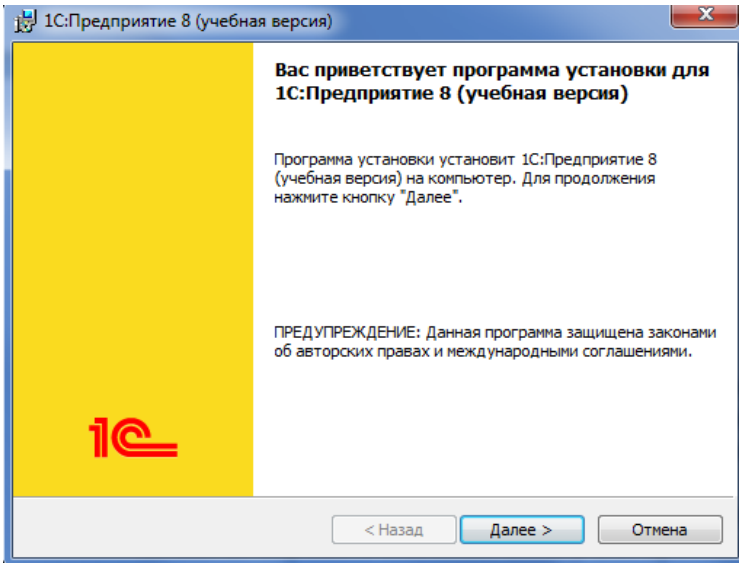


Рис. 1.1.2

Нажимаем кнопку «Далее».

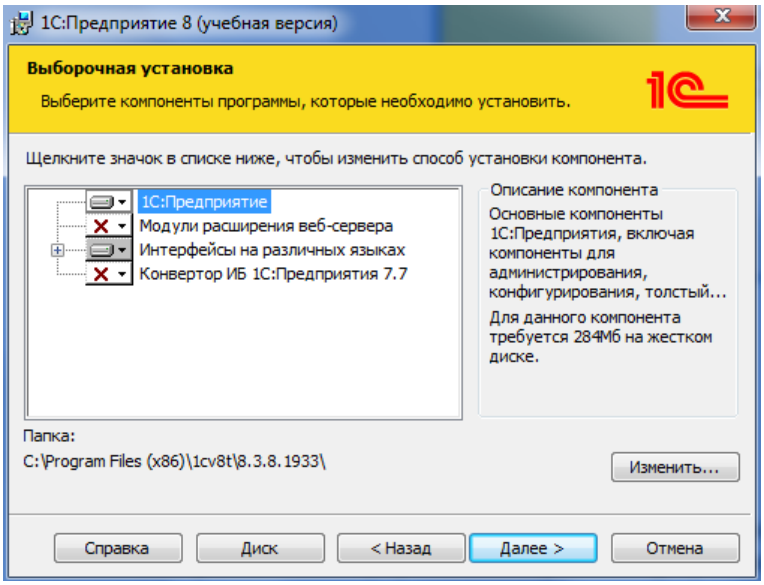


Рис. 1.1.3

В следующем окне обращаем внимание на два момента. Первый момент - это компоненты, которые должны будут быть инсталлированы при установке дистрибутива. Если Вы делаете обычную установку, то достаточно оставить компоненту **1С Предприятие** и **Интерфейсы на различных языках** (то, что предложено по умолчанию). Для начального изучения этих двух компонент Вам хватит.

Второй момент. Обратите внимание, в какой каталог на жестком диске будет установлена Ваша платформа. Вы можете поменять путь на любой другой, какой хотите, нажав на кнопку «Изменить». Мы оставим тот каталог, который есть.

Нажимаем кнопку «Далее».

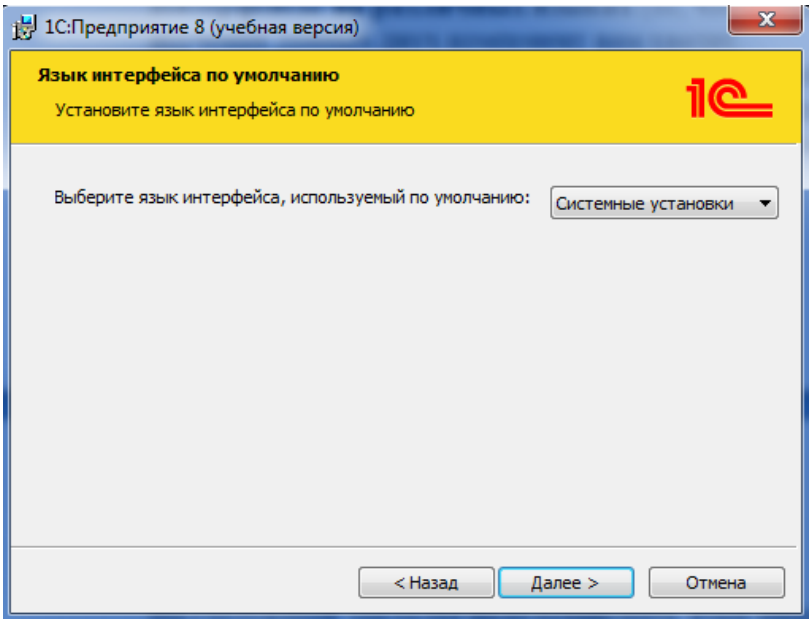


Рис. 1.1.4

На следующей закладке надо установить язык интерфейса. Я рекомендую его не менять и оставлять тот, который по умолчанию. Это либо *Русский*, либо *Системные установки*. Мы ничего менять не будем и нажмем кнопку «Далее».

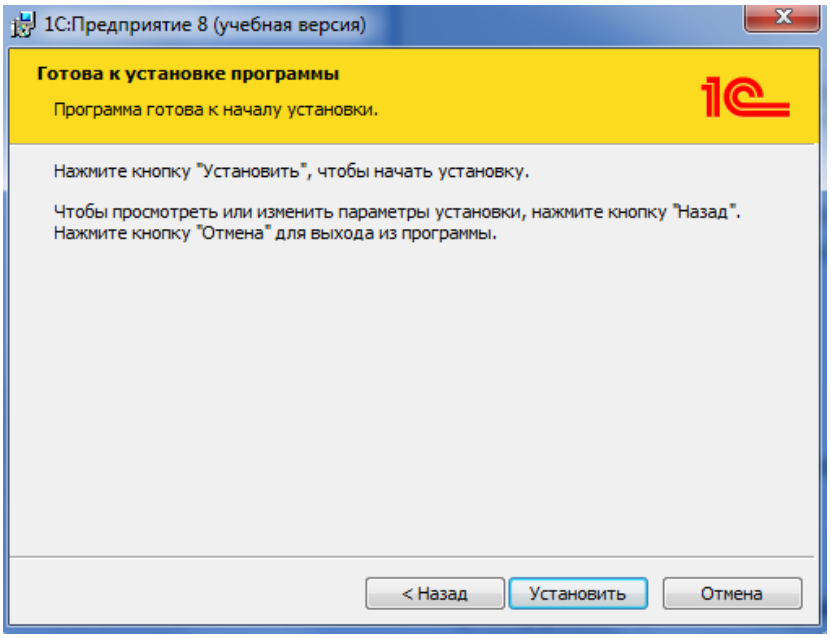


Рис. 1.1.5

В этот раз нажимаем кнопку «Установить». Платформа начнет устанавливаться. Вам необходимо дождаться окончания установки.

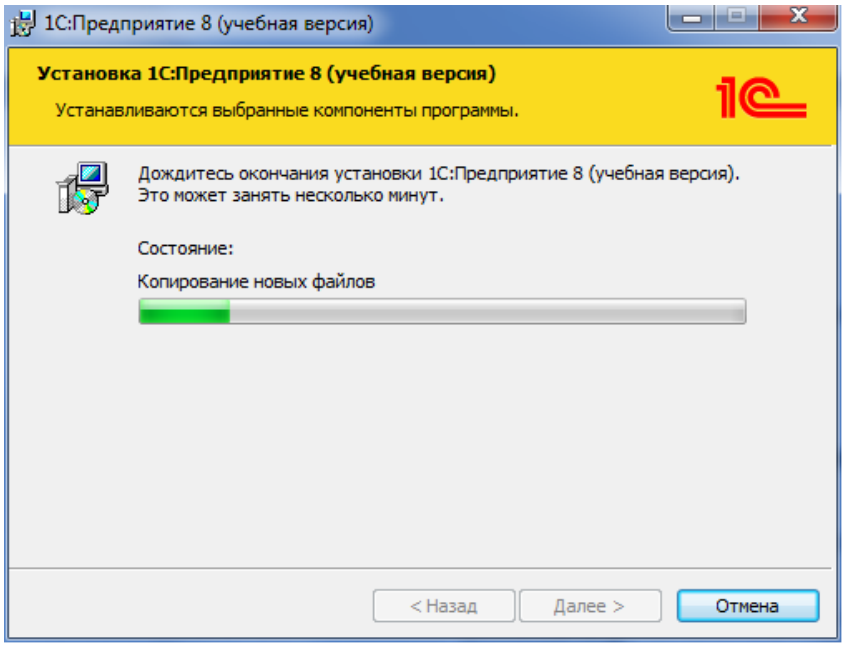


Рис. 1.1.6

Все, платформа установлена. Нажимаем кнопку «Готово».

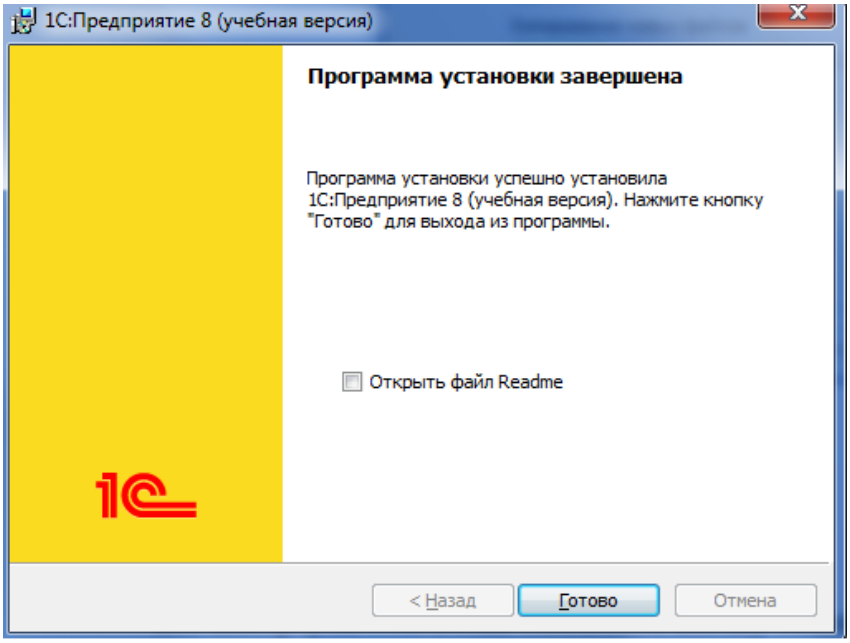


Рис. 1.1.7

Теперь создадим пустую базу, в которой будем изучать материал этой книги.

После установки платформы на Вашем рабочем столе должен появиться ярлык «1С: Предприятие». Запустите стартер «1С: Предприятия», дважды кликнув левой клавишей мышки по этому ярлыку. И сразу же Вам выйдет сообщение о том, что список баз пуст, и вопрос о добавлении базы в список. Отвечаем «Да»

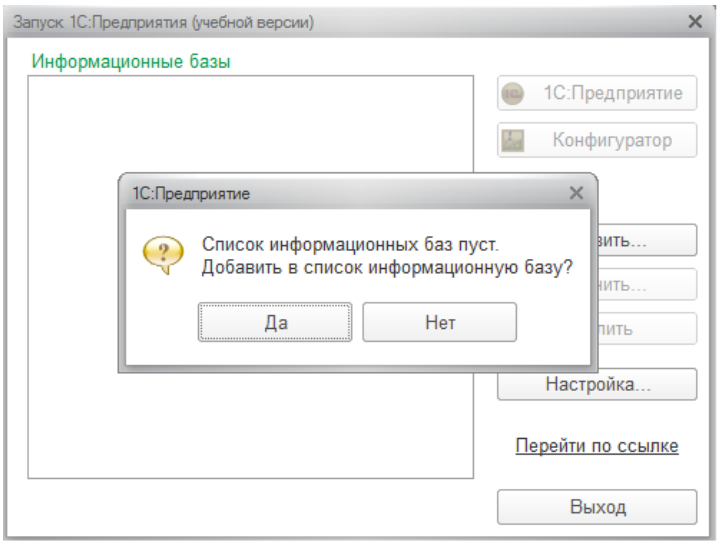


Рис. 1.1.8

В следующих двух окнах выбираем пункты «Создание новой информационной базы» и «Создание информационной базы без конфигурации…».

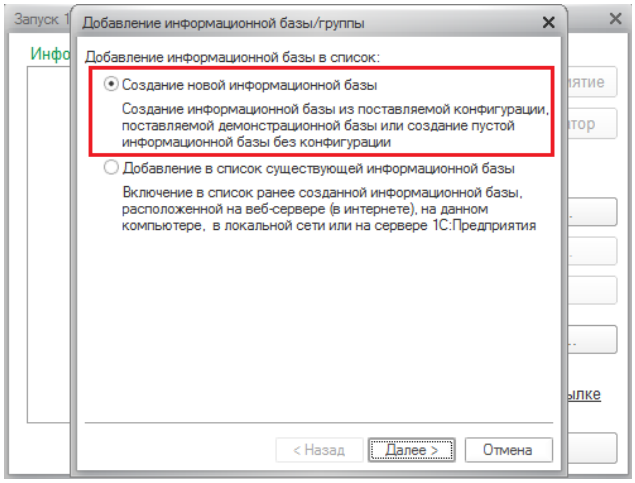


Рис. 1.1.9

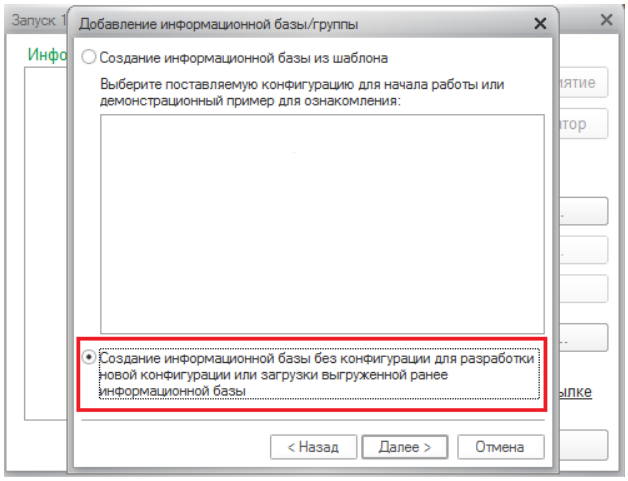


Рис. 1.1.10

В последующих двух окнах Вам нужно придумать название для базы и выбрать каталог, где база будет храниться.

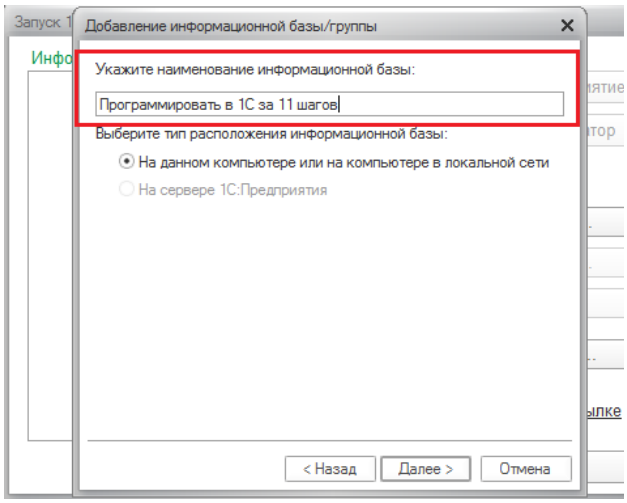


Рис. 1.1.11

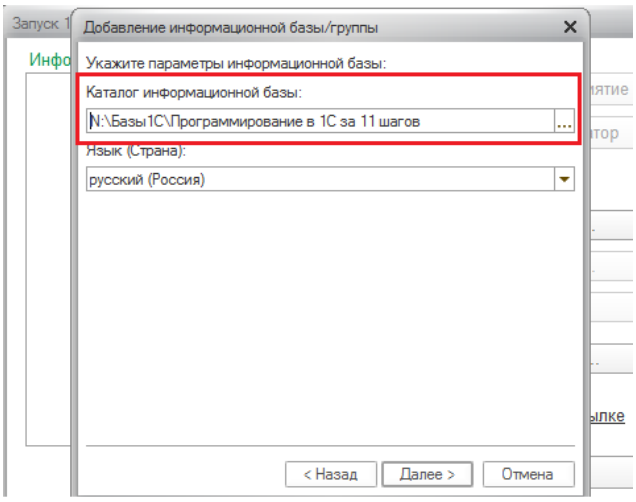


Рис. 1.1.12

В последнем окне ничего не меняем и нажимаем кнопку «Готово». База будет создана и появится в списке.

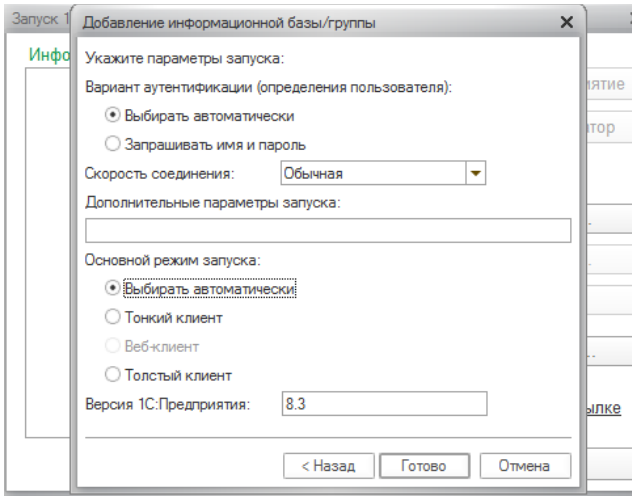


Рис. 1.1.13

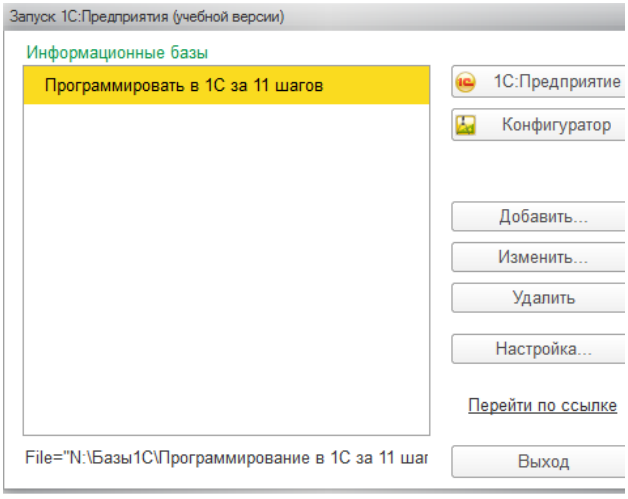


Рис. 1.1.14

Часть 2. Варианты хранения данных 1С: файловый вариант и клиент-сервер. Обычное и управляемое приложение

Варианты хранения данных

Платформа 1С поддерживает два варианта хранения данных – это ***Файловый вариант работы*** и ***Клиент-серверный вариант***.

Файловый вариант предназначен для работы одного или нескольких пользователей в рамках небольшой сети. В этом варианте работы все данные располагаются в одном файле, так называемой файловой СУБД.

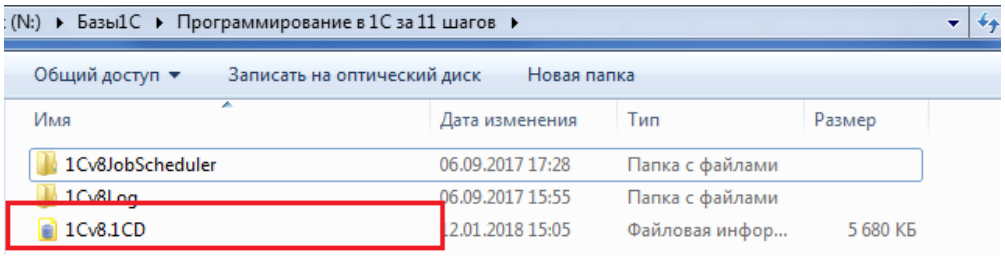


Рис. 1.2.1

Этот файл имеет расширение 1CD и название 1Cv8. Если название или расширение будет другое, то платформа не найдет эту базу!

При подключении базы в стартере 1С необходимо указывать путь расположения файла базы, как это было сделано выше (см. рис. 1.1.12). При выделении уже подключенной базы в окне стартера 1С внизу будет путь к папке, в которой находится файл базы.

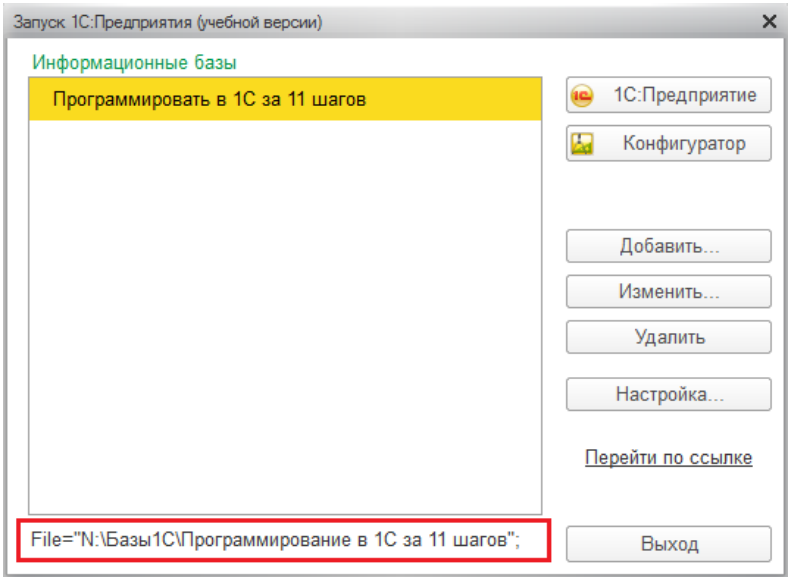


Рис. 1.2.2. Представление файлового варианта в окне выбора баз

Для работы с файловой базой достаточно установить платформу 1С. Схематически работу файловой базы можно представить следующим образом:

Рис. 1.2.3. Схема работы файлового варианта

Второй вариант работы - это ***Клиент-серверный***. Данный вариант предназначен для большого числа пользователей в масштабе предприятия. Он реализован на основе трехуровневой структуры «Клиент-сервер».

Эта структура состоит из трех звеньев. Первое звено - это SQL сервер (как правило, это не имеющее отношение к фирме 1С приложение, например Microsoft SQL Server или Postgre). SQL серверов, которые могут работать с 1С, существует ограниченное количество, но мы не будем их все перечислять. Если кто-то заинтересуется подобной информацией, без труда сможет ее найти. В этом приложении хранятся все данные.

Для того чтобы клиентское приложение работало с SQL сервером, необходим посредник. Эту роль выполняет кластер серверов «1С:Предприятия», который, по сути, связывает конечного пользователя с SQL сервером. Кластер серверов - это процесс (или процессы), который запущен на каком-либо компьютере. Компьютеры, где запущен сервер SQL и кластер серверов 1С, могут быть разными.

И третье звено - это непосредственно клиентское приложение, в котором работает конечный пользователь.

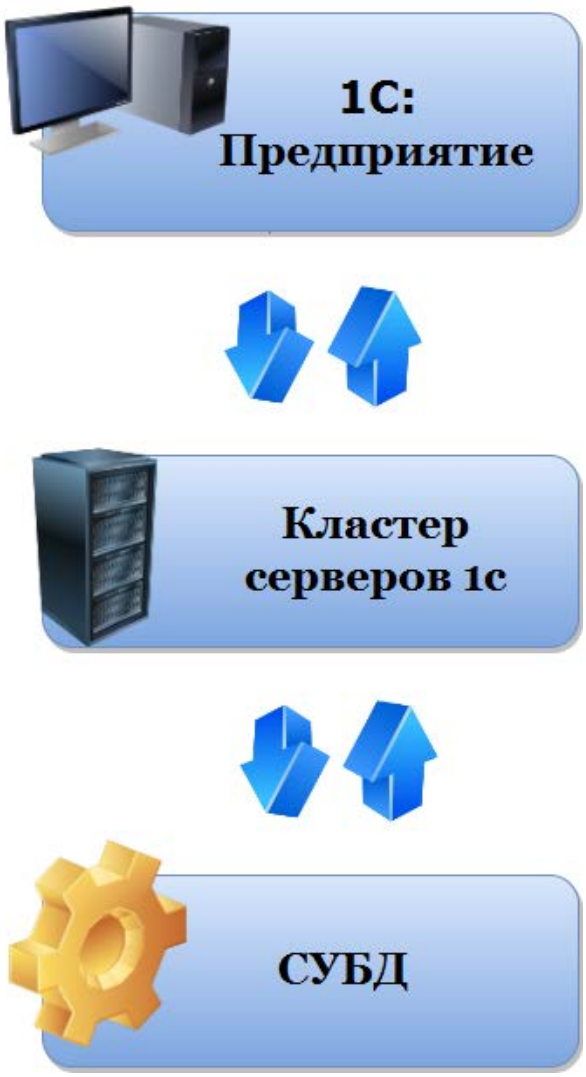


Рис. 1.2.4. Схема работы SQL варианта

В этой книге мы будем рассматривать работу только файлового варианта.

Обычное и управляемое приложение

После выхода платформы 8.2. программа 1С может работать в двух режимах: в режиме обычного приложения и в режиме управляемого приложения. Мы очень кратко разберем особенности работы в них. Более подробно отличия этих приложений даются в первой части книги *«Основы разработки в 1С: Такси. Разработка в управляемом приложении за 12 шагов»*.

В обычном приложении интерфейс пользовательского режима «1С:Предприятия» имеет следующий вид:

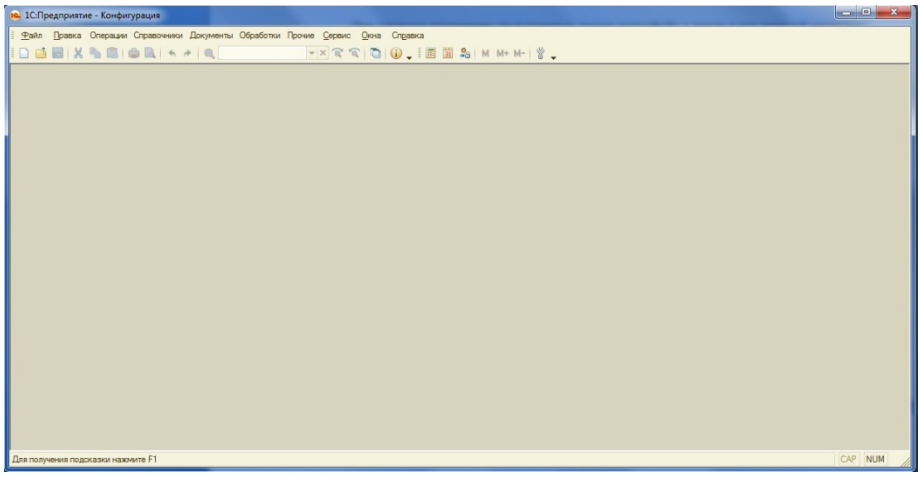
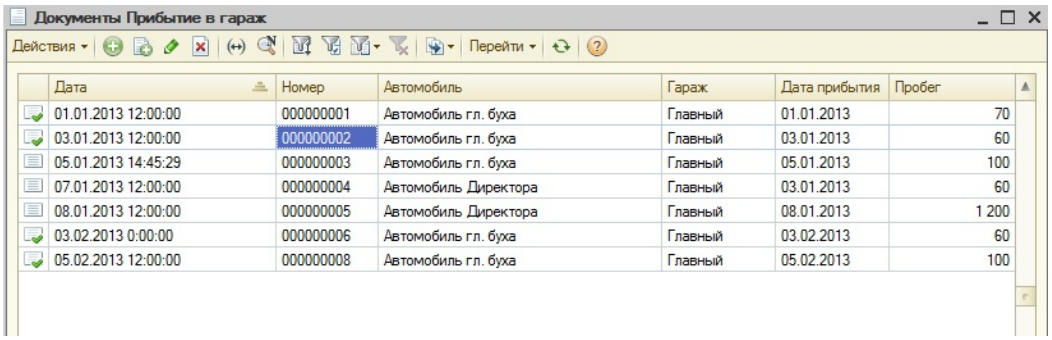


Рис. 1.2.5

Для того, чтобы выполнить какие-то действия (создать документ, распечатать отчет), пользователь должен работать с меню. Переходя по пунктам меню, пользователь может открывать различные формы. В основном это формы списков справочников и документов (см. рис. 1.2.6), но также могут быть отчеты, обработки, планы счетов и пр.



Дата	Номер	Автомобиль	Гараж	Дата прибытия	Пробег
01.01.2013 12:00:00	000000001	Автомобиль гл. буха	Главный	01.01.2013	70
03.01.2013 12:00:00	000000002	Автомобиль гл. буха	Главный	03.01.2013	60
05.01.2013 14:45:29	000000003	Автомобиль гл. буха	Главный	05.01.2013	100
07.01.2013 12:00:00	000000004	Автомобиль Директора	Главный	03.01.2013	60
08.01.2013 12:00:00	000000005	Автомобиль Директора	Главный	08.01.2013	1 200
03.02.2013 0:00:00	000000006	Автомобиль гл. буха	Главный	03.02.2013	60
05.02.2013 12:00:00	000000008	Автомобиль гл. буха	Главный	05.02.2013	100

Рис. 1.2.6

Из формы списка пользователь может открыть форму документа или справочника (см. рис. 1.2.7).

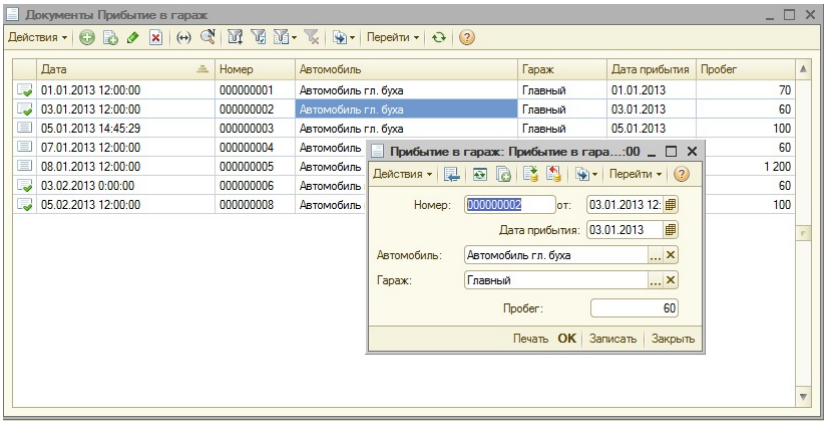


Рис. 1.2.7

В управляемом приложении все несколько по-другому. В случае с интерфейсом *Такси* командный интерфейс выглядит так:

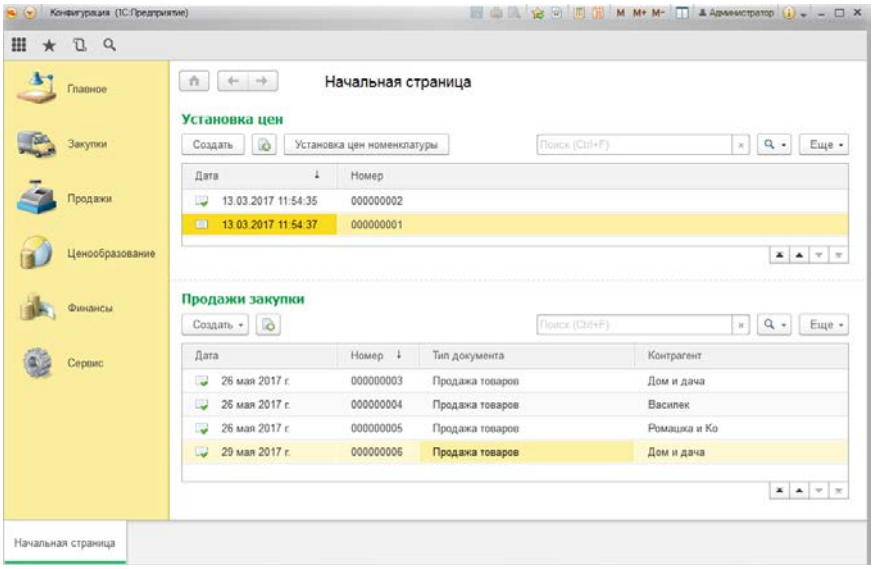


Рис. 1.2.8

А форма документа выглядит следующим образом:

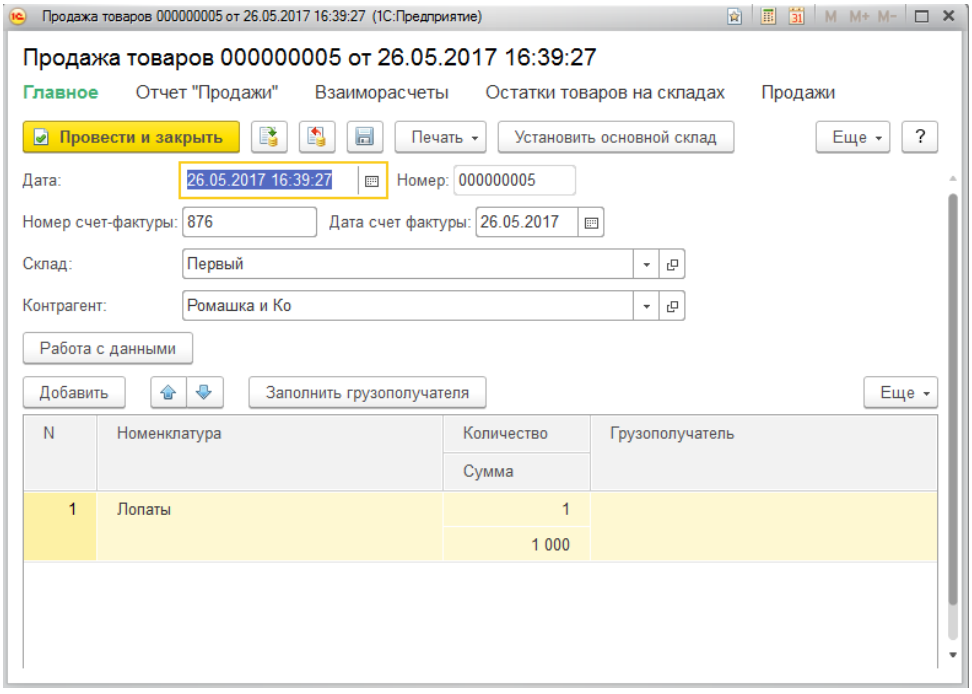


Рис. 1.2.9

В этом труде будет дан минимум по работе с управляемым приложением, кто же заинтересуется особенностями разработки управляемого приложения, может ознакомиться с моей книгой *«Основы разработки в 1С: Такси. Разработка в управляемом приложении за 12 шагов»*.

Все примеры в этой книге приведены под управляемым приложением и на управляемых формах.

Часть 3. 1С: Предприятие и 1С:Конфигуратор

Система программ «1С: Предприятие» стоит на двух столпах. Это технологическая платформа и прикладные решения.

Технологическая платформа - это, по сути, оболочка, в которой существуют прикладные решения. Сами по себе прикладные решения существовать не могут. Но и платформа без прикладного решения нужна только разработчикам. Конечным пользователям она ни к чему.

Прикладное решение - это автоматизация определенной области хозяйственного учета. Например, такие конфигурации как «Бухгалтерия предприятия», «Управление торговлей» и «Зарплата и управление персоналом» являются прикладными решениями, разработанными фирмой 1С. Прикладные решения разрабатываются в основном фирмой 1С, но их могут создавать и сторонние организации. Например, когда Вы научитесь программировать в 1С, то тоже сможете разработать собственное прикладное решение на базе платформы 1С и продавать его. Иногда прикладное решение еще называют *конфигурацией*. Конфигурация «1С: Бухгалтерия предприятия», конфигурация «1С: Управление торговлей» и т.д.

Платформа 1С может работать в двух режимах: в режиме «1С:Предприятие» и в режиме «1С:Конфигуратор». Режим «1С:Предприятие» предназначен непосредственно для работы конечных пользователей. В дальнейшем в книге режим работы конечных пользователей я буду называть или «1С:Предприятие», или *пользовательский режим*.

Начиная с платформы 8.2 режим «1С:Предприятие» может работать в трех видах клиентских приложений - это «Толстый клиент», «Тонкий клиент» и «Веб-сервис». Работа *тонкого клиента* и *веб-клиента* возможна только в *управляемом приложении.* В *обычном приложении* возможна работа только под *толстым клиентом*. Под *толстым клиентом* можно работать как в *обычном*, так и в *управляемом приложении*.

В платформах 8.0 и 8.1 была возможность работы только под *толстым клиентом*. Тогда это был обычный и единственный вид клиентского приложения и так не назывался. Почему появилась необходимость разделения обычного клиентского приложения на три вида – «тонкий», «толстый» и «веб-клиент»? Виной этому развитие технологий вообще и интернета в частности. Очень часто стала появляться необходимость в работе с «1С:Предприятием» через сеть Internet, и это стало накладывать определенные ограничения, поскольку пропускная способность сети Internet гораздо уже, чем обычной локальной сети. Поэтому те технологии платформы, которые существовали при редакциях 8.0 и 8.1, стали неприменимы в новых реалиях. Как следствие, была разработана платформа 8.2, в которой приложение разделилось на три вида: *толстый*, *тонкий* и *веб-клиент*.

Толстый клиент - это обычный клиент «1С:Предприятия». Под этим клиентом возможна работа только в локальной сети Ethernet. Это отдельное приложение, которое устанавливается на компьютер пользователя. Посредством этого приложения можно получить доступ к базам с прикладными решениями, как на компьютере пользователя, так и к базам в локальной сети Ethernet. Только под приложением для *толстого клиента* возможна разработка, т.е. есть доступ в *конфигуратор 1С*!

Под *тонким клиентом* возможна работа как по локальной сети Ethernet, так и по сети Internet. Это специальное отдельное приложение, которое тоже устанавливается на компьютер пользователя. Но посредством этого приложения можно получить доступ как к базам в локальной сети Ethernet, так и к базам через сеть Internet (для этого на компьютере, где находится база данных, должен быть развернут веб-сервер).

Веб-клиент работает только через веб-браузер. Через веб-браузер возможно зайти в любую базу прикладного решения (если она опубликована на веб-сервере). Для этого нет необходимости в установке какого-то приложения, достаточно на компьютере иметь какой-нибудь веб-браузер.

Все примеры в этой книге могут работать как в режиме *тонкого клиента*, так и в режиме *толстого клиента*. На будущее, когда Вы будете разрабатывать или дорабатывать прикладное решение, ориентируйтесь на то, что пользователи будут работать *под тонким клиентом*. Более подробно об этом в 5-й главе.

Режим «1С:Конфигуратор» предназначен для разработки и отладки прикладного решения. Т.е. если Вам надо написать какой-нибудь отчет, обработку, или внести изменение в имеющееся прикладное решение, то сделать это Вы сможете только в конфигураторе. Обычно режим «1С:Конфигуратор» все просто называют *конфигуратор.*

Для того чтобы запустить конфигуратор, необходимо нажать на кнопку «Конфигуратор» окна запуска программы 1С (см. рис. 1.3.1).

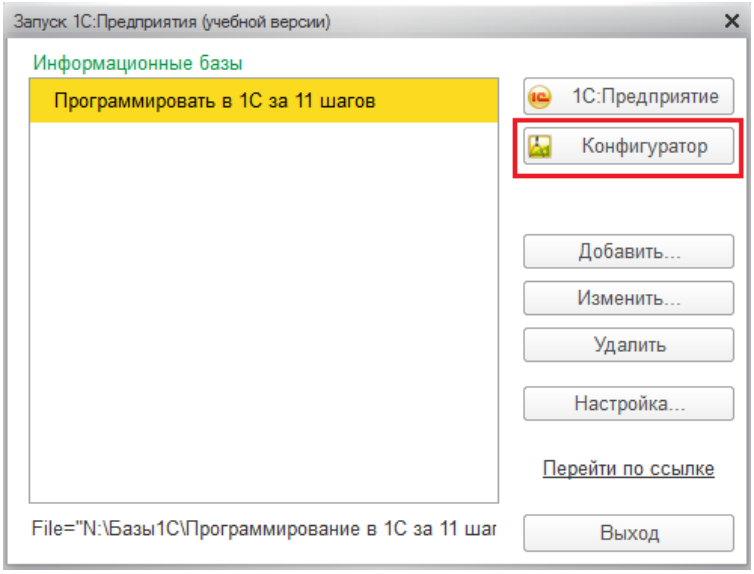


Рис. 1.3.1

Конфигуратор откроется, и Вы увидите большое окно с различными меню вверху. Вот это большое окно будем называть *Рабочий стол разработчика* (см. рис. 1.3.2):

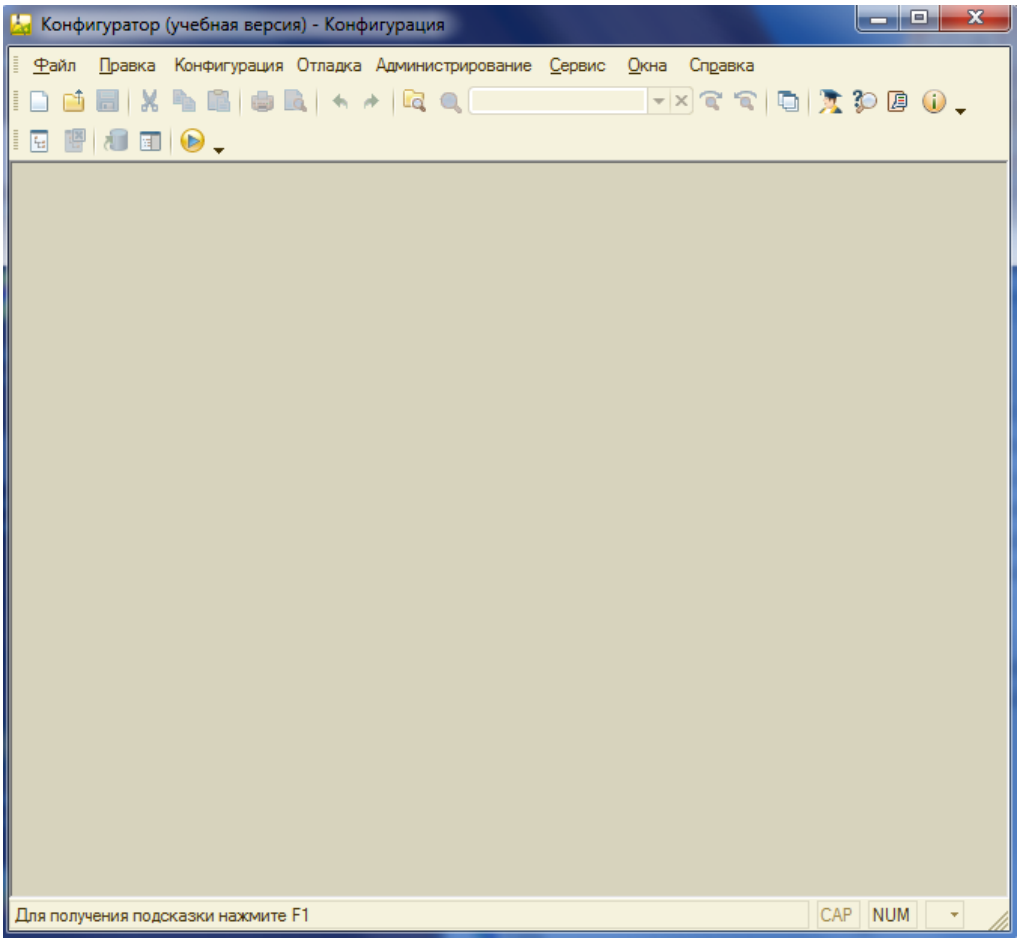


Рис. 1.3.2

Слева обычно расположено дерево конфигурации. В нашем случае оно сейчас закрыто. Для того чтобы открыть его, необходимо нажать на кнопку *«Открыть конфигурацию»* (см. рис. 1.3.3):

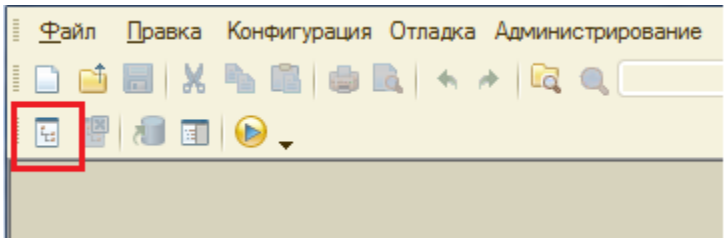


Рис. 1.3.3

Конфигурация - это некий набор метаданных, с помощью которого функционирует прикладное решение (справочники, документы, планы счетов и т.д.).

В нашем случае набор метаданных пустой, но в том случае, если Вы откроете конфигурацию какого-нибудь прикладного решения, он будет заполнен. Пока не будем обращать внимания на метаданные, потому что вопросы по работе с ними мы будем рассматривать в последующих главах этой книги.

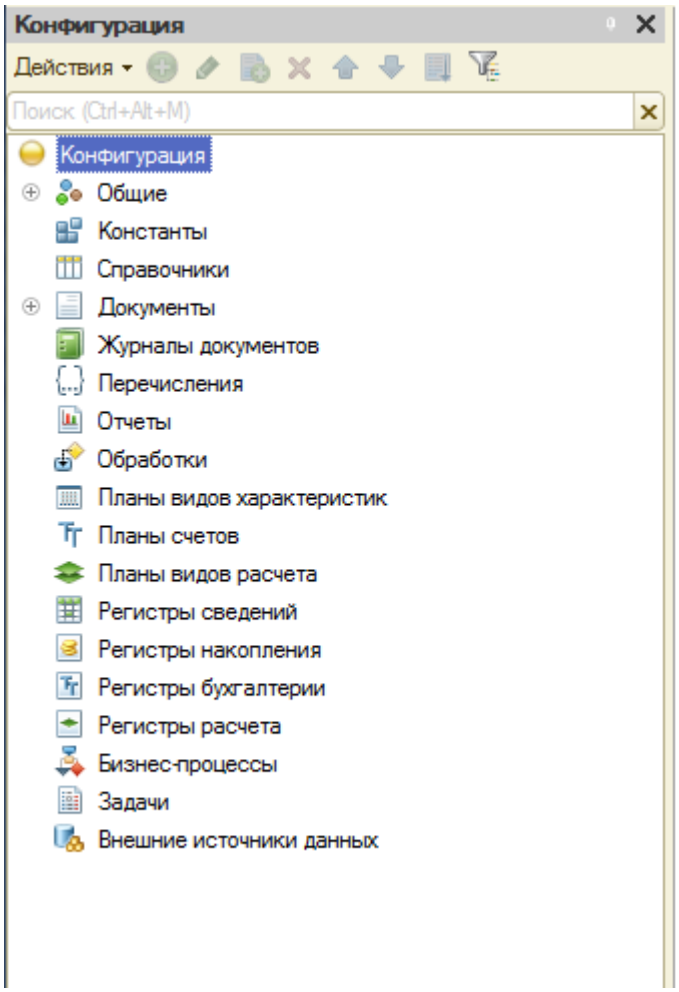


Рис. 1.3.4

Эта вся информация о работе с конфигуратором, которая Вам необходима на данном этапе изучения языка программирования 1С.

Для того чтобы запустить «1С:Предприятие», необходимо нажать в окне запуска кнопку *«1С:Предприятие»*. Запустить «1С:Предприятие» можно непосредственно из конфигуратора, нажав на кнопку «Начать отладку».

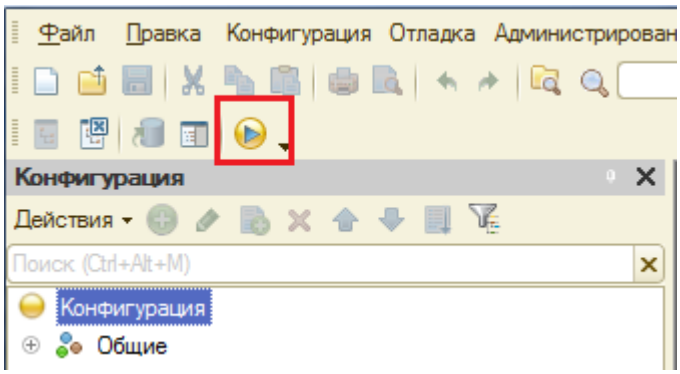


Рис. 1.3.5

Тогда у Вас откроется прикладное решение, в котором Вы сможете вносить и обрабатывать данные, согласно алгоритмам, которые созданы разработчиками. Если у Вас еще

нет никаких метаданных (как в нашем случае), то просто откроется окно *1С:Предприятия* (см. рис. 1.3.6):

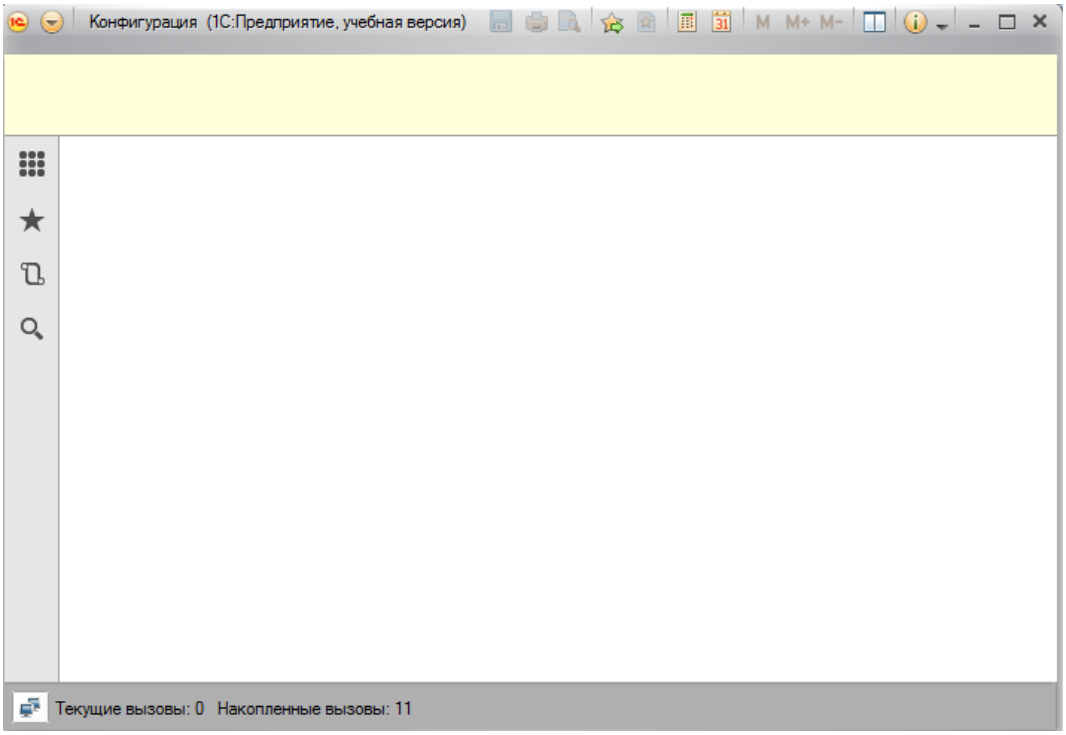


Рис. 1.3.6

Часть 4. Язык программирования в 1С

Вся разработка в 1С осуществляется при помощи встроенного языка программирования в 1С. Посредством этого языка разрабатываются различные алгоритмы поведения программы (расчеты, проведение документов и т.п.). В этой части мы рассмотрим базовые основы языка программирования 1С.

Программный модуль

И начнем мы с программного модуля. Что такое программный модуль? Программный модуль - это место, где непосредственно пишется исполняемый код.

В отличие от остальных языков, в 1С программный модуль не является самостоятельной программой и функционирует только в рамках конфигурации 1С, т.к. он является частью этой конфигурации, или это модуль внешней обработки или отчета (модуль формы внешней обработки(отчета)), которые тоже не могут самостоятельно существовать, а работают только в программе «1С:Предприятие».

Программный модуль – это среда, в которой пишется текст на встроенном языке программирования 1С. Этот текст содержит процедуры и функции (их еще называют одним словом *Методы*), описывающие те или иные алгоритмы работы программы, которые вызываются системой в определенные моменты (например, при нажатии на кнопку).

Всего существует 11 видов программных модулей, но мы в этой книге затронем только некоторые из них: модуль формы, модуль объекта и т.д. Этого будет вполне достаточно для изучения языка программирования 1С.

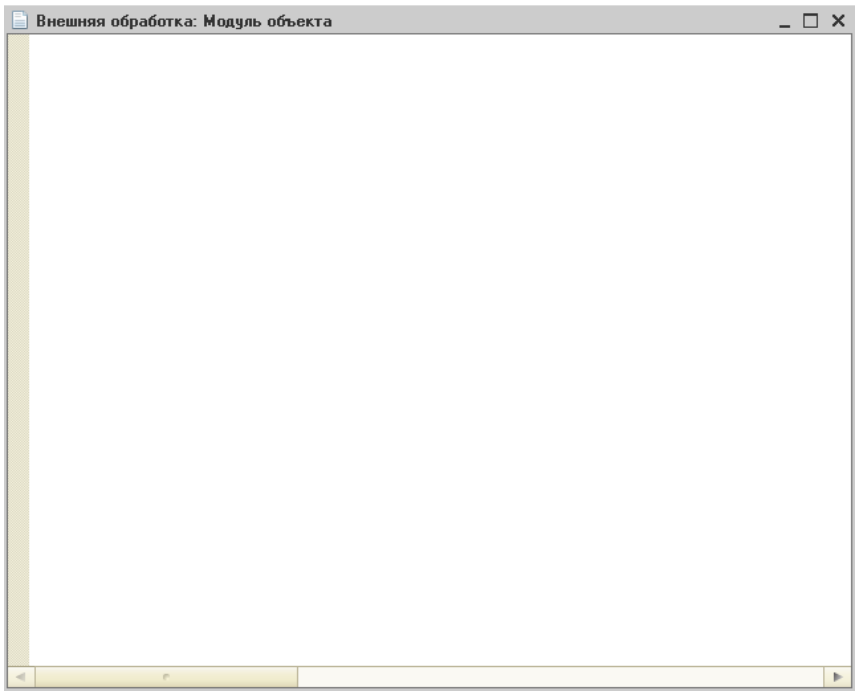


Рис. 1.4.1. Модуль объекта (внешней обработки)

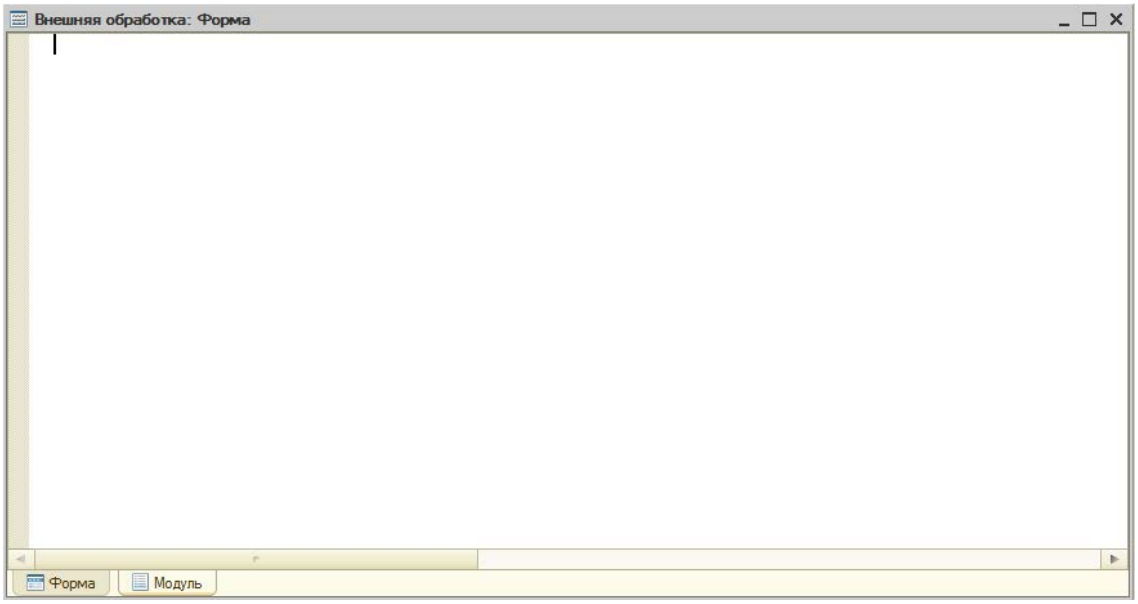


Рис. 1.4.2. Модуль формы

Более подробно программные модули будем разбирать в четвертой главе.

Исходный текст программного модуля состоит из операторов и комментариев.

Комментарии

Разберем первым делом комментарии.

Для того, чтобы в программном модуле внести пояснения к коду, программист может писать комментарии, которые не будут читаться компилятором и не будут участвовать в общей работе кода. Делается это достаточно просто.

Посмотрите, на примере, как это выглядит:

```
А = 1; //присвоили 1
//Написали комментарий
```

Рис. 1.4.3

Как Вы видите на рис. 1.4.3, пишутся две наклонные вправо черты, после них весь текст будет зеленого цвета. Хорошим тоном программирования считается написание исчерпывающих комментариев к каждому действию. Обращаю Ваше внимание, что после того как Вы стали писать комментарий, на этой строке нельзя поставить какой-либо оператор, только на следующей строке.

Многострочных комментариев в 1С нет. Но есть способ закомментировать сразу несколько строк. Для этого надо выделить текст и выбрать команду «*Текст*»-«*Блок*»-«*Добавить комментарий*» (см. рис. 1.4.4 и 1.4.5):

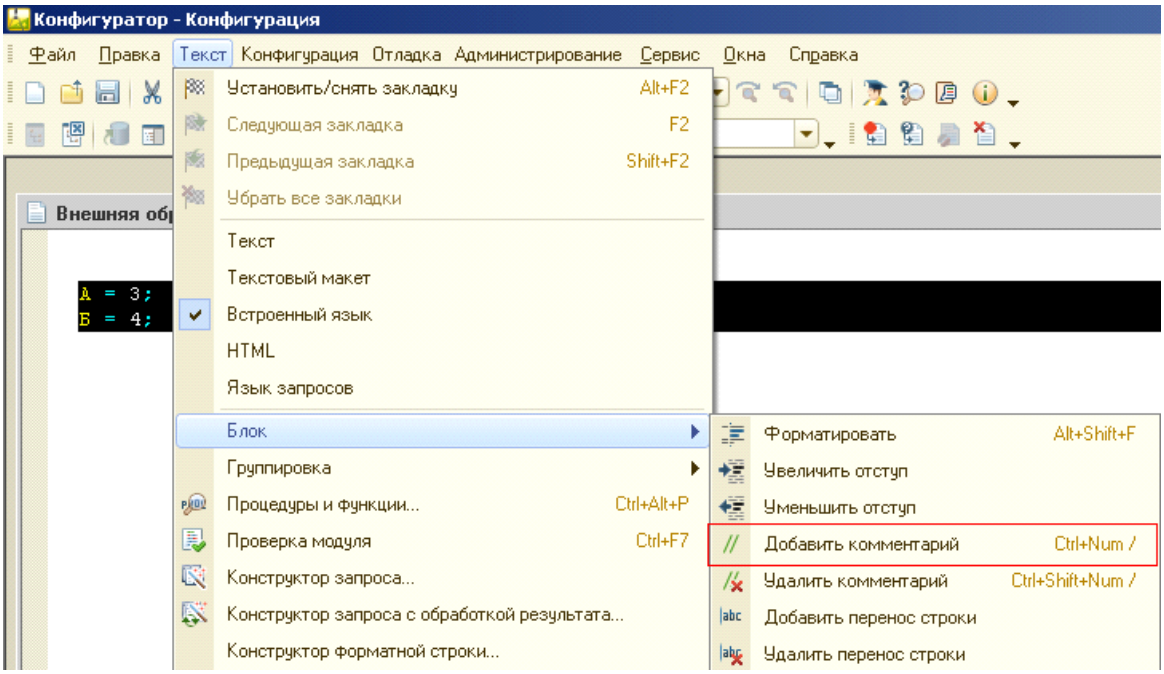


Рис. 1.4.4

```
//А = 3;
//Б = 4;
```

Рис. 1.4.5

Операторы

Разберем основной столп языка программирования 1С. Это операторы.

Что такое оператор? ***Оператор*** - это наименьшая автономная часть языка программирования. Любая программа представляет собой последовательность операторов. Операторы 1С можно поделить на две группы: это ***Операторы объявления переменных*** и ***Исполняемые операторы***.

Операторов объявления переменных два - это *Оператор явного объявления переменных*:

Перем А;

и *Оператор неявного объявления* - Оператор присваивания:

А = 1;

Оператор присваивания может не только объявлять переменную, но и присваивать имеющейся переменной какое-нибудь значение.

Перем А;

А = 1;

В данном примере переменная уже объявлена, и затем ей присваивается значение.

Исполняемые операторы совершают определенные действия, которые манипулируют переменными, они могут быть в виде процедур, либо в виде синтетических конструкций типа «*Если», «Цикл»* и т.п.

Все операторы должны быть разделены между собой точкой с запятой, перенос строки не является признаком разделения оператора.

Ключевые слова

Ключевое слово в языке программирования - это слово, которое имеет определенное значение для компилятора. Его нельзя использовать в названии переменных, процедур и функций. В языке программирования 1С ключевые слова также называют зарезервированными.

В программе 1С список ключевых слов фиксированный, и Вы сможете ознакомиться с ним в приложении к главе.

Язык написания, имена переменных, функций и процедур

Язык написания в программе 1С является двуязычным: кириллица и латиница. Т.е. в написании имен переменных, функций и процедур можно использовать английские и русские имена без каких-либо ограничений. Все ключевые слова могут быть написаны как в русском, так и в английском варианте. Причем регистр написания букв никакой роли не играет. Как Вы уже узнали, в написании имен переменных, функций и процедур можно использовать любые латинские и кириллические символы, а также цифры и знак подчеркивания «_»

Например, можно назвать переменные так:

Перем НашаПеремА,А1, п2, V3,wефв_3;

Но платформа выдаст ошибку, назови Вы переменные следующим образом:

Перем НашаПерем%,А#, п2?, V*3,wефв_3-4;

Часть 5. Работа со справкой

Поскольку язык программирования 1С очень обширен, то не все функции и процедуры можно обхватить текущей книгой. Кое-что будет упущено не специально, а кое-что – преднамеренно, для того чтобы Вы могли научиться работать со справкой.

Работать со справкой очень просто. Для начала необходимо ее открыть в конфигураторе.

Перейдите *«Главное меню»* - *«Справка»* – *«Синтакс-помощник»*. И откройте ее.

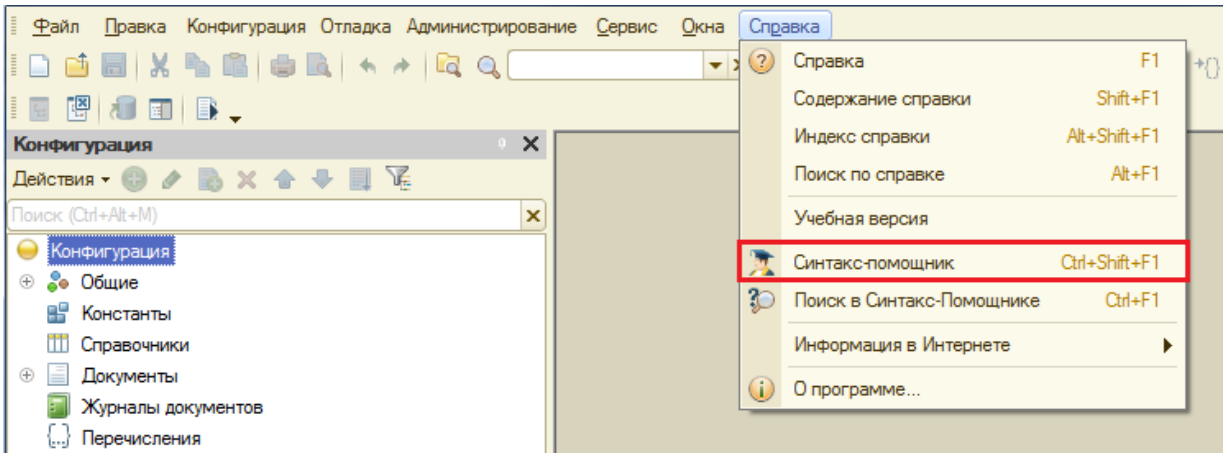


Рис. 1.5.1

Окно справки состоит из трех закладок: *«Содержание»*, *«Индекс»* и *«Поиск»*.

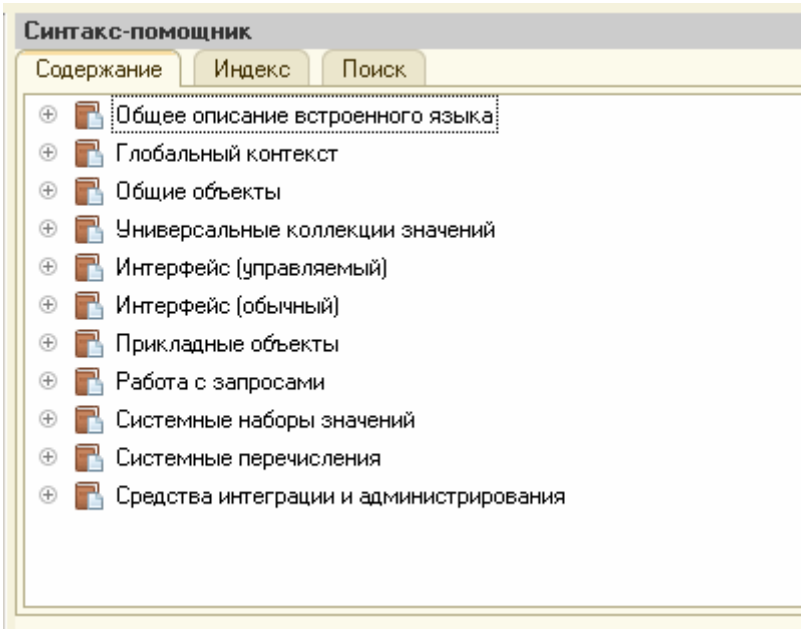


Рис. 1.5.2

Закладка *«Содержание»* представляет собой дерево справочной информации. Сейчас для Вас все это темный лес. Но когда Вы уже немного начнете разбираться в языке программирования 1С, то информация в данной закладке станет более осмысленной. Например, в следующем уроке мы будем проходить переменные, в том числе и строки. Если Вам интересно посмотреть, какие функции и процедуры по работе со строками есть в платформе 1С, Вы сможете без труда это сделать в справке.

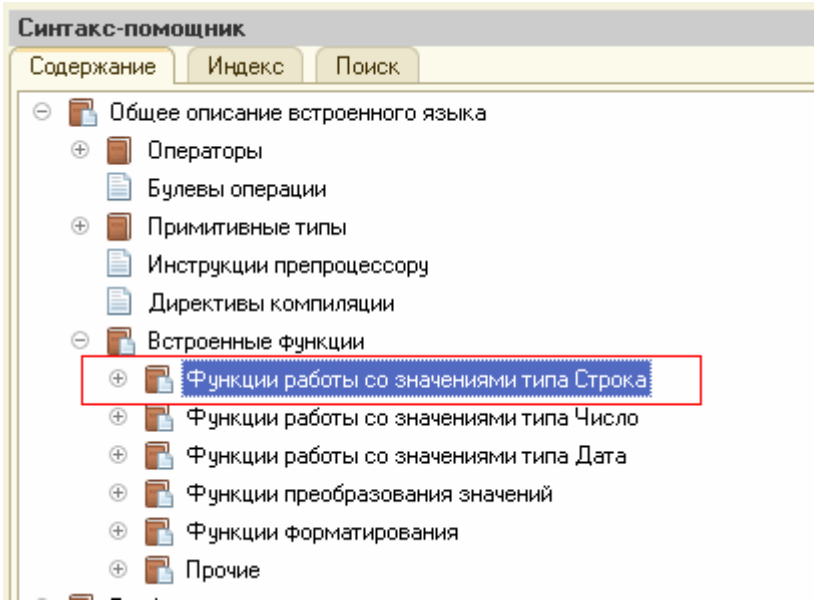


Рис. 1.5.3

Другие закладки более универсальные.

К примеру, Вы забыли, как работает функция *«Найти»* по работе со строками. Вы можете зайти на закладку *«Индекс»* и в окне поиска написать слово *«Найти»*.

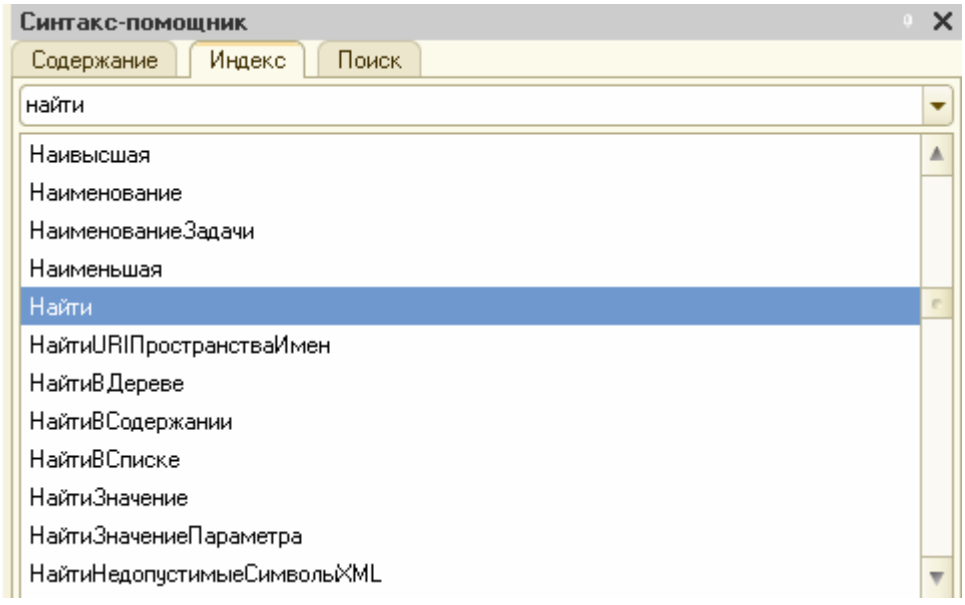


Рис. 1.5.4

И индекс Вам выдал, что существует функция, процедура или метод какого-нибудь объекта с таким именем в принципе. Для того чтобы перейти непосредственно к искомой функции, надо дважды кликнуть мышкой по слову *«Найти»* в поле индекса.

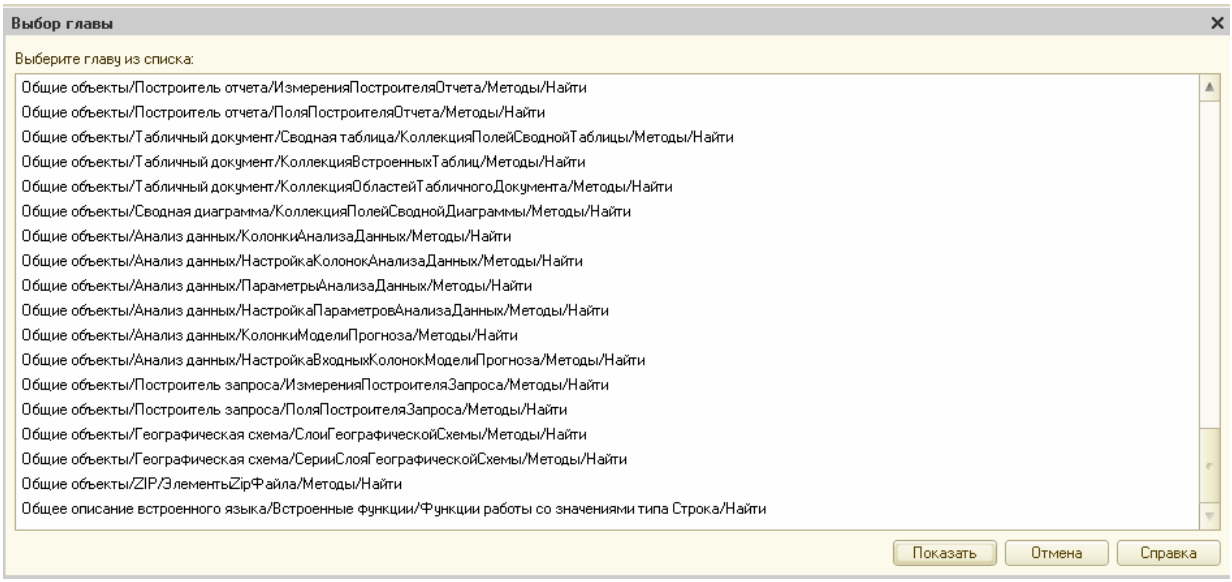


Рис. 1.5.5

Открылось окно, где показаны все методы *«Найти»* для всех существующих объектов 1С. Их очень много, и придется из них найти нужный, ориентируясь на главу списка. Нас интересует самая нижняя строка.

После того как мы ее визуально нашли, кликаете по ней дважды, и Вы перейдете на нужную нам справку.

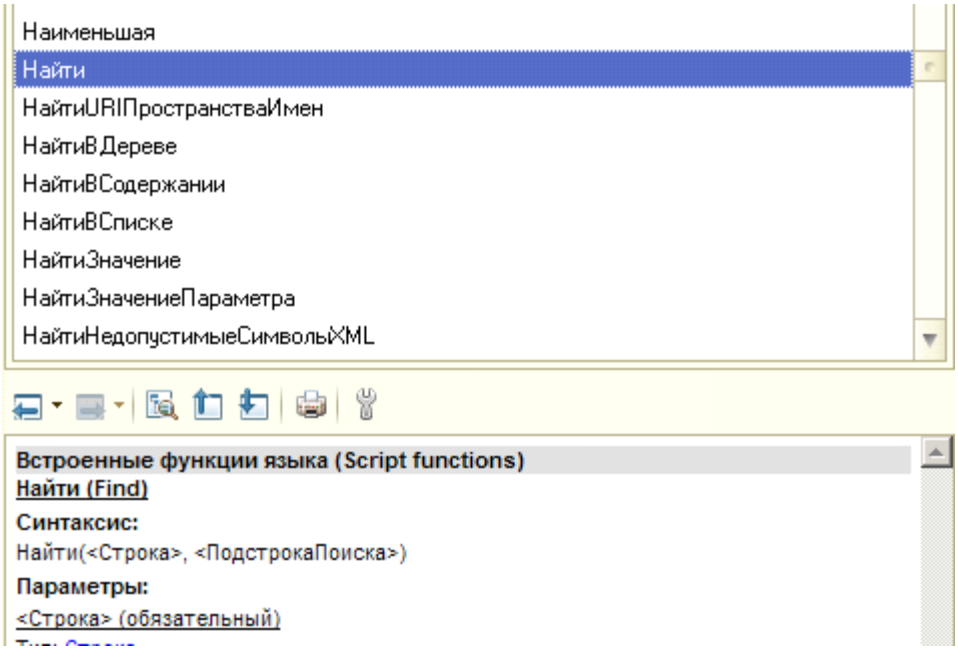


Рис. 1.5.6

Закладка *«Поиск»* выводит все объекты, у которых есть метод *«Найти»*.

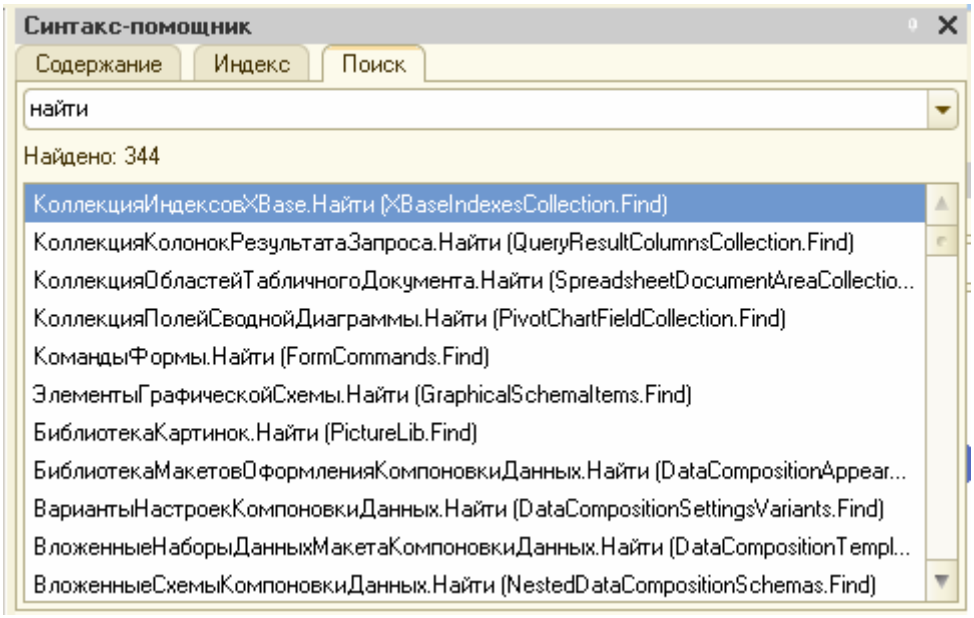


Рис. 1.5.7

Не очень-то удобный способ поиска, поэтому я в основном предпочитаю поиск по индексу.

Часть 6. Свойства конфигурации

Разберем основные свойства конфигурации, которые, так или иначе, влияют на функционал приложения 1С.

Откройте конфигуратор базы, которую Вы создали в первой части этой главы, нажав на кнопку «Конфигуратор» в окне запуска 1С. Откроем конфигурацию и перейдем в её свойства. Для этого необходимо выделить мышкой корень дерева конфигурации (в самом верху), правой кнопкой мышки вызвать контекстное меню и в этом меню нажать на пункт «Свойства» (см. рис. 1.6.1).

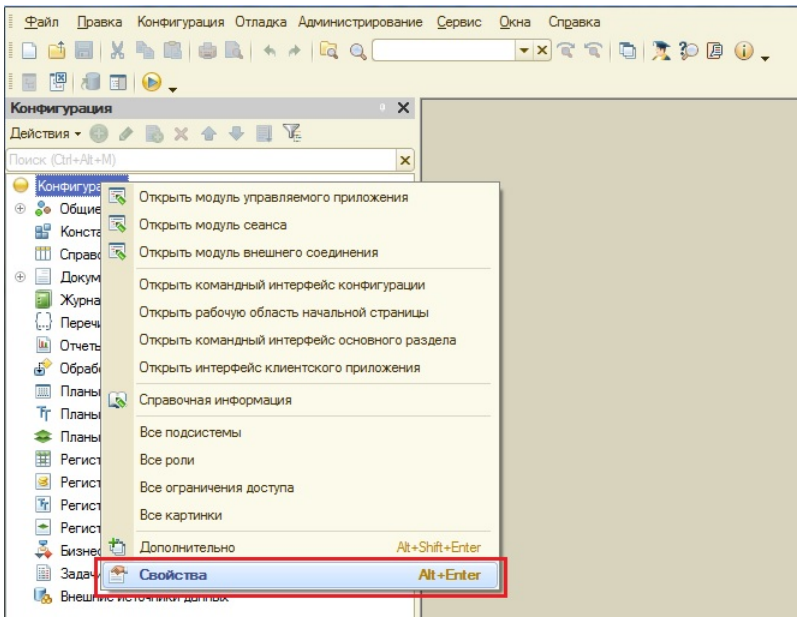


Рис. 1.6.1

В правой части рабочего стола разработчика откроется палитра свойств конфигурации (см. рис. 1.6.2).

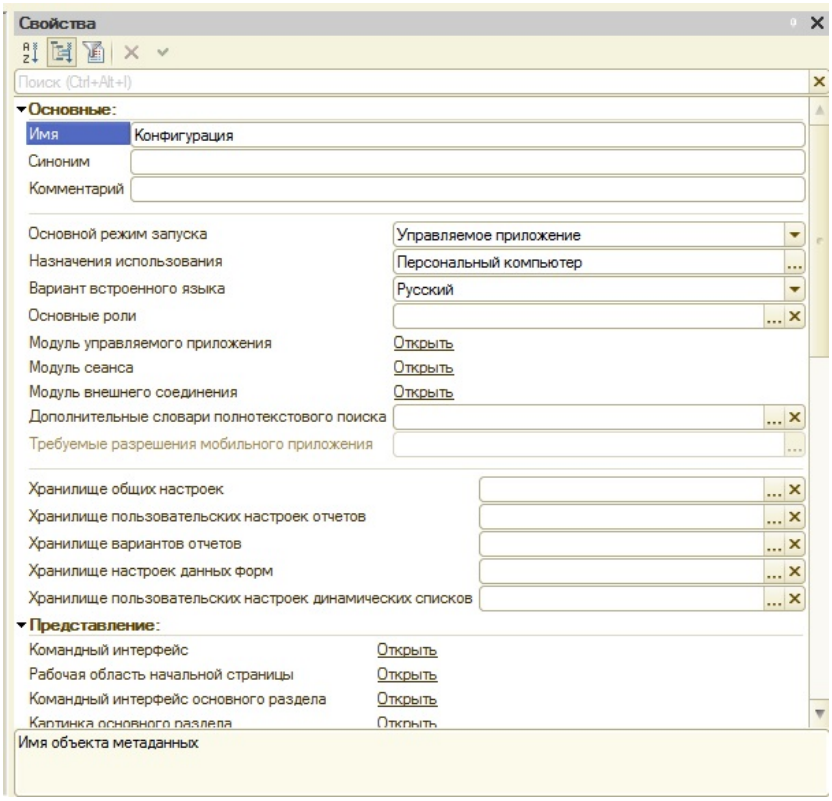


Рис. 1.6.2

В палитре свойств Вы можете задать название Вашей конфигурации и синоним (свойства «Имя» и «Синоним»). Мы не будем разбирать все свойства. А разберем самые значимые для нас. И первое свойство, которое Вам нужно знать, это «Основной режим запуска». Значение этого свойства определяет, в каком виде будет работать наше «1С:Предприятие» - в виде *управляемого приложения* или в виде *обычного приложения*. В это свойство можно установить одно из двух значений: «Управляемое приложение» или «Обычное приложение» (см. рис. 1.6.3).

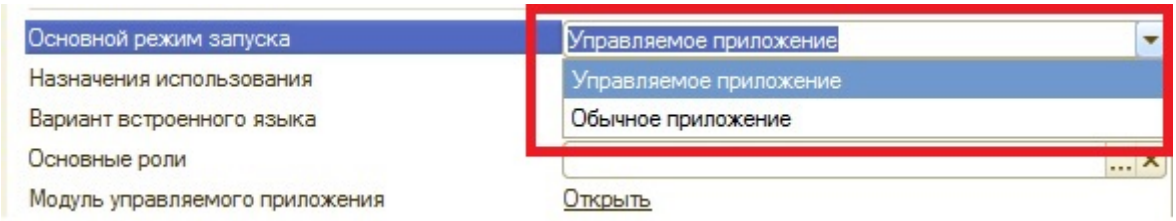


Рис. 1.6.3

Мы оставим у нашей конфигурации свойство «Управляемое приложение». Работа под обычным приложением подробно рассматривается в моей книге «Программирование в 1С за 9 шагов».

Из остальных свойств конфигурации нас интересует свойство «Режим использования модальности», в котором необходимо установить значение «Использовать» (см. рис. 1.6.4).

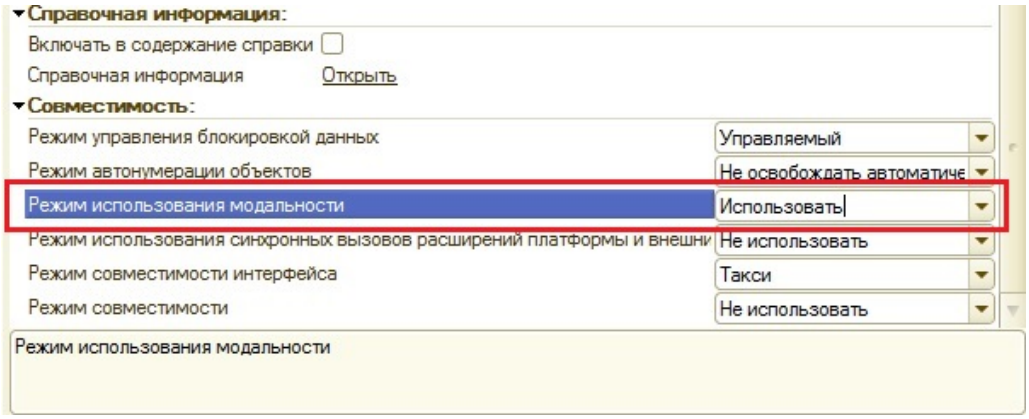


Рис. 1.6.4

После того, как Вы измените что-либо в конфигурации, то в окне конфигурации рядом с заголовком «Конфигурация» появится символ * (см. рис. 1.6.5).

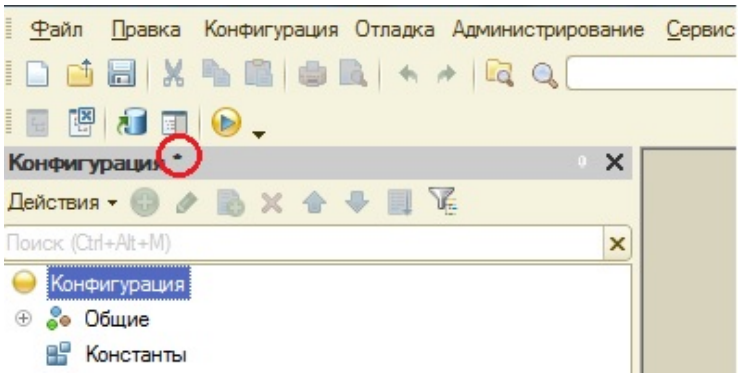


Рис. 1.6.5

Это значит, что Ваша конфигурация отредактирована и её необходимо сохранить. Сохранить конфигурацию можно нажав на кнопку «Сохранить» или комбинацией клавиш Ctrl+S (см. рис. 1.6.6).

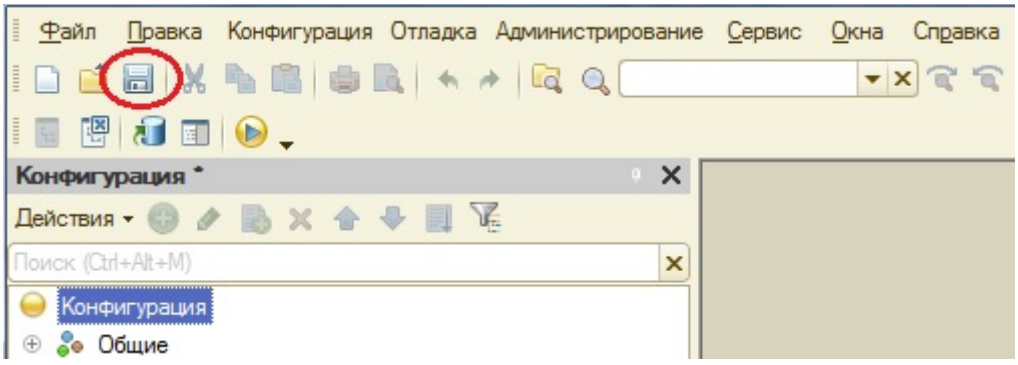


Рис. 1.6.6

После рядом с заголовком «Конфигурация» появится символ «!», это означает, что изменения в конфигурацию внесены, но еще не попали в базу. Для того, чтобы все изменения попали в базу и пользователь начал с ними работать, нужно нажать на кнопку «Обновить конфигурацию базы данных» или клавишу F7 (см. рис. 1.6.7).

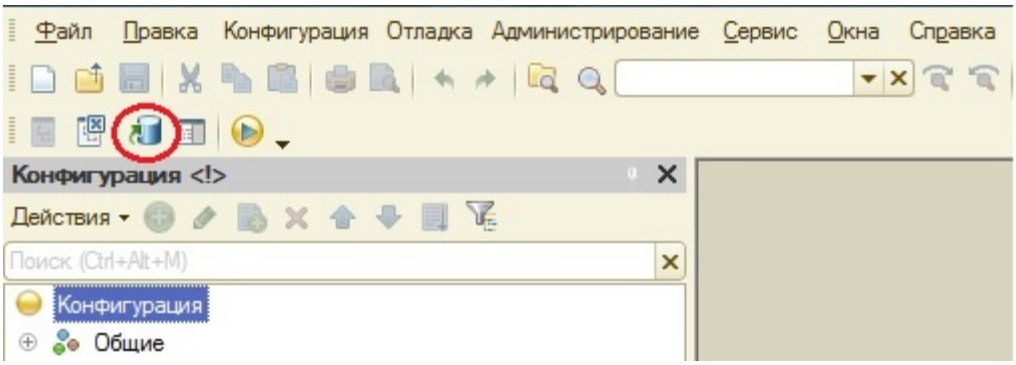


Рис. 1.6.7

Только после этого все изменения будут внесены в базу данных и Вы сможете на них посмотреть в пользовательском приложении.

Часть 7. Создание внешних обработок

Для того, чтобы начать приводить примеры по тем или иным возможностям языка программирования 1С, первоначально мы будем использовать внешние обработки. Что такое обработка? ***Обработка*** **-** это объект конфигурации, предназначенный для реализации различных механизмов обработки информации и сервисных функции. Более подробно вопросы работы с обработками и иными объектами конфигурации мы будем проходить в четвертой главе. В основном мы будем их использовать для изучения механизмов языка программирования.

В этой части изучим, как создаются и сохраняются новые обработки, в которых Вы будете делать все Ваши тренировки. В дальнейшем рекомендую Вам на каждую часть главы (и даже, возможно, на каждый пример) создавать отдельную обработку и самостоятельно прописывать все те примеры, которые даются в книге. Если Вы будете прописывать все приведенные мною примеры и выполнять все те рекомендации, которые я дам в главах книги, то усвоите нужный Вам материал гораздо лучше, чем просто пролистывая книгу.

Итак, как создать новую внешнюю обработку? Для этого в конфигураторе Вашей базы перейдите в меню: *«Файл» - «Новый»*.

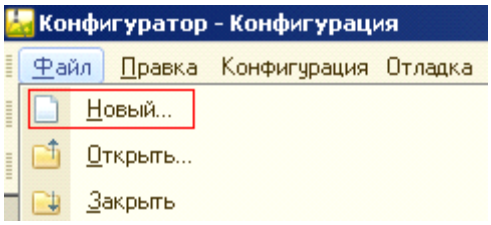


Рис. 1.7.1

Выбираем в имеющемся списке «*Внешняя обработка*» и нажимаем кнопку «*ОК*».

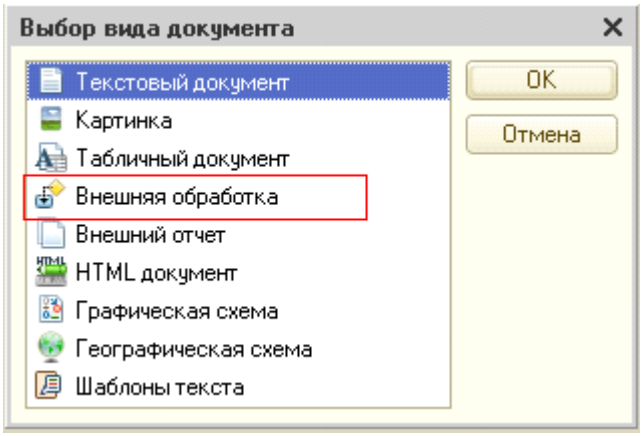


Рис. 1.7.2

Открылась вновь созданная внешняя обработка.

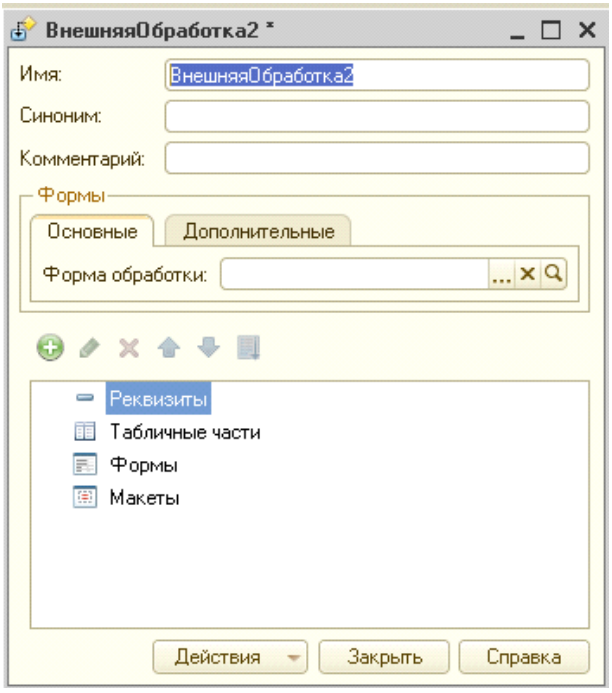


Рис. 1.7.3

Назовите как-нибудь Вашу обработку. Я рекомендую Вам в дальнейшем давать имена им по номеру главы, части и примера.

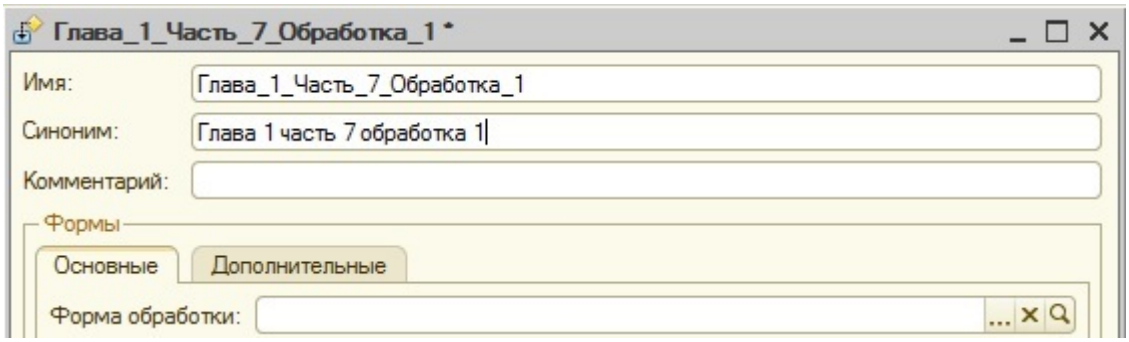


Рис. 1.7.4

Сохраните ее в любое место на жестком диске.

Каким образом Вы будете работать с новой обработкой? Делать это мы будем, используя управляемую форму.

В текущем этапе обучения мы не будем подробно затрагивать работу с управляемыми формами, все интересующие нас вопросы на эту тему мы изучим в пятой главе этой книги.

Просто пока запомните этапы создания новой обработки, формы, команды и её обработчиков команд.

Создайте форму. Для этого наведите курсор на элемент «*Формы*» в списке и выделите этот элемент.

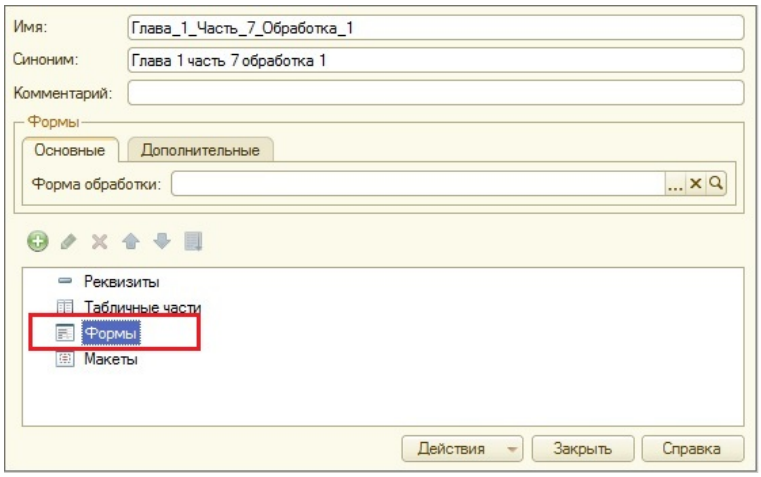


Рис. 1.7.5

Нажмите правую кнопку мышки и выберите пункт *«Добавить»*.

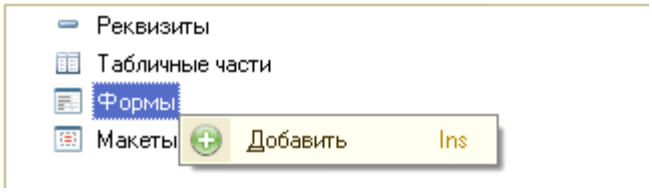


Рис. 1.7.6

В открывшемся окне ничего не меняйте и нажмите кнопку *«Готово»*.

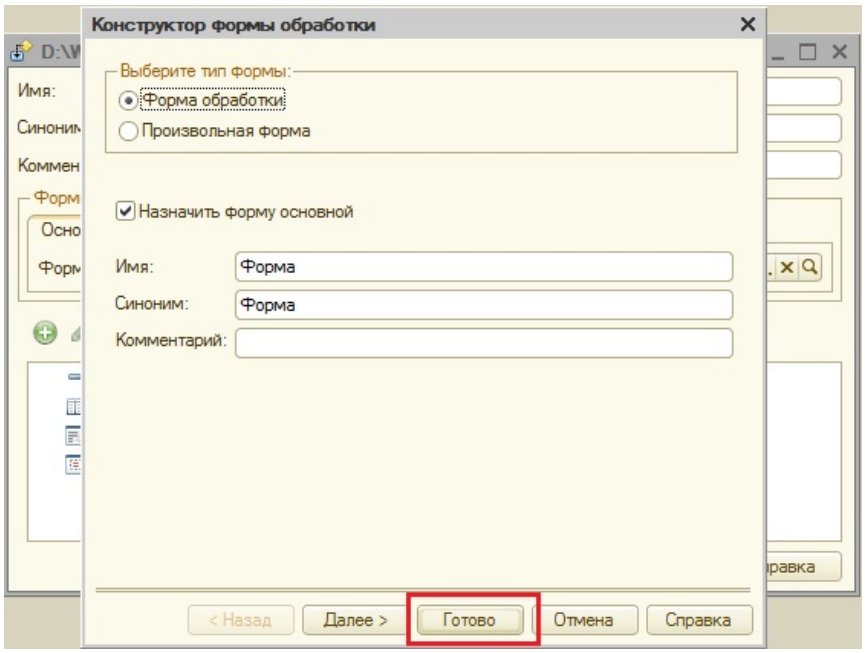


Рис. 1.7.7

Открылась Ваша вновь созданная форма.

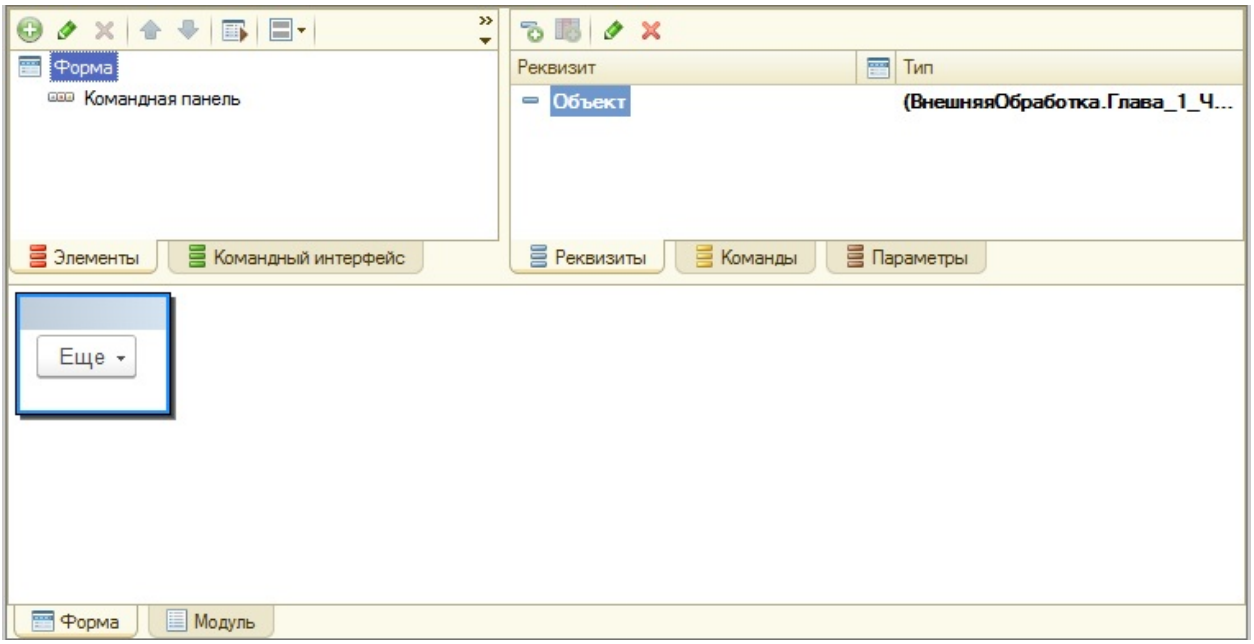


Рис. 1.7.8

Перейдем на закладку «Команды» в правом верхнем окне, и в подзакладке «Команды формы» создадим новую команду, нажав на кнопку «Добавить». Команда будет создана, а справа откроется палитра свойств этой команды (см. рис. 1.7.9).

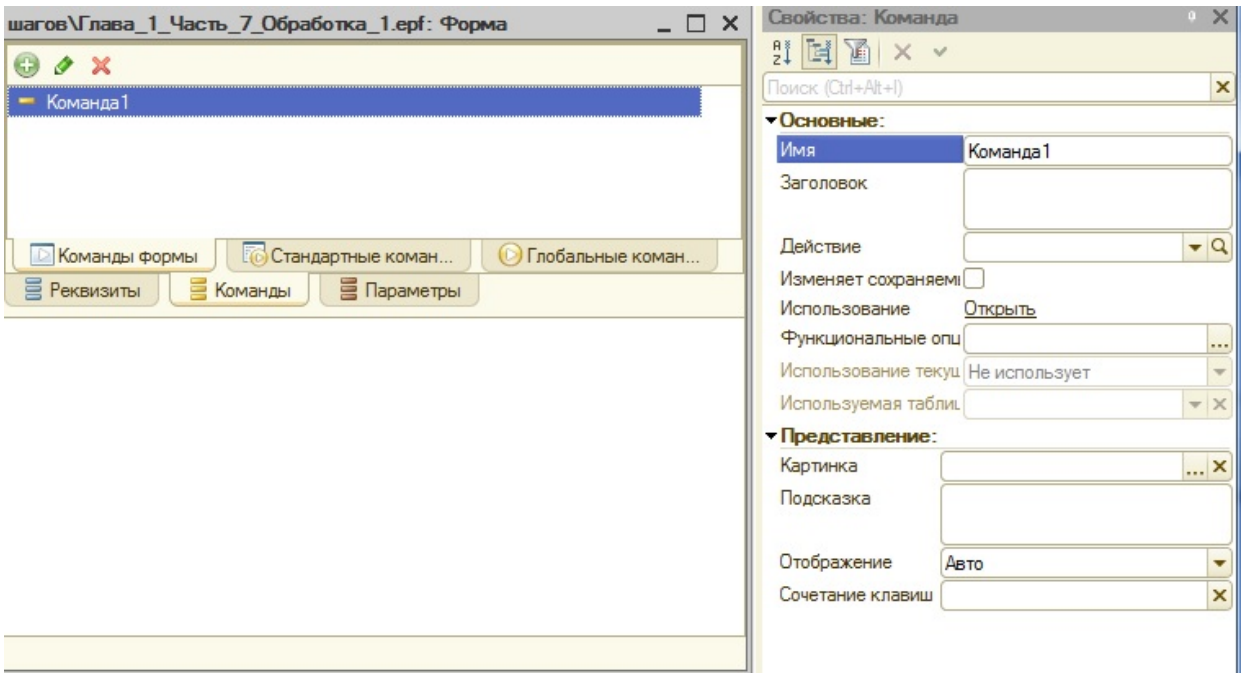


Рис. 1.7.9

Назовем нашу команду «ПриветМир», для этого изменим свойство «Имя» (см. рис. 1.7.9). А после поместим её на форму в виде кнопки. Делается это так: выделяем команду мышкой и «тащим» ее в левое верхнее окно в закладку «Элементы» в командную панель формы (см. рис. 1.7.10 и 1.7.11).

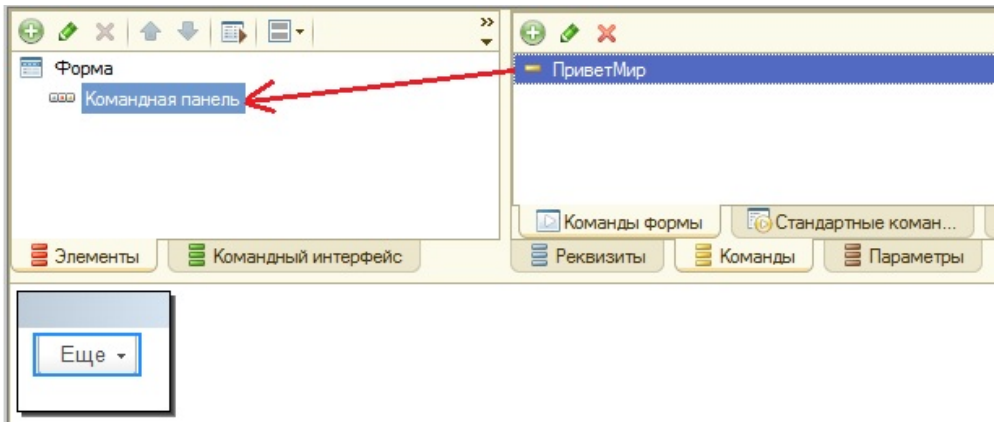


Рис. 1.7.10

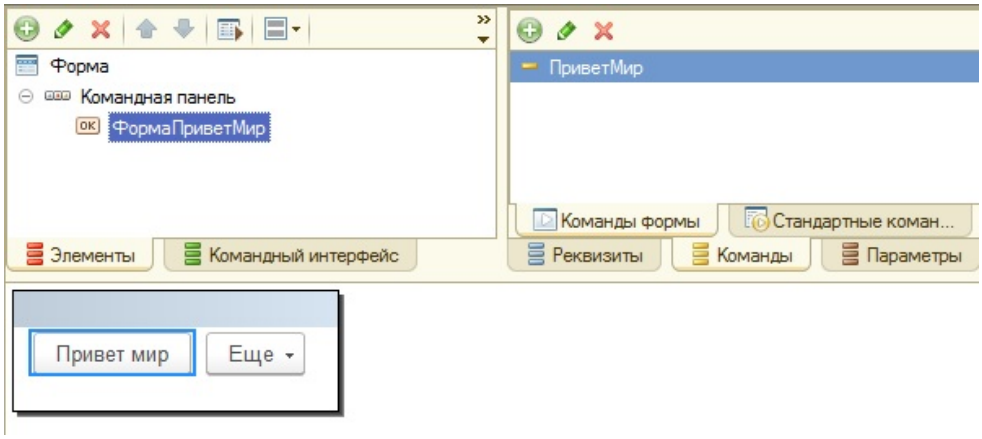


Рис. 1.7.11

Нам осталось создать обработчик команды «ПриветМир». Для этого необходимо опять зайти в палитру свойств команды (чтобы палитра открылась, необходимо кликнуть по команде мышкой, палитра свойств откроется в правой части рабочего стола) и у свойства «Действие» нажать на кнопку «Лупа» (см. рис. 1.7.12).

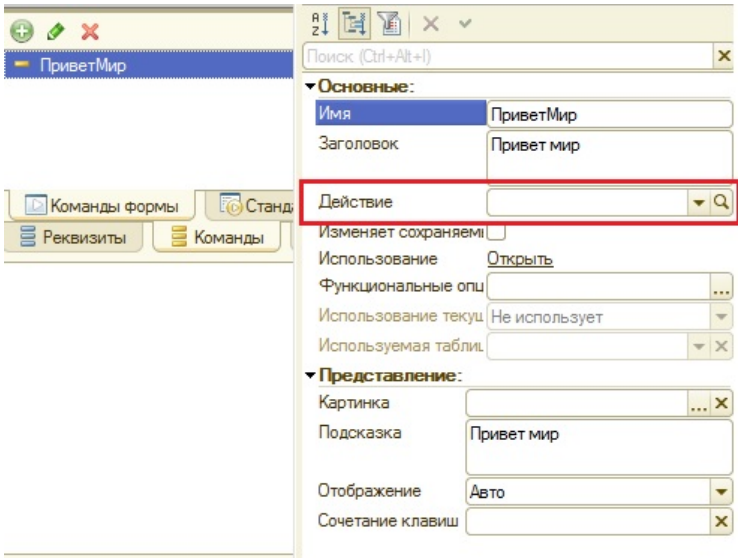


Рис. 1.7.12

Откроется окно, где будет предложено выбрать место создания обработчика команды (см. рис. 1.7.13). Оставляем «Создать на клиенте» и нажимаем кнопку «Ок» (см. рис. 1.7.14).

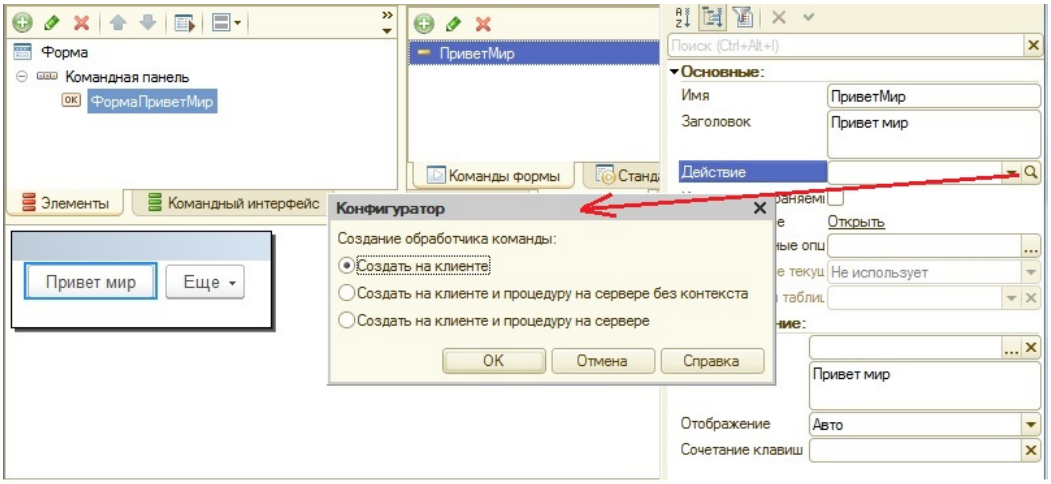


Рис. 1.7.13

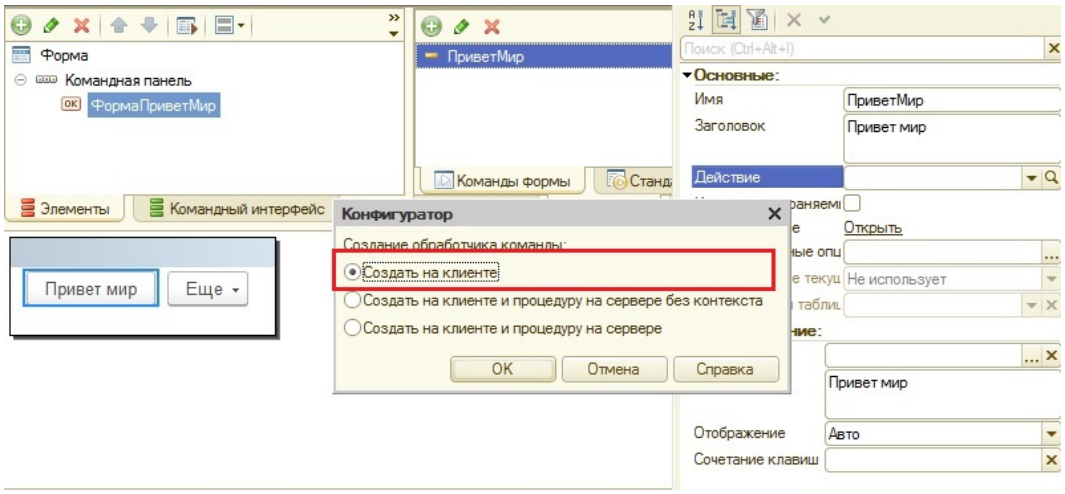


Рис. 1.7.14

После этого в модуле формы будет создана процедура «ПриветМир» (см. рис. 1.7.15).

```
&НаКлиенте
Процедура ПриветМир(Команда)

КонецПроцедуры
```

Рис. 1.7.15

Напишем в ней следующий код (см. листинг 1.7.1).

```
&НаКлиенте
Процедура ПриветМир(Команда)

    Предупреждение("Привет, мир!!!");

КонецПроцедуры
```

Листинг 1.7.1

Это процедура предупреждение и в ней текст «Привет, Мир!». Обязательно сохраните обработку!

Нам осталось посмотреть, как Ваша новая обработка выполнится. Для этого запустим отладку прямо из конфигуратора, нажав кнопку *«Начать отладку»*.

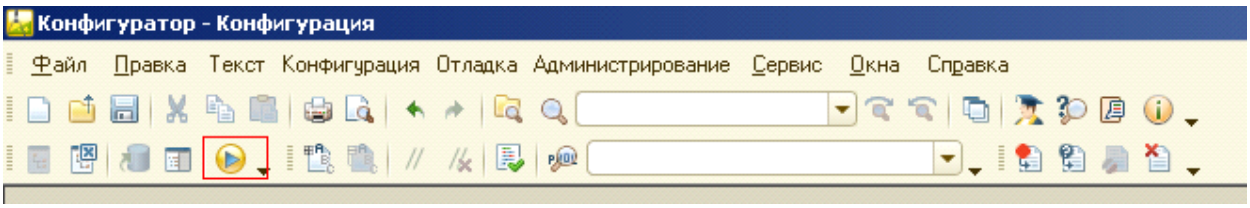


Рис. 1.7.16

Вы увидите, что сразу выйдет окно «1С:Предприятия». В этом окне откройте Вашу только что созданную обработку. Идем: «Главное меню» - *«Файл»* - *«Открыть»* (см. рис. 1.7.17), находим файл внешней обработки в том месте, куда Вы ее сохранили. Открываем. Нажимаем кнопку *«Привет мир»* (см. рис. 1.7.18) - и выйдет Ваше первое предупреждение (см. рис. 1.7.19).

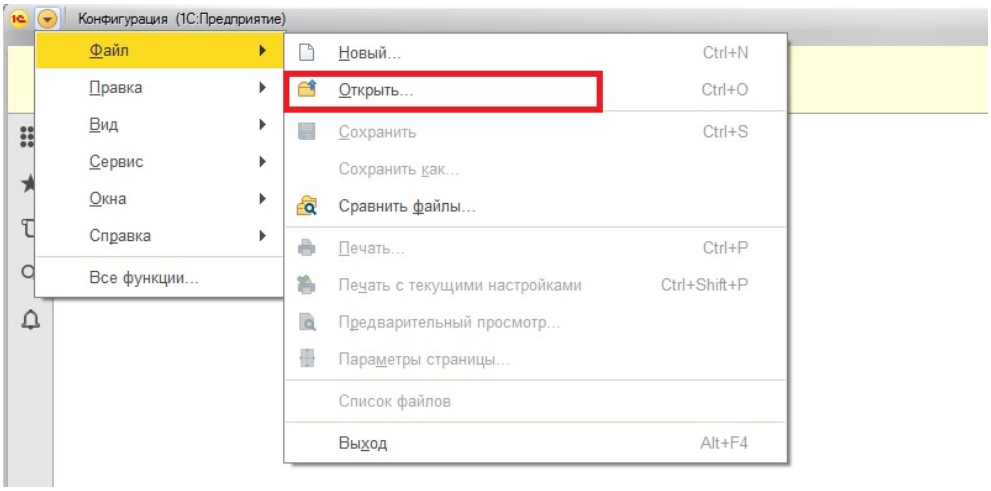


Рис. 1.7.17

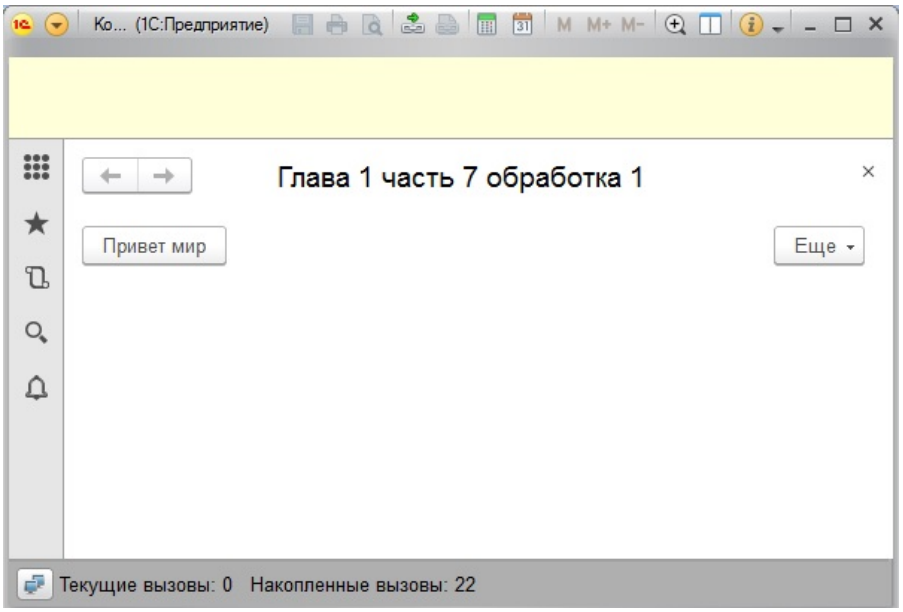


Рис. 1.7.18

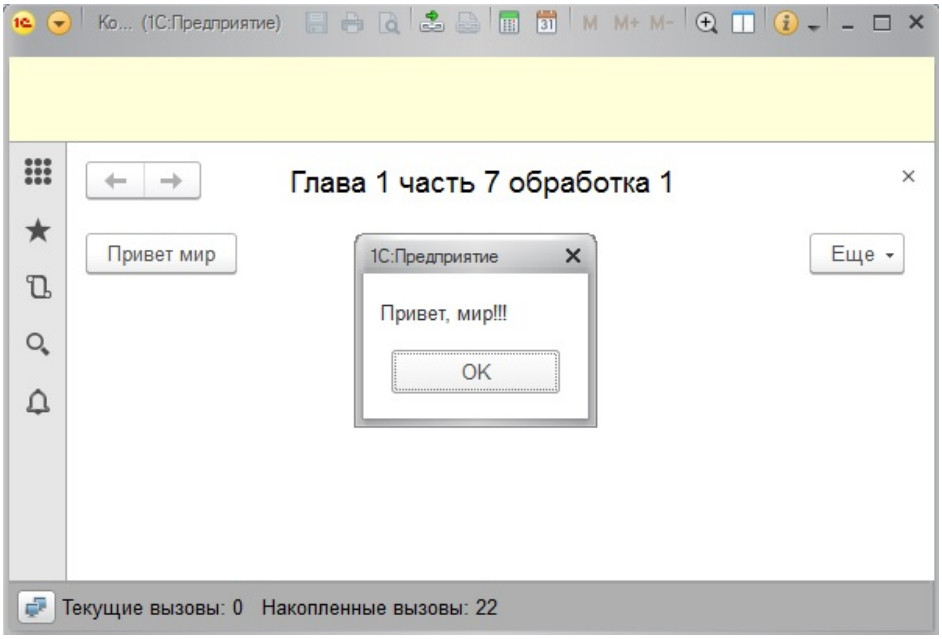


Рис. 1.7.19

Часть 8. Отладка

Очень часто бывает необходимо посмотреть, как работает тот или иной код в программе: какие значения принимают те или иные переменные, как выполняется обход цикла, срабатывают условия и тому подобное. Для этого существует отладка. Для того, чтобы программа остановилась на нужном нам участке кода, достаточно поставить на этот участок точку останова.

Научимся работать с отладкой на примере той обработки, которую сделали в предыдущей части («Привет, Мир»). Для этого переделаем код в команде «ПриветМир» формы обработки. Присвоим переменным некоторые значения.

```
&НаКлиенте
Процедура ПриветМир(Команда)
    А = 10;
    В = "Текст";
    Предупреждение("Привет, мир!!!");
КонецПроцедуры
```

Листинг 1.8.1

Сохраните обработку, запустите её в «1С:Предприятии», но не нажимайте кнопку «Привет, Мир».

Для того, чтобы отладка работала, она должна быть подключена. Если у Вас на панели вместо кнопки «Начать отладку» (см. рис. 1.8.1) кнопка «Продолжить отладку» (см. рис. 1.8.2), то Ваш конфигуратор подключен к отладке.

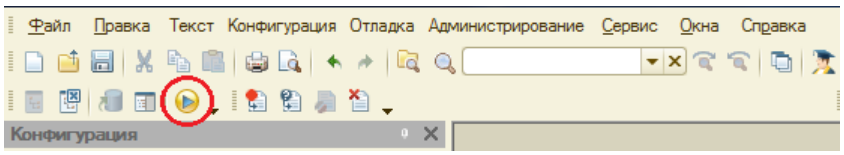


Рис. 1.8.1. Начать отладку

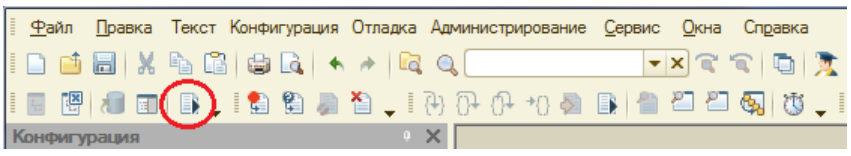


Рис. 1.8.2. Продолжить отладку

Иногда бывают ситуации, когда нужно отладить уже запущенное приложение. В конфигураторе можно посмотреть, подключены какие-либо сеансы к отладке или нет. Для этого нужно воспользоваться окном «Предметы отладки». Открыть это окно можно через «Главное меню» - «Отладка» - «Подключение».

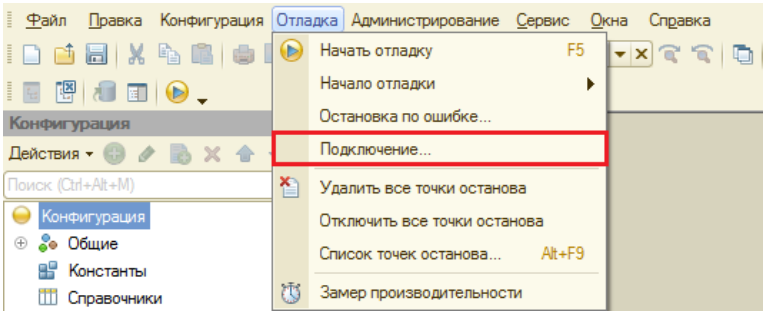


Рис. 1.8.3

Окно разделено на две части: в верхней части все доступные сеансы, а в нижней те сеансы, которые уже подключены к отладке.

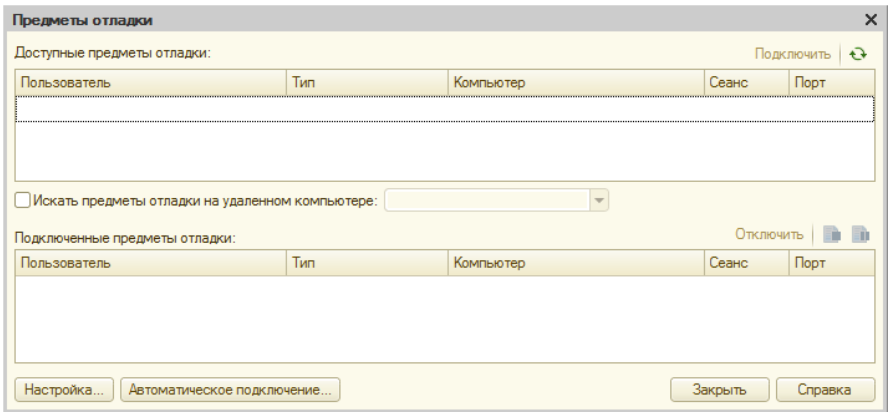


Рис. 1.8.4

Если Вы запустите сеанс Вашей базы не через конфигуратор, а через стартер 1С, то обнаружите, что этот сеанс не появился в верхнем окне. Для этого нужно разрешить отладку в текущем сеансе.

Чтобы разрешить отладку, зайдем в параметры клиентского приложения («1С:Предприятия»).

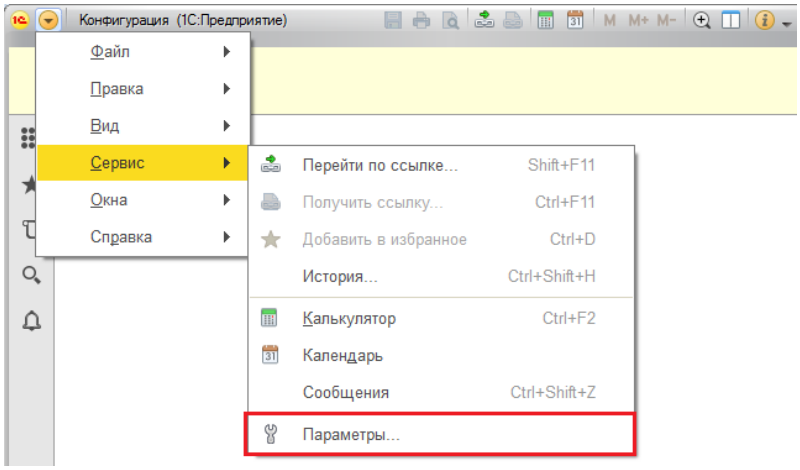


Рис. 1.8.5

В открывшемся окне установите в параметры *Откладка в текущем сеансе* и *Откладка при перезапуске* значение *Разрешена (протокол TCP/IP)*. А после нажмите кнопку «Применить».

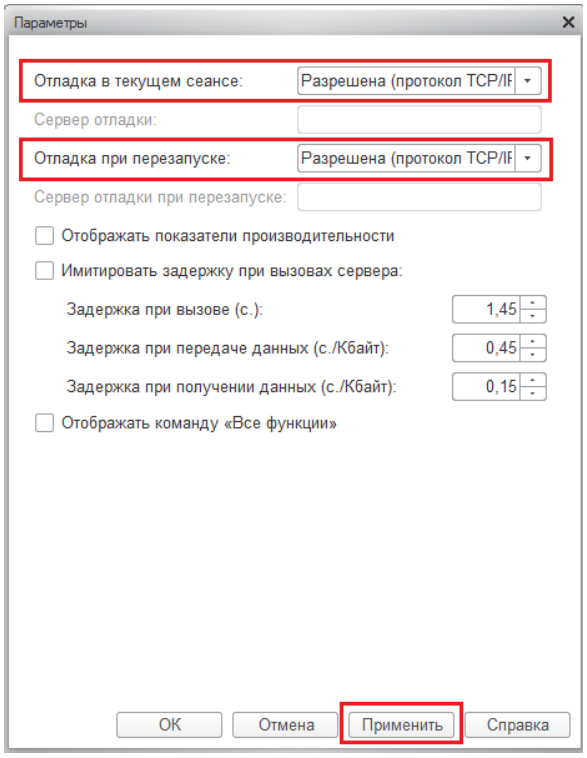


Рис. 1.8.6

Если окно «Предметы отладки» уже открыто, то нажмите кнопку «Обновить» (см. рис. 1.8.7), чтобы подключенный сеанс появился в верхнем окне (см. рис. 1.8.8).

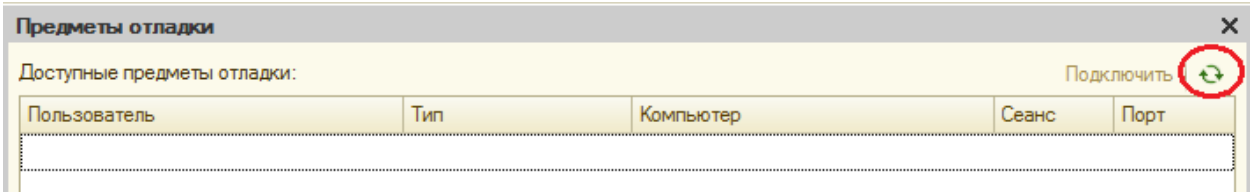


Рис. 1.8.7

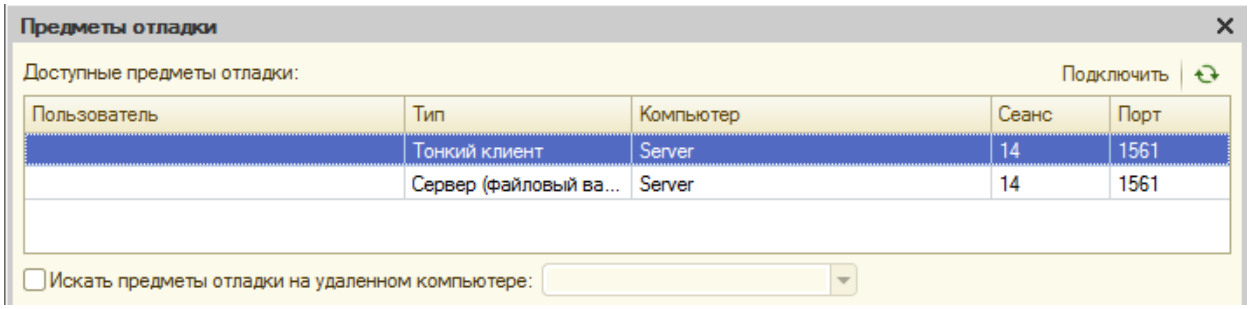


Рис. 1.8.8

У нас они без пользователя, потому что мы его не завели, позже, когда научимся создавать пользователей (в четвертой главе), колонка *Пользователь* будет заполнена. У нас две строки с

отладкой – *тонкий клиент* и *сервер*, пока на это не обращаем внимание, подключаем их обе: выделяем строку и нажимаем кнопку «Подключить».

После этого сеансы исчезнуть в верхнем окне и появятся в нижнем.

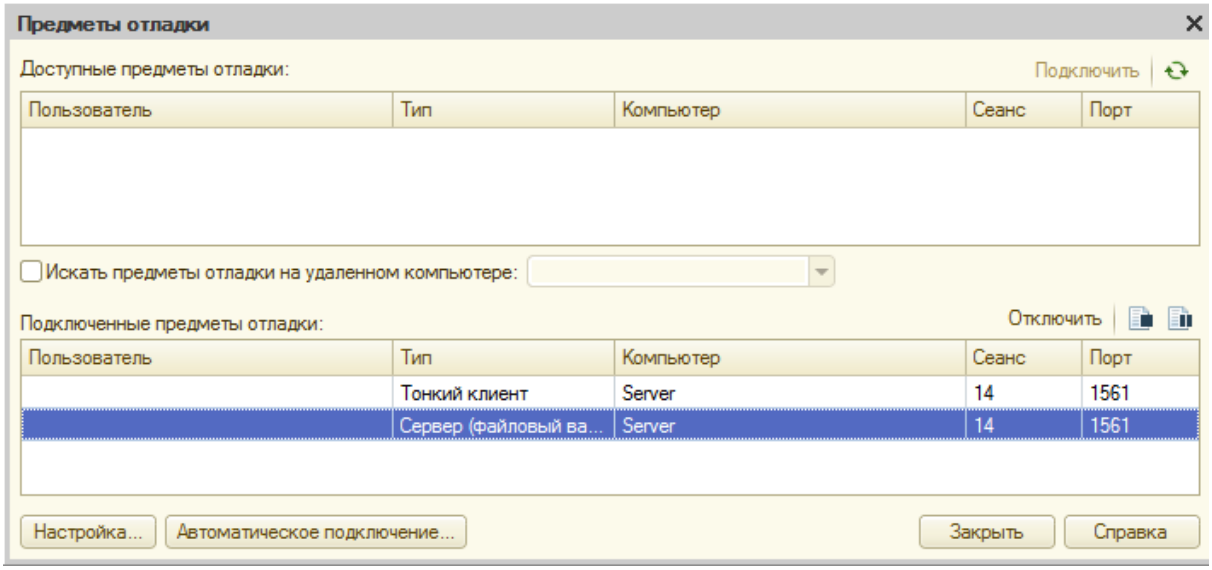


Рис. 1.8.9

Можете спокойно работать с отладкой. Если же пользовательские сеансы не запущены, то их можно запустить из конфигуратора, нажав на кнопку «Начать отладку».

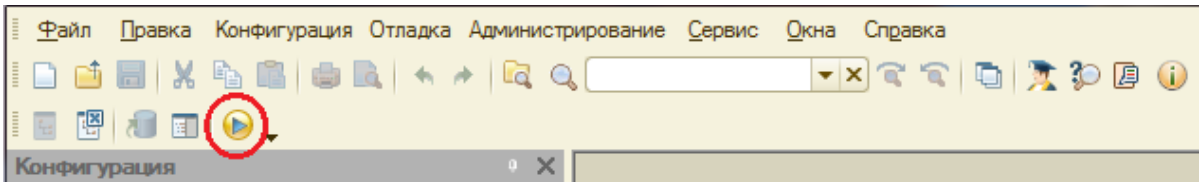


Рис. 1.8.10

Итак, мы научились подключать отладку, теперь научимся с ней работать. Запустим сеанс «1С:Предприятия», используя кнопку «Начать отладку». А после этого в конфигураторе откроем модуль формы предыдущей обработки и установим точку останова в начале процедуры «ПриветМир». Для этого нужно установить курсор на первую строку тела процедуры (где начинается текст `A = 10;`, см. листинг 1.8.1) и нажать на кнопку панели «Точка останова» (см. рис. 1.8.11) или в меню «Главное меню» - «Отладка» - «Точка останова» (см. рис. 1.8.12).

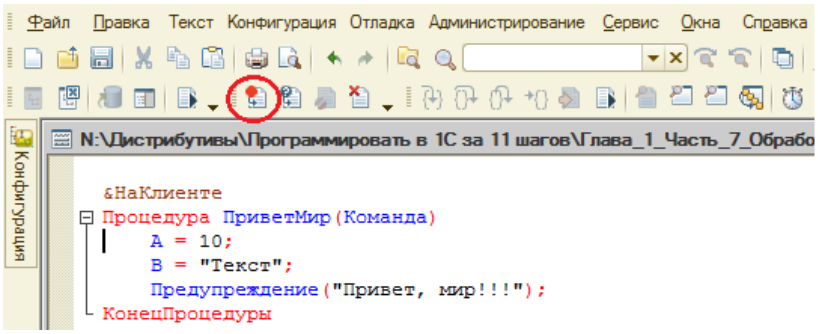


Рис. 1.8.11

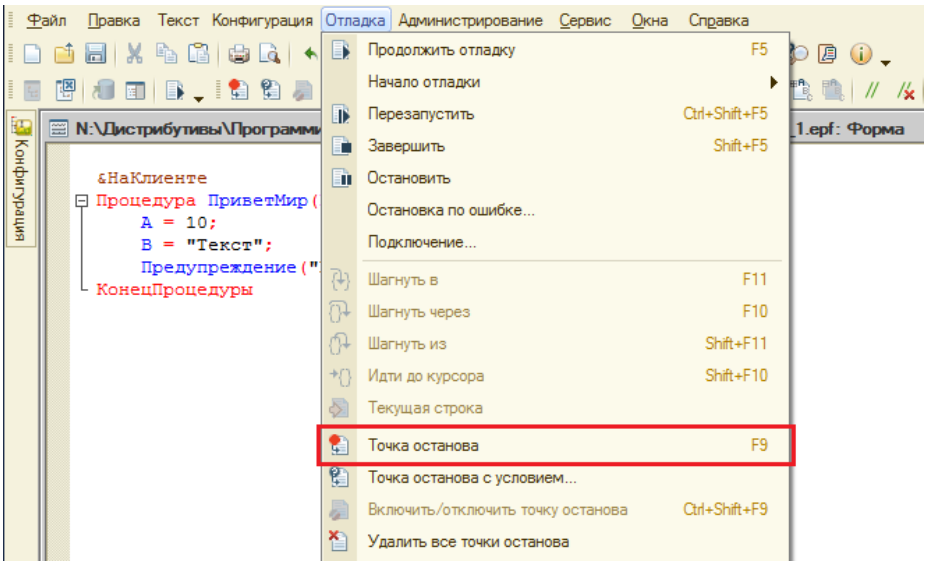


Рис. 1.8.12

После этих действий на поле слева напротив нужной строки кода появится красный кружок, это и есть точка останова.

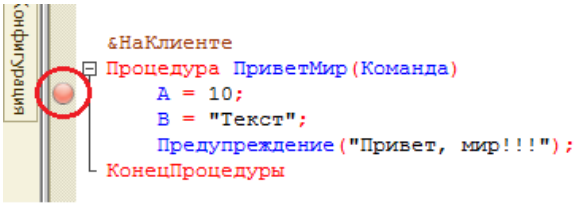


Рис. 1.8.13

Учитывайте следующий момент: точку останова нужно устанавливать на строке, где написан какой-то код. Если установите точку останова на пустой строке (или где комментарий), то она не сработает.

Если сейчас мы перейдем в запущенное «1С:Предприятие», откроем обработку и нажмем на кнопку «Привет, мир», то в конфигураторе, рядом с этой точкой останова появится стрелочка (см. рис. 1.8.14). Это значит, что точка останова сработала и Вы можете начать отладку процедуры. В это время работа «1С:Предприятия» остановится (как бы зависнет).

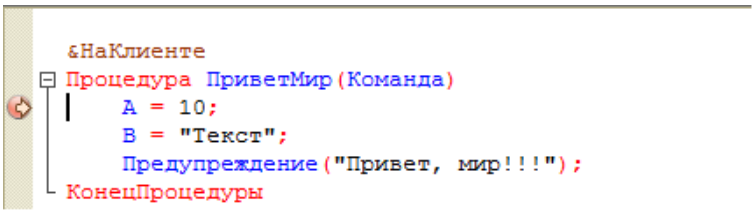


Рис. 1.8.14

Передвигаться по коду можно с помощью кнопок панели:

«Шагнуть в» (F11) – переходим на следующую строку кода. Но если в *текущей* (там, где стрелка) строке процедура или функция, созданные разработчиком (об этом в третей главе), то заходим в этот метод.

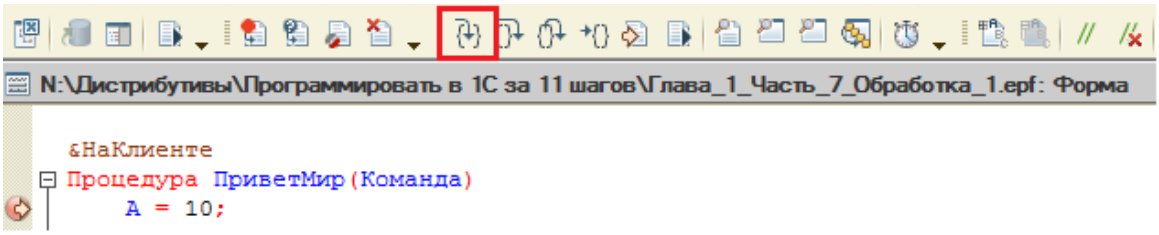


Рис. 1.8.15

«Шагнуть через» (F10) – переходим на следующую строку кода. Если в текущей строке метод, созданный разработчиком, то *не* заходим в этот метод.

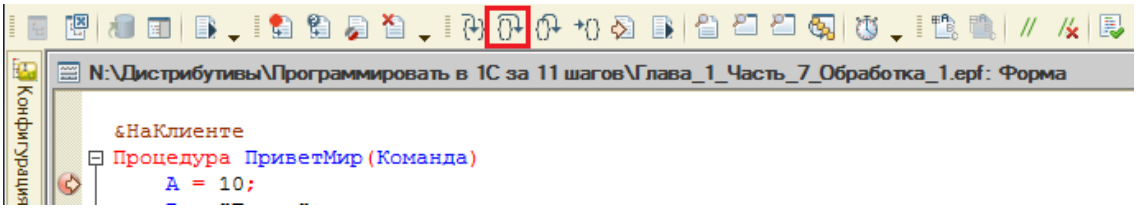


Рис. 1.8.16

«Шагнуть из» (Shift + F11) – если нам не посчастливилось зайти в собственный метод, то при помощи этой кнопки можно из него выйти.

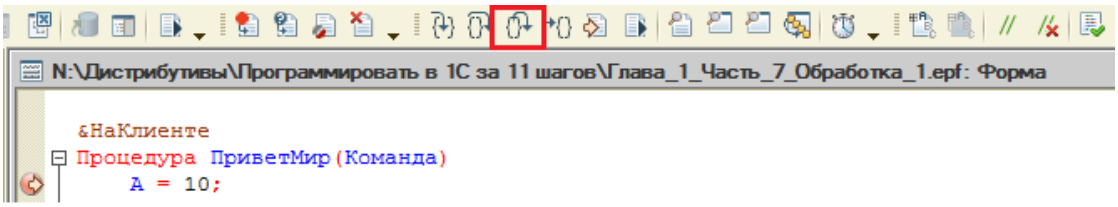


Рис. 1.8.17

«Идти до курсора» (Shift + F10) – можно установить курсор на любой строке кода, и после нажатия на эту кнопку отладка перейдет на неё.

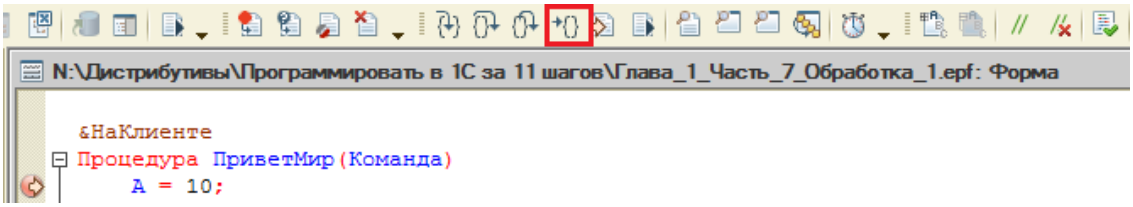


Рис. 1.8.18

Заметьте, код выполнится только после того, как мы пройдем его! Т.е. если мы сейчас на строке, где переменной А присваивается значение 10, то оно еще не присвоилось. Для выполнения этого кода (присвоение значения), нужно перейти на следующую строку.

С движением по отладке разобрались. Потренируйтесь отлаживать Ваш пусть даже маленький код (удобнее всего это делать с помощью клавиш).

Научимся смотреть на значения переменных. Для этого дойдем в отладке до нижней строки кода и посмотрим значения переменных А и В.

```
&НаКлиенте
Процедура ПриветМир(Команда)
    А = 10;
    В = "Текст";
    Предупреждение("Привет, мир!!!");
КонецПроцедуры
```

Рис. 1.8.19

Посмотреть значения той или иной переменной можно двумя способами. Первый - используя окно «Вычислить выражение». Путь к этому окну: «Главное меню» - «Отладка» - «Вычислить выражение».

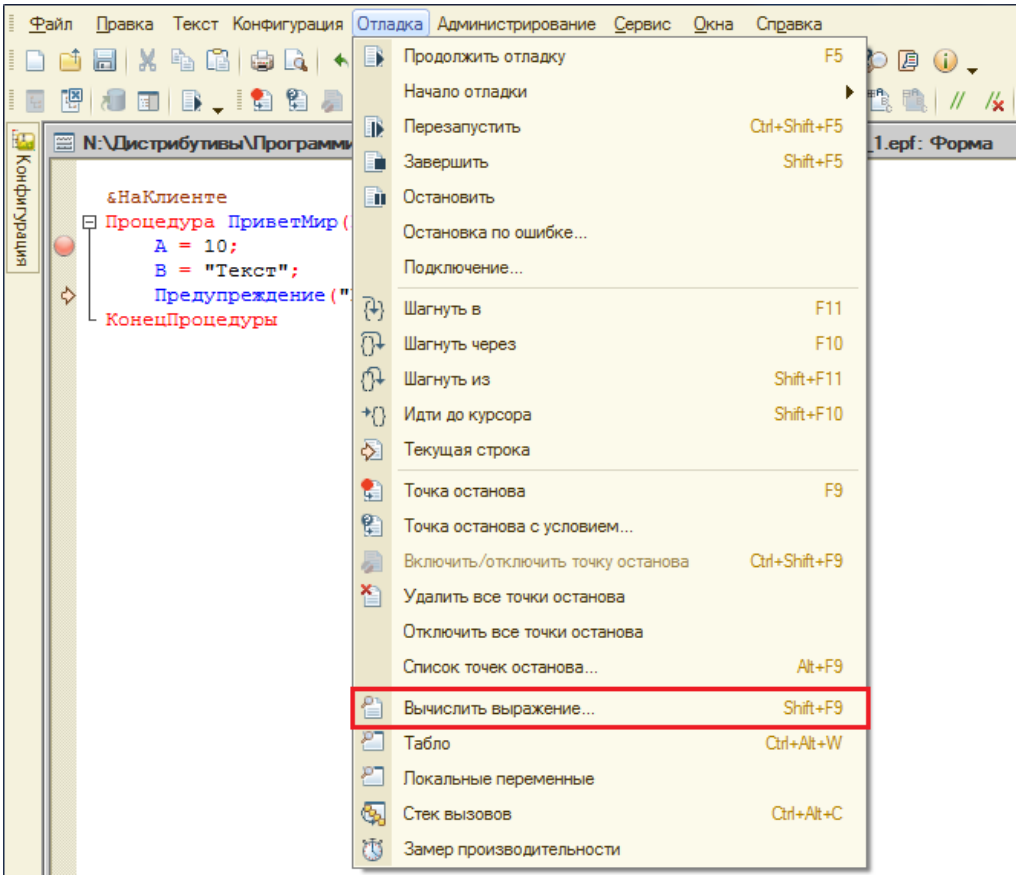


Рис. 1.8.20

В открывшемся окне введите нужную переменную и нажмите на кнопку «Рассчитать».

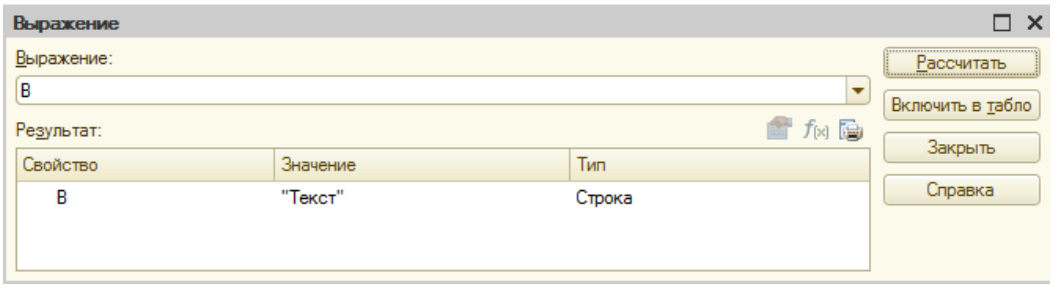


Рис. 1.8.21

Второй способ узнать значение нужной переменной - используя Табло. Оно открывается по следующему пути: «Главное меню» - «Отладка» - «Табло».

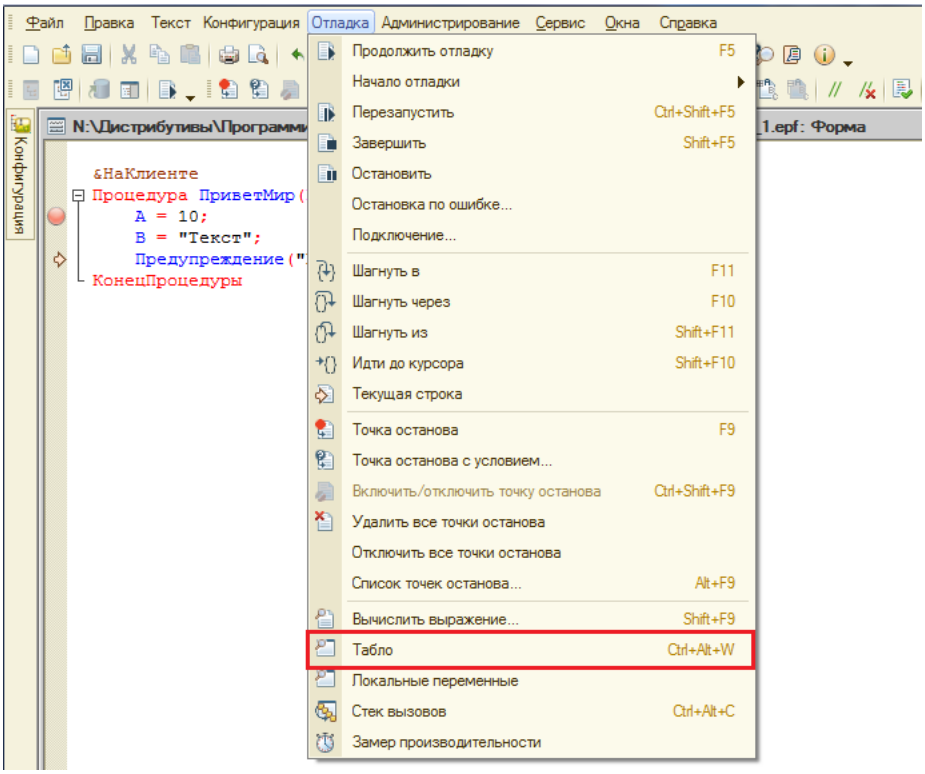


Рис. 1.8.22

Табло откроется внизу конфигуратора, и Вы можете в него вводить любые переменные (в колонку *Выражение*).

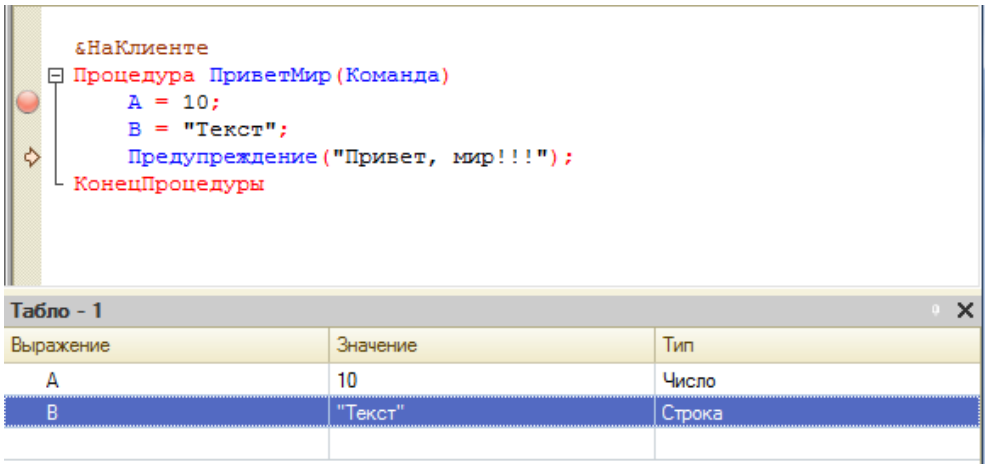


Рис. 1.8.23

И в табло, и в «Вычислить выражение» можно производить различные действия над переменными.

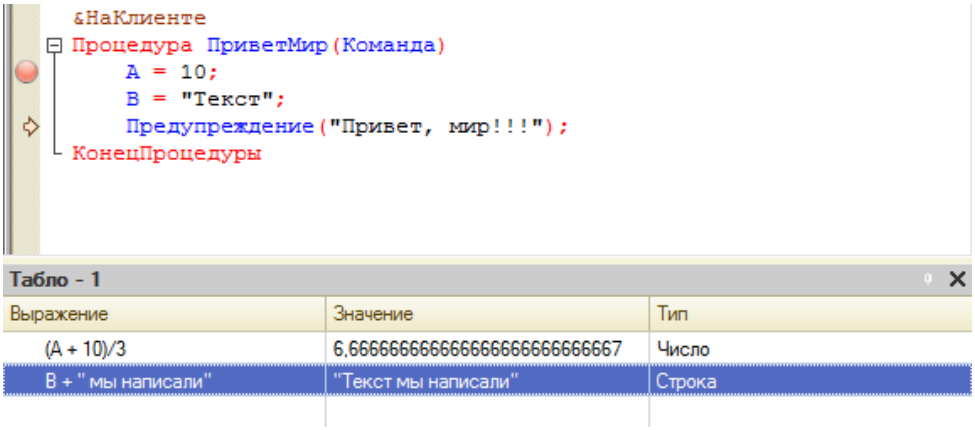


Рис. 1.8.24

Эта вся информация об отладке, которая Вам нужна на данном этапе обучения. Работайте с отладкой, программируйте в отладке, если Вы не понимаете, как работает тот или иной код, то Вам прямой путь в отладку. Это очень мощный и нужный инструмент разработчика.

Резюме

Итак, мои уважаемые будущие коллеги, той информации, которую я Вам предоставил в этом уроке, вполне достаточно для того, чтобы начать делать первые шаги в изучении языка программирования 1С. Задача этой главы подготовить Вас к освоению нового, уже более интересного материала. Потренируйтесь создавать обработки, формы и примитивные команды. Не думайте пока, для каких целей нужны те или иные вещи, а просто набейте руку. Потом Вы поймете, что для чего нужно. Попробуйте поработать с отладкой. После этой главы можете смело переходить ко второй главе, которая будет уже более глубокой и содержательной. Всего доброго и продуктивного обучения.

Глава 2. Примитивные типы и переменные

Часть 1. Примитивные типы

Сначала рассмотрим примитивные типы, используемые в языке программирования 1С.

Всего существует пять основных примитивных типов языка программирования 1С: ***число, строка, дата, булево*** (истина-ложь), а также типы ***Неопределенно*** и ***NULL***.

Примитивные типы
Число Строка Дата Булево Неопределено NULL

Рис. 2.1.1

Для тех, кто изучал другие языки программирования: в языке программирования 1С не надо явно задавать тип переменной. Достаточно просто указать для данной переменной какое-нибудь значение (число, строку, булево значение), и она автоматически присвоит себе данный тип. В то же время при создании реквизитов объектов конфигурации необходимо тип задать явно, но об этом Вы узнаете в четвертой главе.

Итак, первый примитивный тип ***Число***. Это могут быть любые числа: целые и дробные числа, положительные и отрицательные.

Например: **23**; **23.4**; **233.33333.** Можно использовать любые цифры, дробная часть разделяется символом «.» (точка). По справочной информации, максимальная разрядность числа 38 знаков. Так как основные задачи в 1С не связаны с астрономическими расчетами и т.п., то этой разрядности будет достаточно.

Над типом ***Число*** определены главные математические операции: сложение, вычитание, умножение, деление. Вот как они выглядят:

```
2 + 2
3 - 4
5*6
7/2
```

Рис. 2.1.2

Подробно об операциях с числами Вы узнаете в части 4 этой главы.

Второй примитивный тип - это тип ***Строка***. ***Строка*** - это любой набор символов заключенный в кавычки: "" . Строки могут быть однострочные и многострочные (текст).

Однострочная строка будет выглядеть так:

```
"Я помню чудное мгновенье"
```

Рис. 2.1.3

А многострочная строка задается следующим образом:

```
"Я помню чудное мгновенье
 | Передо мной явилась ты
 | Как мимолетное виденье
 | Как гений чистой красоты"
```

Рис. 2.1.4

Т.е. каждая из строк не заканчивается кавычкой, а каждая последующая начинается с символа «|» (вертикальная черта).

Для строк есть операция сложения, выполняемая оператором *«Плюс»* (+). Это значит, что при сложении строки *«Мама мыла »* и строки *«раму»* получится строка "Мама мыла раму":

"Мама мыла" + "раму" = "Мама мыла раму".

Пустая строка задаётся следующим образом: "" (две сдвоенных кавычки без пробела).

Подробнее об операциях со строками Вы узнаете в части 5 этой главы.

Третий примитивный тип – это тип ***Дата***. Значения данного типа содержат дату григорианского календаря с 1 января 1 года нашей эры. Так что если кто-то изобретет машину времени, то сможет на языке 1С написать программу для ведения бухгалтерии Понтия Пилата. Дата в языке программирования 1С задается в виде строки цифр в следующем формате 'ГГГГММДДччммсс',

где:

ГГГГ – четыре цифры года включая тысячелетие и век, например, 2013 или 1997;

ММ - две цифры месяца, например, 02 – это февраль, а 12 – декабрь;

ДД - две цифры даты, например, 01 – первое число, или 23 – двадцать третье число;

чч - две цифры часов в 24-часовом формате, к примеру: 01 – час ночи, 13 – час дня;

мм – две цифры минут, к примеру: 30 – 30 минут, 01 – 1 минута;

сс – две цифры секунд, 20 - это 20 секунд, и так далее.

Также дату можно задавать в таком виде: 'ГГГГММДД'. При данном формате часы, минуты и секунды будут равны нулю.

Пустая дата задается следующим образом: '00000000000000' .

Таким образом, если Вы родились 20 января 1989 г., то задать данную дату на языке программирования 1С можно двумя способами:

'19890120' или '19890120000000'. А если Вы хотите просто задать время 1 час 30 минут, то будет следующий вид: '00010101013000'. Подробно об операциях с датами Вы узнаете в части 6 этой главы.

Четвертый примитивный тип - это тип ***Булево***. ***Булево*** - это логический тип, у которого всего два значения - *Истина* и *Ложь*.

```
Истина
Ложь
```

Рис. 2.1.5

На данном этапе обучения этого достаточно. Более подробно об операциях с данным типом Вы узнаете в соответствующей части настоящей главы.

Ну и напоследок осталось два типа - это ***NULL*** и ***Неопределено***.

Тип ***NULL*** – значения данного типа используются исключительно для определения отсутствующих значений при работе с запросами. С языком запросов мы научимся работать в девятой главе.

Тип ***Неопределено*** применяется, когда надо использовать пустое значение, не принадлежащее ни к одному другому типу. Данный тип так и задается.

```
А = Неопределено;
```

Итак, мы разобрали примитивные типы языка программирования 1С.

Переменные

Прежде чем начать изучать переменные определенных примитивных типов, определимся с понятием «Переменная».

Что такое переменная? Научным языком, ***Переменная*** - это идентификатор определенный некоторым типом, способный менять свое значение в ходе выполнения программы. Каким образом объявляются переменные в языке программирования 1С? Делается это двумя способами:

Первый: переменная может быть объявлена явным образом, тогда она сразу начинает свое существование. Пишем оператор *Перем*, название этой переменной, потом либо запятая и название другой переменной, либо точка с запятой.

Рис. 2.1.6

Данное явное задание переменных должно быть либо в начале модуля, либо в начале процедуры или функции.

Второй способ: переменная может быть объявлена неявным образом. Тогда просто пишем название переменной и присваиваем ей какое-нибудь значение.

Например:

Рис. 2.1.7

Символ «=» (равно) - это операция присваивания. То есть какой-то переменной присваивается определенное значение.

Обращаю внимание тех, кто изучал другие языки программирования. Непосредственно при написании кода в конфигураторе 1С не нужно явно задавать тип переменной. Достаточно просто указать для данной переменной какое-нибудь значение (число, строку, булево значение), и данная переменная автоматически присвоит себе данный тип. Кроме того, в языке 1С переменная (явно или неявно заданная) может по желанию разработчика в любом месте поменять свой тип.

Часть 2. Примитивные операции вывода информации

В этой части Вы узнаете о самых простых способах вывода информации в программе 1С.

Процедура «Сообщить»

Первый пример, который мы рассмотрим, будет хрестоматийным, с него начинают изучение почти все языки программирования. Это вывод сообщения *«Привет, Мир!»*. Сейчас текст *«Привет, Мир!»* мы выведем в окно сообщений программы 1С. И сделаем мы это с помощью оператора ***Сообщить***.

Создайте обработку, форму, команду формы «ВывестиСообщение», разместите команду на форме, и создайте обработчик этой команды на клиенте (подробно об этом см. глава 1, часть 7, стр. 38), напишите в обработчике команды следующий код:

```
&НаКлиенте
Процедура ВывестиСообщение(Команда)

    Сообщить("Привет, мир!");

КонецПроцедуры
```

Листинг 2.2.1

Обращаю Ваше внимание, что в конце всех операторов всегда необходимо ставить точку с запятой. Это операнд, который дает знать компилятору 1С, что данный блок программы завершен и надо переходить к следующему блоку.

Сохраним и запустим обработку в «1С:Предприятии».

Мы видим, что в нижней части экрана открылось окно *«Сообщения»* и в нем вышел наш текст *«Привет, Мир!»*.

Рис. 2.2.1

Таким образом мы видим, что процедура ***Сообщить*** - это встроенная процедура Конфигуратора 1С, которая выводит текстовую информацию в окно сообщений.

Процедура «Предупреждение»

Еще один интересный оператор вывода информации - это процедура ***Предупреждение***. Данная процедура выводит на экран окно предупреждения с нужным Вам текстом.

Обращаю Ваше внимание, что данная процедура работает только тогда, когда в свойство конфигурации «Режим использования модальности» установлено значение «Использовать» (см. глава 1, часть 6, стр. 34). Иначе нужно использовать процедуру «ПоказатьПредупреждение». На данном этапе обучения Вам не стоит особо обращать внимание на этот режим. Более подробно о работе с выключенным режимом использования модальности рассказывается в моей книге «Основы разработки в 1С: Такси».

Создайте новую команду формы «ВывестиПредупреждение», разместите её на форме, создайте обработчик команды и внутри обработчика команды «ВывестиСообщение» напишите следующий код:

```
&НаКлиенте
Процедура ВывестиПредупреждение(Команда)

    Предупреждение("Привет, мир!");

КонецПроцедуры
```

Листинг 2.2.2

Теперь перейдите в «1С:Предприятие», запустите обработку и нажмите кнопку «Вывести предупреждение».

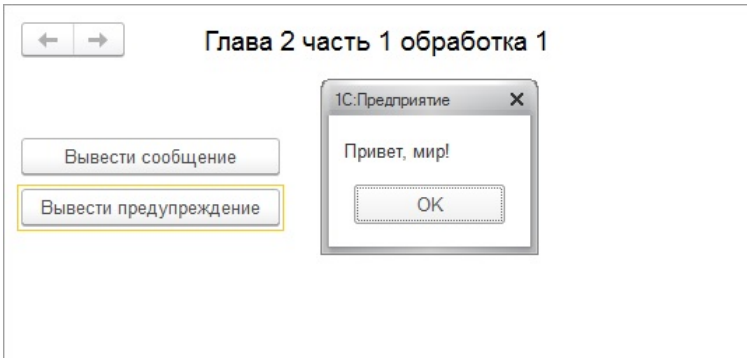


Рис. 2.2.2

Видите: вышло окно с кнопкой *«ОК»* и нашей надписью.

Очень удобно выводить такое предупреждение, когда надо сообщить пользователю то, что он обязательно должен увидеть, а то иногда на окно сообщений внизу не обращают внимания.

Какие еще особенности есть у данной процедуры?

Это *таймаут* и *заголовок окна*. Таймаут задает время в секундах, в течение которого окно будет открыто, а заголовок - это непосредственно заголовок данного окна. По умолчанию таймаут равен 0, это означает, что время не ограничено.

Измените обработку.

```
&НаКлиенте
Процедура ВывестиПредупреждение(Команда)

    Предупреждение("Привет, мир!",5,"Наш заголовок");

КонецПроцедуры
```

Листинг 2.2.3

Запустите ее заново и посмотрите, что получилось:

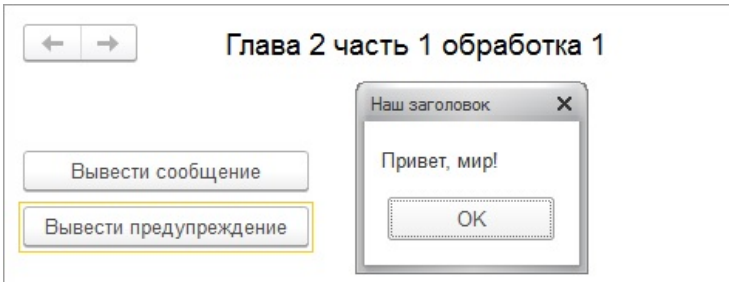


Рис. 2.2.3

Окно откроется, и через несколько секунд закроется.

А можно написать и так:

```
&НаКлиенте
Процедура ВывестиПредупреждение(Команда)

    Предупреждение("Привет, мир!",,"Наш заголовок");

КонецПроцедуры
```

Листинг 2.2.4

Обращаю Ваше внимание, что пока пользователь не нажмет кнопку *«ОК»*, дальнейшее выполнение кода программы не происходит. Для того чтобы это увидеть, напишите процедуру «Сообщить» после процедуры «Предупреждение»:

```
&НаКлиенте
Процедура ВывестиПредупреждение(Команда)

    Предупреждение("Привет, мир!",5,"Наш заголовок");
    Сообщить("Привет, мир!");

КонецПроцедуры
```

Листинг 2.2.5

Убедились?

Резюме

Мы научились работать с примитивными операциями вывода информации в программе 1С. В дальнейших уроках Вы научитесь выводить информацию гораздо более удобными способами, но все равно данные методы Вам пригодятся в будущей работе.

Часть 3. Переменные типа Булево

Перед тем, как мы начнем изучать числа, строки и даты более подробно, нам нужно будет освоить одну маленькую, но очень важную тему. Это переменные типа *Булево*. Создайте новую обработку, форму, команду формы и т.д.

Булево - это признак истинности или ложности какого-нибудь утверждения.

Например, утверждение *«5 меньше 7»* истинно.

А утверждение, что 5 = 7 ложно.

Как же присваиваются значения переменным типа булево? Делается это очень просто.

Первый способ:

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)

    А = Ложь;
    Б = Истина;

    Сообщить(А);
    Сообщить(Б);

КонецПроцедуры
```

Листинг 2.3.1

Посмотрите, как выводится все это в окно сообщений.

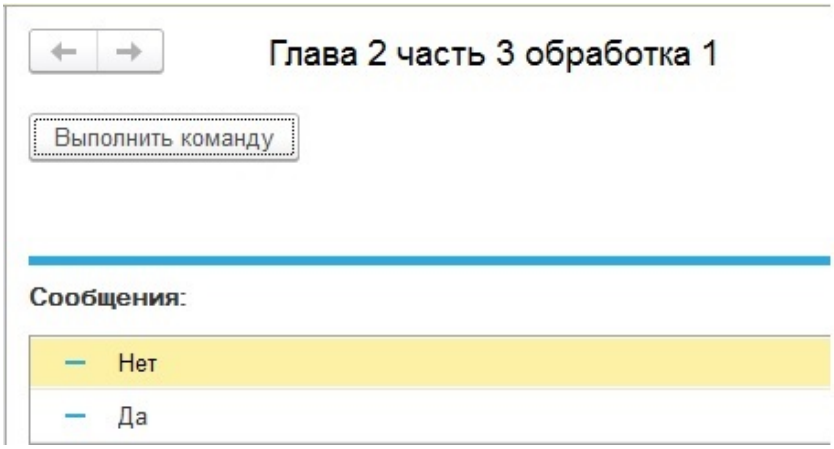


Рис. 2.3.1

Видите, что разработчики сделали вывод булевых значений на экран в наиболее комфортном пользователю виде.

Второй способ: они могут представляться в виде какого-либо выражения, например:

```
&НаКлиенте
Процедура ВыполнитьКоманду2(Команда)

    А = 5 > 7;
    Б = 5 < 7;

    Сообщить(А);
    Сообщить(Б);

КонецПроцедуры
```

Листинг 2.3.2

Смотрим, что выйдет в окно сообщений.

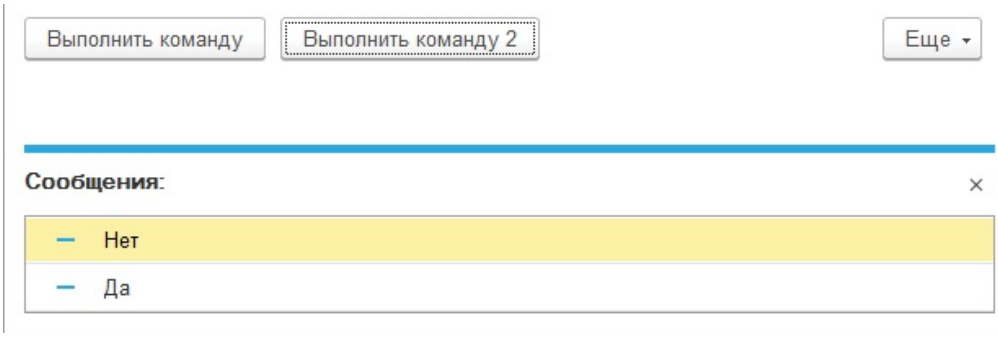


Рис. 2.3.2

Ответьте, почему вышли такие сообщения?

Операции над переменными типа Булево

Всего в языке программирования 1С существует три операции над переменными типа булево: это ***НЕ***, ***И***, ***ИЛИ***.

Для того чтобы их понять, рассмотрим три таблицы:

Первая операция – ***НЕ,*** Логическое отрицание. Данная операция меняет булево значение на противоположное.

```
А = Не Б;
```

Понять функционирование данной операции Вам поможет следующая таблица:

А	**Б**
Истина	Ложь
Ложь	Истина

Табл. 2.3.1

Следующая операция – ***И***, конъюнкция. Данная операция возвращает истину только тогда, когда оба значения истинны.

```
С = А И Б
```

Понять функционирование данной операции Вам поможет следующая таблица:

С = А И Б	**А**	**Б**
Истина	Истина	Истина
Ложь	Ложь	Истина
Ложь	Истина	Ложь
Ложь	Ложь	Ложь

Табл. 2.3.2

Третья операция – ***ИЛИ***, дизъюнкция. Операция ***ИЛИ*** возвращает истину тогда, когда хотя бы одно значение истинно.

```
С = А ИЛИ Б
```

Понять функционирование данной операции Вам поможет следующая таблица:

С = А ИЛИ Б	**А**	**Б**
Истина	Истина	Истина
Истина	Ложь	Истина
Истина	Истина	Ложь
Ложь	Ложь	Ложь

Табл. 2.3.3

Если у Вас возникли проблемы с осознанием данных операций, перепишите эти таблицы куда-нибудь и используйте их всякий раз, когда будете работать с данным типом.

Впоследствии Вы освоите эти логические операции и легко будете ими манипулировать.

Для наглядности я приведу пару примеров из таблиц на языке программирования 1С (для этого я создал новую обработку):

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)
    А = Ложь;
    Б = Истина;

    С = Не А;
    Сообщить(С);

    С = А И Б;
    Сообщить(С);

    С = А Или Б;
    Сообщить(С);
КонецПроцедуры
```

Листинг 2.3.3

Смотрим, что получилось.

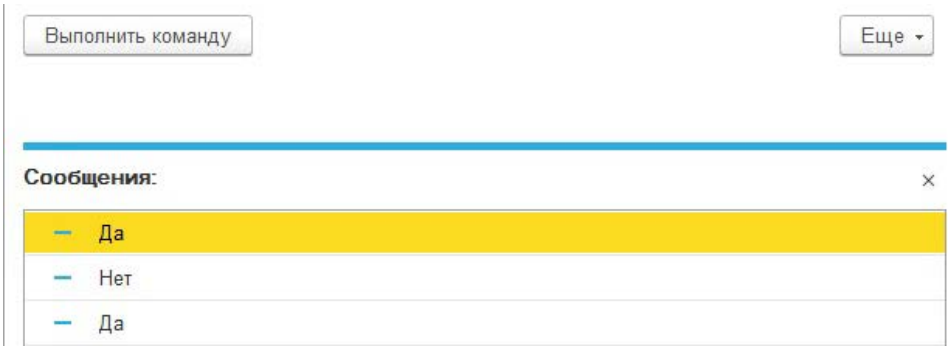


Рис. 2.3.3

Теперь самостоятельно выполните все операции из таблиц и проверьте, правда ли там написана.

А можно ли выполнять данные операции подряд?

Да, можно, данные операции будут выполняться слева направо. И иметь следующий уровень старшинства:

Первый: в первую очередь выполняются операции в скобках

Второй: Операция НЕ

Третий: Операция И

Четвертый: Операция ИЛИ.

Чтобы не путаться в операциях, я советую Вам использовать скобки где только возможно.

Рассмотрим пример.

```
&НаКлиенте
Процедура ВыполнитьКоманду2(Команда)

    А = Ложь;
    Б = Истина;
```

```
    С = Ложь;

    Д = А и С или Б;

    Сообщить(Д);

КонецПроцедуры
```

Листинг 2.3.4

В данном случае сначала будет работать операция *И* между А и С.

Смотрим таблицу А – *Ложь*, С – *Ложь*, результат А *И* С будет *Ложь*.

Следующим шагом будет выполнение операции *ИЛИ* между *Ложью* (Результат предыдущей операции) и значением Б, которое *Истина*.

Результат будет *Истина*.

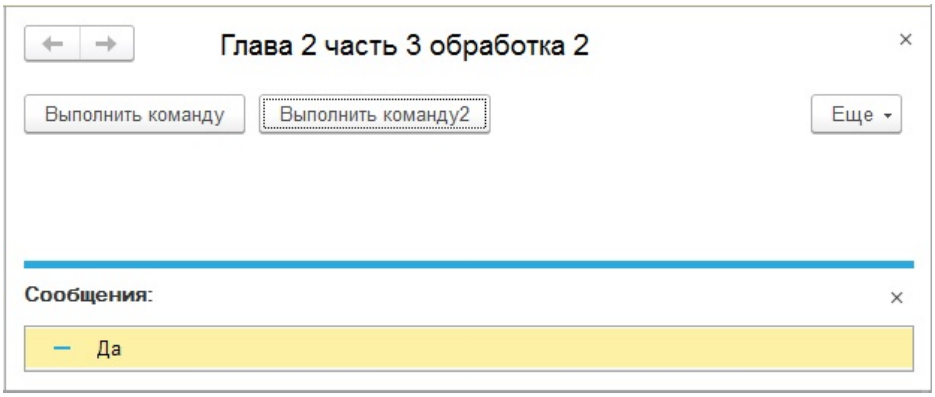


Рис. 2.3.4.

А если нам надо, чтобы прошла сначала операция *ИЛИ* между С и Б, а потом только операция *И* между А и тем, что получилось, то для этого необходимо использовать скобки.

Смотрим:

```
&НаКлиенте
Процедура ВыполнитьКоманду2(Команда)

    А = Ложь;
    Б = Истина;

    С = Ложь;

    Д = А и (С или Б);

    Сообщить(Д);

КонецПроцедуры
```

Листинг 2.3.5

Результат диаметрально противоположный. Почему?

Сейчас разберем. Благодаря скобкам сначала выполняется операция и между С и Б,

т.к. С - *Ложь*, а Б – *Истина*, результат будет *Истина* (см табл. 1.3.3). Потом между значением А (которое *Ложь*) и значением *Истина* (результатом предыдущей операции) выполняется операция *И*. Результат будет *Ложь* (см табл. 1.3.2).

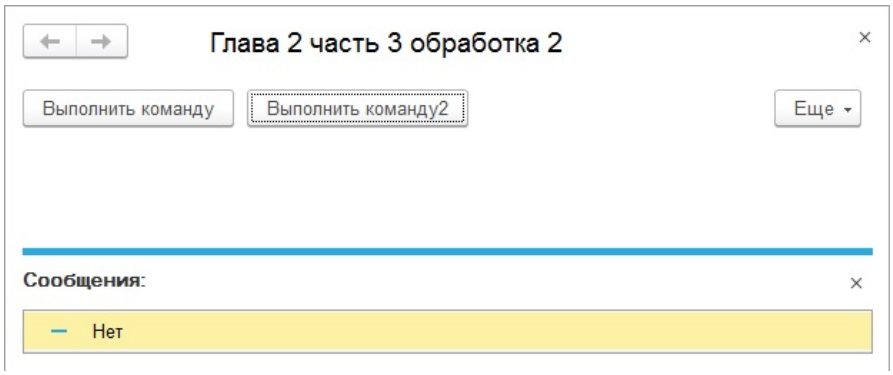


Рис. 2.3.5

Рассмотрим еще один пример со скобками и операцией *НЕ*

```
&НаКлиенте
Процедура ВыполнитьКоманду3(Команда)
    А = Ложь;
    Б = Истина;

    Д = Не А Или Б;
    Сообщить(Д);
    Д = Не (А Или Б);
    Сообщить(Д)
КонецПроцедуры
```

Листинг 2.3.6

Посмотрите, какой будет результат, и попробуйте сами его объяснить, используя вышеприведенные таблицы (табл. 1.3.1 – 1.3.3).

Операторы сравнения

Теперь мы рассмотрим более подробно операции сравнения. Это:

Больше: >

Меньше: <

Больше или равно: >=

Меньше или равно: <=

Не равно: <>

Равно: =

Все эти операции применимы для всех примитивных типов.

Рассмотрим их для чисел.

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)
    Д = 5 > 5;
    Сообщить("5 > 5 это - ");
    Сообщить(Д);

    Д = 5 >= 7;
    Сообщить("5 >= 7 это - ");
    Сообщить(Д);

    Д = 6 < 8;
    Сообщить("6 < 8 это - ");
    Сообщить(Д);

    Д = 7 <= 9;
    Сообщить("7 <= 9 это - ");
    Сообщить(Д);

    Д = 7 <> 10;
    Сообщить("7 <> 10 это - ");
    Сообщить(Д);

    Д = 7 = 7;
    Сообщить("7 = 7 это - ");
    Сообщить(Д);
КонецПроцедуры
```

Листинг 2.3.7

Смотрим, что получилось.

Рис. 2.3.6

Точно так же с данными операциями сравнения можно применять операции *НЕ*, *ИЛИ*, Скомпонуем из вышеприведенного примера следующий:

```
&НаКлиенте
Процедура ВыполнитьКоманду2(Команда)

    Д = (5 > 5) И (5 >= 7);
    Сообщить(Д);
    Д = (6 < 8) Или (7 <= 9);
    Сообщить(Д);
    Д = Не (7 <> 10) Или (7 = 7);
    Сообщить(Д);

КонецПроцедуры
```

Листинг 2.3.8

Смотрим, что получилось. Объясните данный результат.

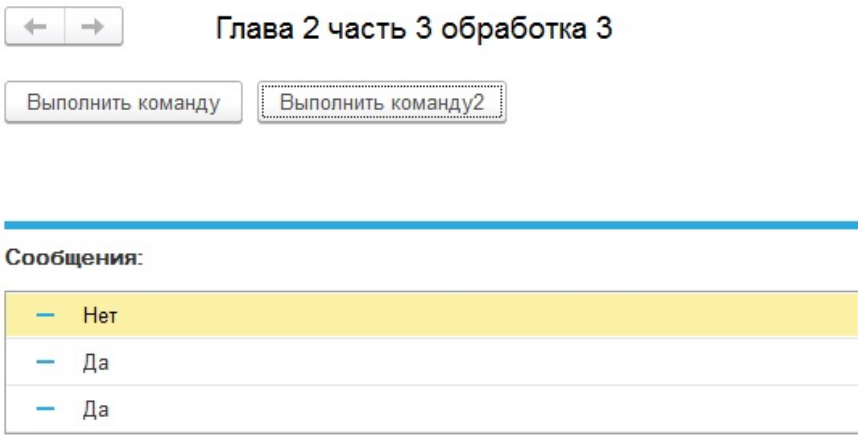


Рис. 2.3.7

Часть 4. Переменные типа Число

В этой части главы мы научимся работать с примитивным типом *Число* и освоим все необходимые для программирования операции над числами. Все остальное Вы без труда сможете найти в справке.

Создайте новую обработку, в которой Вы будете проводить все манипуляции с переменными, как Вы делали в предыдущих частях (форму обработки, команду и обработчик команды).

Зададим простые переменные:

```
&НаКлиенте
Процедура ВыполнениеКоманды(Команда)

    Перем А, Б, С;
    Перем Д, Г, Е;

КонецПроцедуры
```

Листинг 2.4.1

Теперь присвоим им числа и выполним элементарные операции.

```
    А = 10; Б = 12; С = 3;

    Д = А + Б - С;
    Г = А * Б;
    Е = А/Б;
```

Листинг 2.4.2

И выведем результат в окно сообщений.

```
    Сообщить(Д);
    Сообщить(Г);
    Сообщить(Е);
```

Листинг 2.4.3

Сохраняем обработку и смотрим на результат.

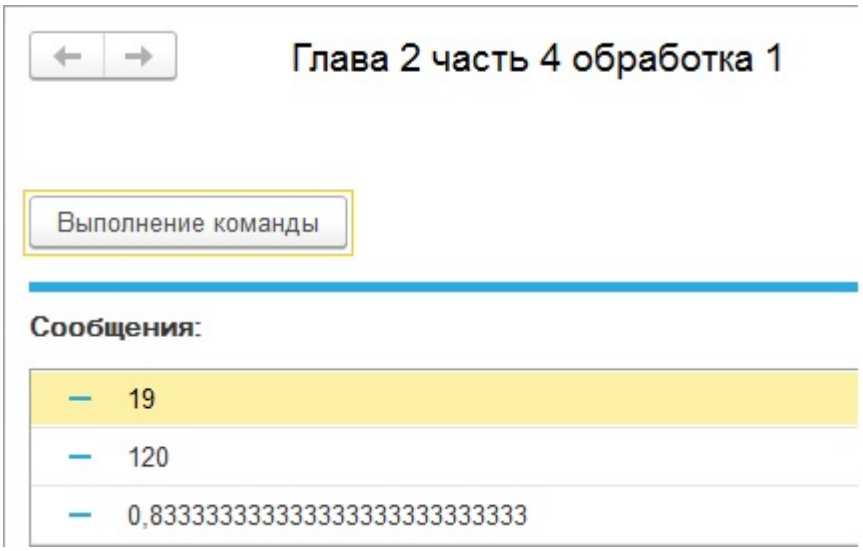


Рис. 2.4.1

Теперь о порядке выполнения операций. Сначала выполняются операции умножения и деления, потом операции сложения и вычитания. Операции умножения и деления выполняются слева направо.

К примеру, в операции

```
E = A/Б/C;
```

сначала будет А делиться на Б, потом все это поделится на С.

А при операции

```
E = A - Б/C;
```

сначала Б поделим на С, а потом это все вычтем из А.

Что же нам сделать, если в выражении

```
E = A/Б/C;
```

сначала надо поделить Б на С, а потом все это поделить на А? Или в другом выражении сначала вычесть А из Б, а потом поделить на С? Для этого нам пригодятся операторные скобки.

Запишем так:

```
E = A/(Б/C);
```

или:

```
Е = (А - Б)/С;
```

Проверим данные свойства на практике:

```
&НаКлиенте
Процедура ВыполнениеКоманды2(Команда)
    Перем А, Б, С;
    Перем Д, Г, Е;
    А = 23;
    Б = 7;
    С = 4;

    Е = А/Б/С;
    Д = А - Б/С;
    Сообщить("А/Б/С = ");
    Сообщить(Е);
    Сообщить("А - Б/С = ");
    Сообщить(Д);

    Е = А/(Б/С);
    Д = (А - Б)/С;
    Сообщить("А/(Б/С) = ");
    Сообщить(Е);
    Сообщить("(А - Б)/С = ");
    Сообщить(Д);
КонецПроцедуры
```

Листинг 2.4.4

Посмотрим результат:

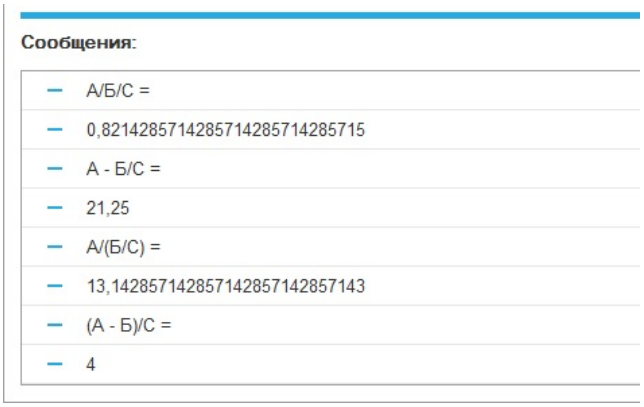


Рис. 2.4.2

Мой Вам совет, можете вдоволь злоупотреблять этими скобками. Если Вы сомневаетесь, какая операция будет выполняться в первую очередь, заключайте в скобку ту операцию, которая Вам нужна.

Продолжаем работу с числами. Следующий оператор, который нас интересует, это оператор %.

Например:

```
Е = А % Б;
```

в данном случае берется остаток от деления А на Б.

Например: 10 % 7 = 3.

23 % 7 = 2.

При помощи этого оператора Вы можете проверить кратно одно число другому или нет: если одно число кратно другому, то остаток от деления будет 0.

Узнаем при помощи этого оператора, кратно ли число 22 двум и четырем (см. листинг 2.4.5).

```
&НаКлиенте
Процедура ВыполнениеКоманды3(Команда)

    Перем А, Б, С;
    Перем Д, Г, Е;

    А = 22;
    Б = 2;
    С = 4;

    Д = А % Б;
    Г = А % С;

    Сообщить("А % Б = ");
    Сообщить(Д);

    Сообщить("А % С = ");
    Сообщить(Г);

КонецПроцедуры
```

Листинг 2.4.5

Рис. 2.4.3

Начнем изучать функции языка программирования 1С для чисел.

Функция Окр

И первая функция, с которой мы будем иметь дело, это функция ***Окр***. Данная функция округляет число до нужного количества знаков после запятой. Если мы напишем:

```
Окр(12.122123, 3)
```

то результат будет 12,122.

А при

```
Окр(12.122123, 1)
```

результат будет 12.1,

А при

```
Окр(12.122123, 0)
```

результат будет 12.

У данной функции два параметра. Первый – число, которое нужно округлить. Второй – сколько знаков после запятой нужно оставить.

Посмотрим, как это работает (я это сделал в новой обработке).

```
&НаКлиенте
Процедура ВыполнениеКоманды(Команда)
    В = Окр(12.122123,3);
    Сообщить(В);
    В = Окр(12.124623,1);
    Сообщить(В);
    В = Окр(12.124623,0);
    Сообщить(В);
КонецПроцедуры
```

Листинг 2.4.6

Теперь смотрим, что у нас получилось. Запустите обработку в «1С:Предприятии».

Рис. 2.4.4

Все очень просто. В функцию *Окр* можно передавать еще третье значение, это системное перечисление *РежимОкругления.* Оно бывает двух видов:

РежимОкругления.Окр15как10 и

РежимОкругления.Окр15как20

В первом случае число 1,5 округляется до 1 в меньшую сторону, во втором случае число 1,5 округляется до 2 в большую сторону.

Рассмотрим пример:

```
&НаКлиенте
Процедура ВыполнениеКоманды2(Команда)

    В = Окр(12.1345,3, РежимОкругления.Окр15как10);
    Сообщить(В);
    В = Окр(12.1345,3, РежимОкругления.Окр15как20);
    Сообщить(В);

КонецПроцедуры
```

Листинг 2.4.7

Теперь посмотрим, что получилось. Запустите обработку в «1С:Предприятии».

Рис. 2.4.5

Как Вы уже поняли, режим округления необязательный параметр, т.е. можно его опустить, если Вам это не критично.

Функция Цел

Следующая функция - это функция ***Цел***. Данная функция отсекает дробную часть от целого числа. Напишите следующий код:

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)
    Перем А,Б,С;

    А = 23;
    Б = 7;
    С = 4;

    В = Цел(442.11344554);
    Сообщить(В);
```

```
    В = Цел(0.12341);
    Сообщить(В);
    В = Цел ((А*С)/Б);
    Сообщить(В);
КонецПроцедуры
```

Листинг 2.4.8

Теперь посмотрите, что получилось.

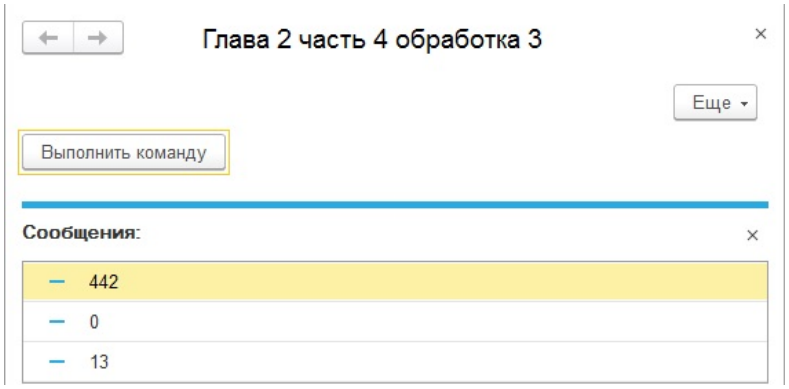


Рис. 2.4.6

Функция Exp

Следующая нужная нам функция - это возведение в степень, функция ***Exp***. Она имеет следующий синтаксис:

Exp(x,y)

Это $\mathbf{x}^{\mathbf{y}}$. Числа **x** и **y** могут принимать любые значения, в том числе дробные и отрицательные. Единственно, если **y** меньше **0**, то число **x** всегда должно быть положительным, иначе сгенерируется ошибка. Рассмотрим примеры:

```
&НаКлиенте
Процедура ВыполнитьКоманду2(Команда)

    В = Pow(100,0.2);
    Сообщить(В);
    В = Pow(100,5);
    Сообщить(В);
    В = Pow(100,-2);
    Сообщить(В);
    В = Pow(100,0);
    Сообщить(В);
    В = Pow(0,100);
    Сообщить(В);

КонецПроцедуры
```

Листинг 2.4.9

Теперь посмотрите, что получилось.

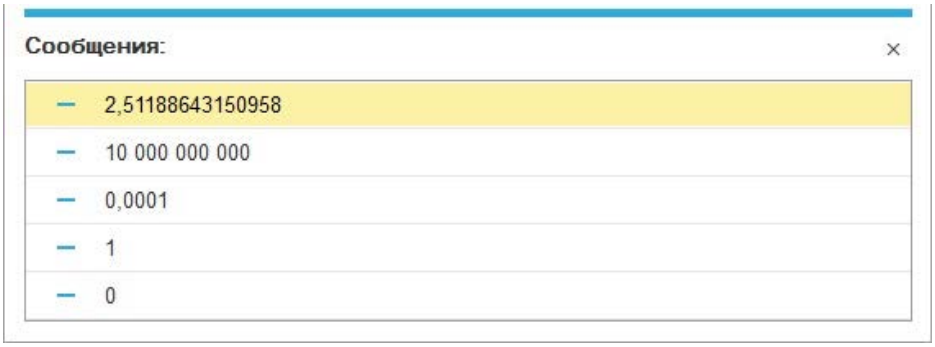


Рис. 2.4.7

Функция SQRT

Еще одна интересная для нас функция - это квадратный корень ***SQRT***. Пишется она в виде

Sqrt(X),

где X – не отрицательное число.

Несколько примеров:

```
&НаКлиенте
Процедура ВыполнитьКоманду3(Команда)

    В = Sqrt(100);
    Сообщить(В);
    В = Sqrt(9);
    Сообщить(В);
    В = Sqrt(7);
    Сообщить(В);

КонецПроцедуры
```

Листинг 2.4.10

Теперь посмотрите, что получилось.

Рис. 2.4.8

Диалог для ввода чисел

Возможно, Вы заметили, что все значения переменных мы вводили в коде программы. Так не всегда практично, иногда нужно, чтобы пользователь сам ввел то или иное число. Чаще всего это выполняется с помощью элементов форм. Пока мы эту возможность не затронем, поскольку элементы формы будем изучать в пятой главе.

В этом уроке мы научимся вводить числа интерактивно, с помощью специального окна ввода. Выполняется это функцией ***ВвестиЧисло***. Она возвращает *Истина*, если число введено, и *Ложь,* если нет. Как пользоваться данной возможностью этой функции, мы узнаем в следующей главе, в части, посвященной условиям. Пока же будем использовать данную функцию для интерактивного ввода чисел.

Так же, как и процедура «Предупреждение», этот метод работает только в режиме включенной модальности!

Разберем данную функцию:

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)

    Перем А;

    ВвестиЧисло(А, "Введите число А",5,2);

КонецПроцедуры
```

Листинг 2.4.11

Первым параметром задается переменная, в которую будет записано число, нами введенное.

Вторым параметром задается заголовок окна, которое выйдет пользователю.

Третий параметр - длина числа

Четвертый параметр - точность.

Обратите Ваше внимание, что длина числа - это общая длина вместе с цифрами перед запятой и после запятой. В нашем случае это будет число типа 200,11 или 99,11, но никак не 1201,12.

Обязательным является только первый параметр. Потому функция вполне может быть и в таком виде:

```
    ВвестиЧисло(А);
```

Листинг 2.4.12

В «1С:Предприятии» данный диалог будет выглядеть следующим образом:

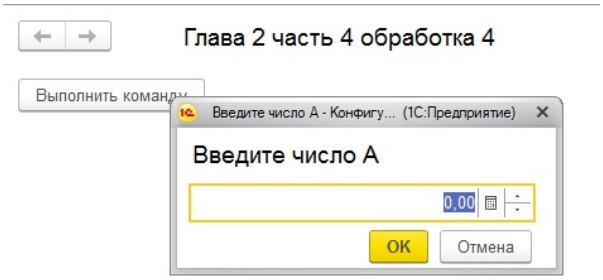


Рис. 2.4.9

Теперь перепишите все примеры, которые мы изучили в четвертой части этой главы, с возможностью ввода чисел.

Итак, мы рассмотрели основные функции, которые необходимы для повседневной работы в программе 1С. Есть и еще функции типа вычисления синуса или косинуса, но они очень редко применяемы. Кому интересно, может посмотреть соответствующий раздел справки.

Если Вы задались вопросом про то, каким образом пользователю вводить числа в программу 1С, то не пугайтесь. Всему свое время. Это мы будем изучать в пятой главе.

Резюме

В этой части Вы научились работать с числами. Функций работы с числами гораздо больше, чем тех, что мы перечислили в этой главе. С ними Вы сможете ознакомиться в справочной информации. Потренируйтесь самостоятельно с оставшимися функциями.

Часть 5. Переменные типа Строка

Теперь мы будем учиться работать со строками.

Создайте новую обработку, в которой Вы будете проводить все манипуляции с переменными типа *Строка*, как Вы делали в предыдущих частях.

Зададим три переменных типа строка (напомню, они задаются посредством написания текста в кавычках).

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)

    ПеремФамилия = "Иванов";
    ПеремИмя     = "Петр";
    ПеремОтчество= "Васильевич";

КонецПроцедуры
```

Листинг 2.5.1

Первая операция, которую мы разберем, это сложение строк, или операция конкатенации:

```
ПеремФИОПолностью = ПеремФамилия + " " + ПеремИмя + " " + ПеремОтчество;
```

Листинг 2.5.2

Добавляем процедуру *Сообщить*, чтобы посмотреть что получилось.

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)
    ПеремФамилия = "Иванов";
    ПеремИмя     = "Петр";
    ПеремОтчество= "Васильевич";

    ПеремФИОПолностью = ПеремФамилия + " " + ПеремИмя + " " + ПеремОтчество;

    Сообщить(ПеремФИОПолностью);
КонецПроцедуры
```

Листинг 2.5.3

Сохраните обработку, запустите и посмотрите, что вышло в окно сообщений.

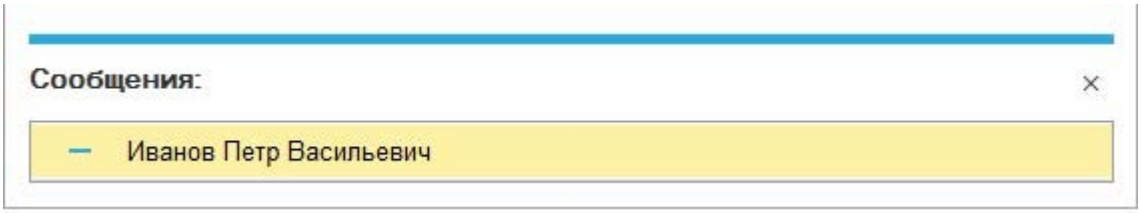


Рис. 2.5.1

Как Вы заметили из данного примера, строки можно складывать как посредством переменных типа строка, так и посредством написания самой строки.

Т.е. можно написать и так:

```
ПеремФИОПолностью = "Фамилия: " + ПеремФамилия +
                    ", имя: " + ПеремИмя +
                    ", отчество " + ПеремОтчество;
```

Листинг 2.5.4

Подправьте предыдущий код, и в окно сообщений должна выйти следующая надпись:

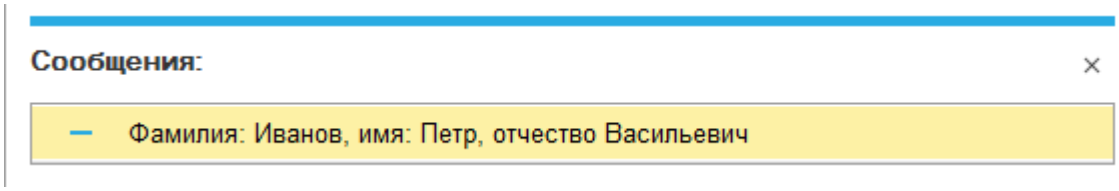


Рис. 2.5.2

Теперь рассмотрим очень интересный и нужный для нас набор системных значений - это ***Символы***. Всего данный набор содержит 6 значений, и мы поочередно рассмотрим некоторые из них на примере Иванова Петра Васильевича. Добавляйте код из листингов в Вашу обработку и смотрите, что будет выходить в окно сообщений.

Первое значение - это *Возврат каретки*, т.е. текст переносится на новую строку в начало.

```
ПеремФИОПолностью = ПеремФамилия + Символы.ВК +
                    ПеремИмя +  Символы.ВК +
                    ПеремОтчество;
```

Листинг 2.5.5

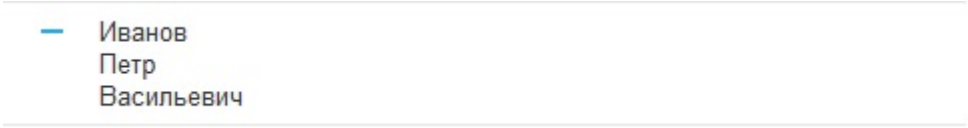


Рис. 2.5.3

Следующее значение - это *Горизонтальная табуляция*, имитирует нажатие клавиши «*Tab*» на клавиатуре при написании текста.

```
ПеремФИОПолностью = ПеремФамилия + Символы.Таб +
                    ПеремИмя +  Символы.Таб +
                    ПеремОтчество;
```

Листинг 2.5.6

Рис. 2.5.4

Продолжаем, теперь символ пробела, называется *Символ непрерывного пробела*.

```
ПеремФИОПолностью = ПеремФамилия + Символы.НПП +
                    ПеремИмя +  Символы.НПП +
                    ПеремОтчество;
```

Листинг 2.5.7

Рис. 2.5.5

Следующий символ, который нас интересует, это *Перенос строки:*

```
ПеремФИОПолностью = ПеремФамилия + Символы.ПС +
                    ПеремИмя +  Символы.ПС +
                    ПеремОтчество;
```

Листинг 2.5.8

Рис. 2.5.6

Еще перенос строки можно сделать, используя символ «|» (вертикальная черта), тогда надо на строке оставлять кавычку открытой, а на следующей строке, не открывая кавычку, сразу ставим вертикальную черту. Например.

```
СтрокаСПереносом = "Тут мы будем переносить строку
                   |вот мы на новой строке";

Сообщить(СтрокаСПереносом);
```

Листинг 2.5.9

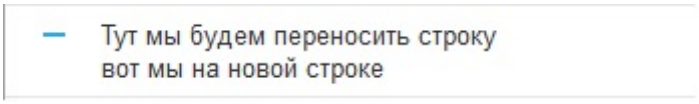


Рис. 2.5.7

Символ *Возврат каретки* и перенос строки абсолютно идентичен.

Символы *Перевод формы* и *Вертикальная табуляция* в данном уроке мы рассматривать не будем. А пока поэкспериментируйте с изученным набором символов, попробуйте использовать разные комбинации символов.

У многих новичков возникает вопрос, как задать кавычки непосредственно в тексте. Сделать это очень просто. Смотрим пример:

```
ФИО = "Иванов Иван Иванович по кличке ""Иваныч""";
Сообщить(ФИО);
```

Листинг 2.5.10

Рис. 2.5.8

Все очень просто: пишем сдвоенные кавычки и внутри нужное слово.

Диалог для ввода строк

Разберем, каким образом пользователю вводить интерактивно (без участия формы) строки.

Для этого мы будем использовать функцию *ВвестиСтроку*. Она возвращает *Истину*, если строка введена, и *Ложь,* если нет. В последующих главах мы узнаем, как использовать данное свойство функции.

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)

    Перем ПеремФамилия;

    ВвестиСтроку(ПеремФамилия,"Введите фамилию",100,Ложь);

    Сообщить(ПеремФамилия);

КонецПроцедуры
```

Листинг 2.5.11

Так же, как и процедура «Предупреждение», этот метод работает только в режиме включенной модальности!

Напишите этот код в обработке, сохраните ее и запустите «1С:Предприятие». При нажатии кнопки *«Выполнить»* должно выйти следующее окно:

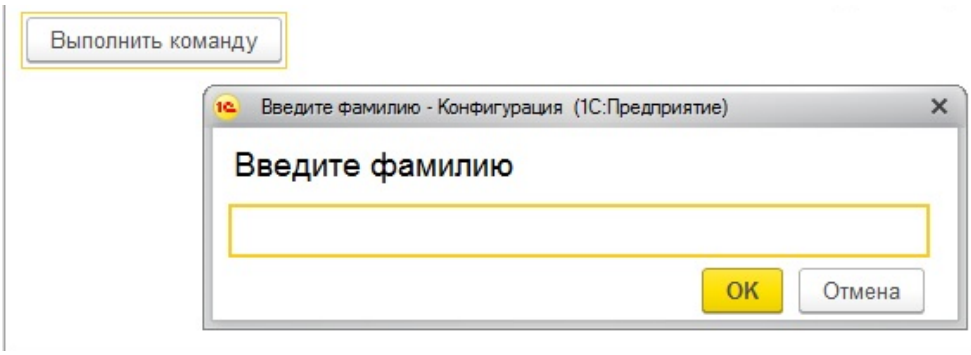


Рис. 2.5.9

Введите в нее любую фамилию, имя, отчество, нажмите кнопку *«ОК»* и посмотрите, что получится.

Разберем теперь данную функцию. Она имеет следующий синтаксис:

ВвестиСтроку(<Строка>, <Подсказка>, <Длина>, <Многострочность>)

Где:

«Строка» - это переменная, в которую мы запишем введенное пользователем значение;

«Подсказка» - это заголовок окна;

«Длина» - это длина строки, т.е. максимальное количество символов, которое поместится в окно;

«Многострочность» - это признак многострочности. Если он указан *Истина*, то можно вводить несколько строк, это, по сути, текст. Измените параметр на *Истина*. В результате выйдет следующее окно:

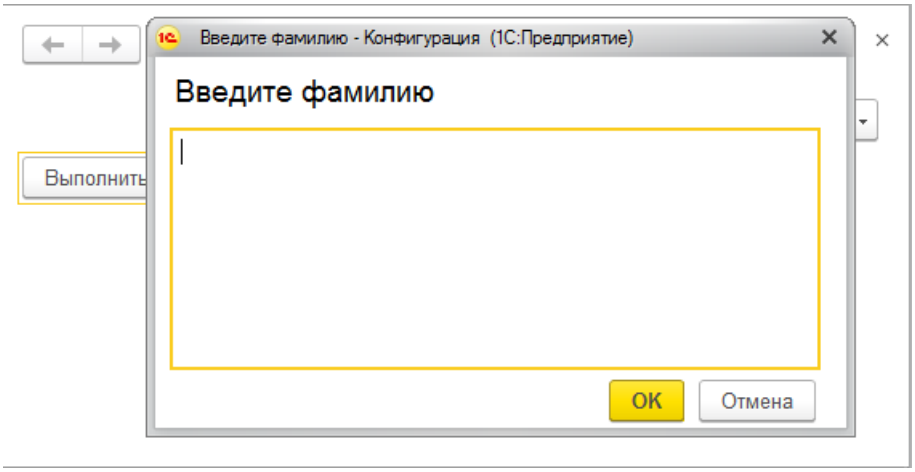


Рис. 2.5.10

Обязательным является только первый параметр. Т.е. данную функцию можно написать и так:

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)

    Перем ПеремФамилия;

    ВвестиСтроку(ПеремФамилия);

    Сообщить(ПеремФамилия);

КонецПроцедуры
```

Листинг 2.5.12

Посмотрите, что получится в этом случае.

Теперь Вам небольшое задание: переделайте код, показанный выше с фамилией, именем, отчеством, так, чтобы данные значения вводились пользователем самостоятельно.

Функция СтрДлина

Так, с самими строками разобрались, с интерактивным вводом строк понятно - теперь перейдем к изучению функций и процедур для работы со строками.

Первая наша функция будет ***СтрДлина***. Данная функция возвращает длину строки, т.е. количество символов в строке, включая пробелы, возвраты кареток, переносы, табуляции и т.д.

Напишите в обработке следующий код (обработку лучше создать новую):

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)

    Перем ПеремФИОПолностью;

    ВвестиСтроку(ПеремФИОПолностью, "Введите ФИО",200,Истина);

    ПеремДлина = СтрДлина(ПеремФИОПолностью);

    Сообщить("СтрДлина (" + ПеремФИОПолностью + ") = " + ПеремДлина);

КонецПроцедуры
```

Листинг 2.5.13

Теперь введем любое ФИО и смотрим результат:

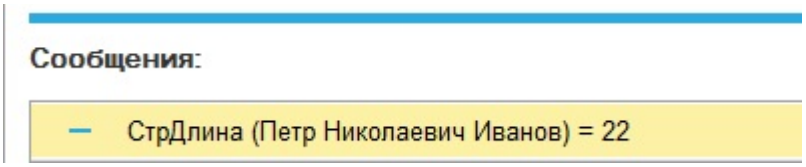


Рис. 2.5.11

Как видим, длина строки *«Петр Николаевич Иванов»* - 22 символа.

Функция простая, и каких-то особых комментариев не требуется.

Функции СокрЛ, СокрП, СокрЛП

Часто бывают случаи, когда строка может задаться пользователем либо перенестись из другой программы в виде

«Иванов Иван Иванович »

И много пробелов в конце строки.

Для того чтобы отсечь все эти не нужные пробелы, есть специальные функции:

СокрЛ(<Строка>) – отсекает пробелы слева.

СокрП(<Строка>) – отсекает пробелы справа.

СокрЛП(<Строка>) – отсекает пробелы справа и слева

Чтобы продемонстрировать возможность этих функций, напишите в обработке такой код:

```
Процедура ВыполнитьКоманду2(Команда)

    Перем НачСтроки, КонСтроки, ФИО;

    НачСтроки = "НачСтр/";
    КонСтроки = "/КонСтр";

    ВвестиСтроку(ФИО, "Введите ФИО");

    Сообщить(НачСтроки + ФИО + КонСтроки);

КонецПроцедуры
```

Листинг 2.5.14

После запуска обработки и выхода окна для ввода строки, специально добавьте пробелы перед самой строкой и после.

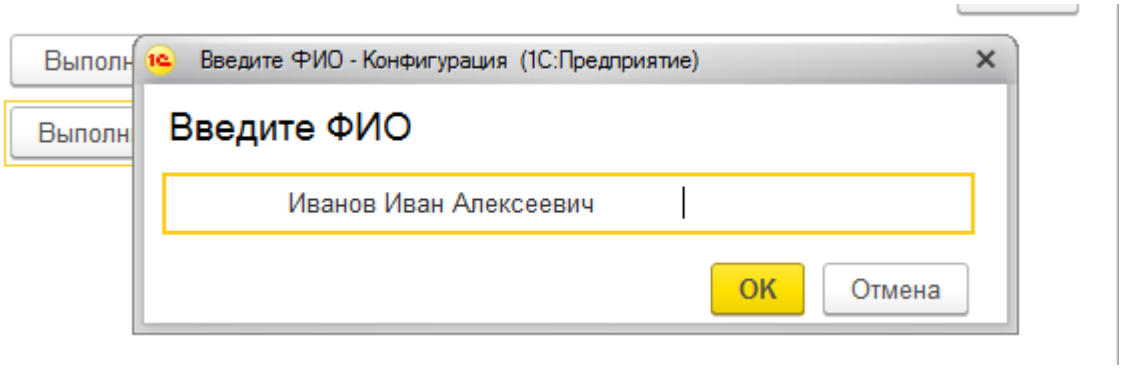


Рис. 2.5.12

Результат должен получиться следующим:

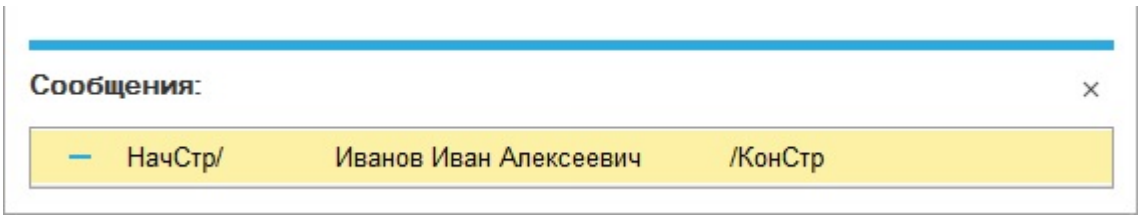


Рис. 2.5.13

Данные пробелы нам мешают. Их надо убрать, для этого используем вышеприведенные функции.

Чтобы показать это, допишите следующий код:

```
    ФИОСокрЛ  = СокрЛ(ФИО);
    ФИОСокрП  = СокрП(ФИО);
    ФИОСокрЛП = СокрЛП(ФИО);

    Сообщить("СокрЛ: " + НачСтроки + ФИОСокрЛ + КонСтроки);
    Сообщить("СокрП: " + НачСтроки + ФИОСокрП + КонСтроки);
    Сообщить("СокрЛП: " + НачСтроки + ФИОСокрЛП + КонСтроки);
```

Листинг 2.5.15

Посмотрите, как должно получиться:

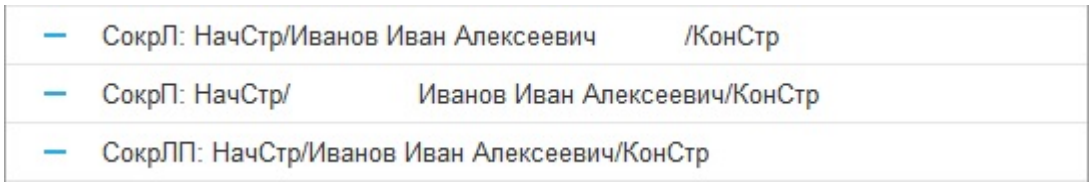


Рис. 2.5.14

Мой Вам совет: как можно больше злоупотреблять данными функциями, поскольку часто многие пользователи по ошибке или по иным причинам ставят пробелы после уже введенного слова. А потом при дальнейшей работе это выглядит очень некрасиво.

Используйте данные функции в том коде, где Вы вводите ФИО отдельно, попробуйте ставить лишние пробелы, и посмотрите, что из этого у Вас получится.

Функции Лев, Прав

Рассмотрим функции ***Лев*** и ***Прав.***

Данные функции выбирают нужное количество символов слева и справа строки соответственно.

Рассмотрим функцию ***Лев***. Данная функция выбирает слева нужное количество символов. Она пишется в таком виде:

Лев(<Строка>,<КоличествоСимволов>)

Где:

«Строка» – та строка, которую функция обрабатывает;

«КоличествоСимволов» – количество символов, которые будут выбраны.

Разберем это на уже знакомом примере с фамилиями. Внесите в Вашу обработку следующий код:

```
Процедура ВыполнитьКоманду3(Команда)

    ПеремФамилия  = "Иванов";
    ПеремИмя      = "Петр";
```

```
    ПеремОтчество = "Сергеевич";

    ФИОКратко = ПеремФамилия + " " +
                Лев(ПеремИмя,1) + ". " +
                Лев(ПеремОтчество,1) + ".";

    Сообщить(ФИОКратко);
КонецПроцедуры
```

Рис. 2.5.15

Выйти должна следующая строка:

Рис. 2.5.16

Рассмотрим другой пример.

Имеем, к примеру, автомобильные номера вида «ва345укRU23». По условиям задачи код региона всегда будет состоять из двух символов. Надо получить номер без кода региона.

```
Номер = "ва345укRU23";
НомерБезКода = Лев(Номер,7);
Сообщить(НомерБезКода);
```

Листинг 2.5.16

Смотрим.

Данный код должен выводить следующий результат:

Рис. 2.5.17

Теперь перейдем к функции ***Прав***. Эта функция выбирает справа нужное количество символов. Она пишется в таком виде:

Прав(<Строка>,<Количество символов>)

Где:

«Строка» – та строка, которую функция обрабатывает.

«Количество символов» – количество символов, которые будут выбраны.

Рассмотрим ее на нашем примере с номером, только на этот раз получим код региона.

```
Номер = "ва345укRU23";
КодРегиона = Прав(Номер,4);
Сообщить(КодРегиона);
```

Листинг 2.5.17

Данный код должен выводить следующий результат:

Рис. 2.5.18

А теперь, используя функцию *Лев* и *Прав*, получите три цифры номера.

Рассмотрим другой пример, который использует также функцию *СтрДлина*. Иногда некоторые поставщики товар выгружают с артикулом перед наименованием, выглядит это, к примеру, так:

«34221 Ручка шариковая» или

«23444 Тетрадь в клетку, 12 листов»

Мы знаем, что артикул всегда состоит из 5 символов, нам надо их отсечь. Как это сделать?

Для этого нам надо отобрать справа количество символов, равное длине названия без артикула. Сделать это необходимо так: узнать длину строки и вычесть из нее 5 (длина артикула априори всегда такая).

```
Товар1 = "34221 Ручка шариковая";
Товар2 = "23444 Тетрадь в клетку, 12 листов";

НаимТовар1 = СокрЛП(Прав(Товар1,СтрДлина(Товар1) - 5));
НаимТовар2 = СокрЛП(Прав(Товар2,СтрДлина(Товар2) - 5));

Сообщить(НаимТовар1);
Сообщить(НаимТовар2);
```

Листинг 2.5.18

На выходе должны получиться следующие строки:

Рис. 2.5.19

Пытливый читатель спросит, почему я вычел 5, а не 6 вместе с пробелом, а потом не применил функцию *СокрЛП*. Дело в том, что в условиях задачи сказано, что длина артикула 5, а не 6. То, что между артикулом и названием стоит пробел, это не более чем особенность выгрузки.

Может вполне быть и так:

Товар1 = "34221 Ручка шариковая";

Товар2 = "23444Тетрадь в клетку, 12 листов";

Что тогда получится, если Вы будете вычитать из длины 6?

А теперь потренируйтесь и получите отдельно артикул и отдельно общее наименование и выведите это в окно сообщений.

Функция Сред

В предыдущем примере я Вас попросил получить из номера три цифры, используя функции *Прав* и *Лев*. Сейчас я расскажу, как можно это сделать проще. Для этого мы будем использовать функцию ***Сред***. Данная функция выберет из строки нужное количество символов, начиная с определенного номера. Она имеет следующий синтаксис:

Сред(<Строка>,<Номер первого символа>, <Количество выбираемых символов>)

Разберем данную функцию подробно.

«Строка» – это наша строка.

«Номер первого символа» - это номер, с которого начинается наша выборка.

«Количество выбираемых символов» - это количество выбираемых символов.

Теперь измените пример с номером:

```
Номер = "ва345укRU23";
НомерЦифры = Сред(Номер,3,3);
Сообщить(НомерЦифры);
```

Листинг 2.5.19

И посмотрите, как работает данный код.

Разберем еще один пример со знакомыми нам уже артикулами:

```
Товар1 = "34221 Ручка шариковая";
Товар2 = "23444 Тетрадь в клетку, 12 листов";

Артикул1 = Сред(Товар1,1,5);
Артикул2 = Сред(Товар2,1,5);
```

```
Сообщить(Артикул1);
Сообщить(Артикул2);
```

Листинг 2.5.20

В этом примере также первый параметр в функциях *«Сред»* – наша строка, второй - это начало артикула (это первый символ), а третий – длина самого артикула.

Рис. 2.5.20

Теперь задание. Используя данную функцию, получите название товара. Подсказка: нужно применять функцию *СтрДлина*.

Функция Найти

Продолжаем изучение функций строк, и следующая функция - это ***Найти***. Данная функция находит вхождение искомой подстроки в строку поиска и возвращает номер первого символа искомой подстроки в строке поиска. Если подстрока не найдена, возвращается 0.

Функция имеет следующий синтаксис

Найти(<Исходная строка>,<Строка поиска>).

Где:

«Исходная строка» – строка, в которой будем искать подстроку;

«Строка поиска»– сама подстрока поиска.

Теперь запишите в Вашей обработке следующий код:

```
Номер = "ва345укRU23";
НомерНачRU = Найти(Номер,"RU");
Сообщить(НомерНачRU);
```

Листинг 2.5.21

Запустите обработку, и Вы увидите, что функция возвратила цифру 8. Это и есть номер символа, с которого начинается наша искомая строка.

Запомните эту функцию, она в Вашей работе может очень пригодиться. Хочу заметить еще, если подстрока поиска входит по нескольку раз в нашу строку, то возвращается самый первый номер вхождения.

Рассмотрим еще пример. Уже более практичный. Нам необходимо отделить имя от фамилии в строке.

Напишите следующий код в форме Вашей обработки:

```
ФамилияИмя1 = "Иванов Алексей";
ФамилияИмя2 = "Маликова Анна";

НомПроб1 = Найти(ФамилияИмя1," ");
НомПроб2 = Найти(ФамилияИмя2," ");

Имя1 = Сред(ФамилияИмя1,
            НомПроб1 + 1,
            СтрДлина(ФамилияИмя1) - НомПроб1 + 1);
Имя2 = Сред(ФамилияИмя2,
            НомПроб2 + 1,
            СтрДлина(ФамилияИмя2) - НомПроб2 + 1);

Сообщить(Имя1);
Сообщить(Имя2);
```

Листинг 2.5.22

Разберем данный код. Первое, мы вычислили номер первого пробела. Теперь нам необходимо, зная эту информацию, получить имя контрагента. Используем функцию *Сред*. Первый параметр – наша строка, второй параметр - номер пробела плюс один (поскольку имя начинается со следующего символа, то прибавим единицу), а третий параметр - это длина оставшегося имени. Ее мы получим, если вычтем из общей длины строки номер, с которого начинается пробел, и прибавим единицу (нам не нужна длина вместе с пробелом).

Рис. 2.5.21

Теперь используя данную функцию получите фамилию контрагента.

Функция СтрЗаменить

Еще одна функция, которая нам тоже много будет помогать, это функция ***СтрЗаменить***. Данная функция ищет определенную подстроку, и если она найдена, то меняет ее на другую подстроку. Она имеет следующий синтаксис:

СтрЗаменить(<Строка>,<Подстрока поиска>,<Подстрока замены>)

Где:

«Строка» - это та строка, с которой ведется работа;

«ПодСтрокаПоиска» – та строка, которую нужно найти;

«ПодСтрокаЗамены» – на что будет заменена *«ПодСтрокаПоиска»,* если будет найдена.

Сделайте следующий пример:

```
ФИО = "Иванов Петр Григорьевич";
ФИОСлит = СтрЗаменить(ФИО," ","");
ФИОСлит2 = СтрЗаменить(ФИО," ","_");

Сообщить(ФИОСлит);
Сообщить(ФИОСлит2);
```

Листинг 2.5.23

В заданной строке мы либо вообще уберем пробел, либо заменим его символом подчеркивания. Посмотрите, как он у Вас выполнится.

Рис. 2.5.22

Еще пара примеров на данную функцию:

```
МногоСтрок = "Первая строка
             |Вторая строка
             |Третья строка";

ОднаСтрока = СтрЗаменить(МногоСтрок,Символы.ПС,Символы.НПП);
Сообщить(ОднаСтрока);
```

Листинг 2.5.24

Теперь, наоборот, допишем к вышеприведенному коду:

```
МногоСтрок2 = СтрЗаменить(ОднаСтрока,Символы.НПП,Символы.ПС);
Сообщить(МногоСтрок2);
```

Листинг 2.5.25

Самостоятельно в ФИО, сделайте Фамилию Имя и Отчество на новой строке, и чтобы после каждого слова стояла точка.

Функция СтрЧислоВхождений

Немного похожая функция - это ***СтрЧислоВхождений***. Она вычисляет число вхождений строки поиска в заданной строке.

Имеет следующий синтаксис:

СтрЧислоВхождений(<Строка>,<Подстрока Поиска>).

«Строка» - это та строка, с которой работает данная функция.

«Подстрока поиска» - это строка, количество вхождений которой вычисляет данная функция.

Рассмотрим пример:

```
ФИО = "Иванов Петр Григорьевич";
ЧислоПробелов = СтрЧислоВхождений(ФИО," ");
Сообщить("Пробел встречается в строке " + ЧислоПробелов + " раз");
```

Листинг 2.5.26

Мы задали строку с пробелами и хотим выяснить, сколько пробелов в нашей строке. Для этого используем функцию *СтрЧислоВхождений*, в которой задаем строку, и искомые символы. В данном случае это пробел.

Сейчас мы рассмотрим функции для работы с многострочными строками.

Это функции *СтрЧислоСтрок* и *СтрПолучитьСтроку*.

Функция СтрЧислоСтрок

Начнем с функции ***СтрЧислоСтрок***.

СтрЧислоСтрок(<Строка>)

Она имеет только один параметр - это сама строка, а возвращает количество строк в заданной строке, если она многострочная. И 1, если это одна строка.

Сделайте пример с использованием данной функции:

```
МногоСтрок = "Первая строка
             |Вторая строка
             |Третья строка";
КолСтрок1 = СтрЧислоСтрок(МногоСтрок);

Сообщить(МногоСтрок);
Сообщить("В этой строке " + КолСтрок1 + " строки");

ОднаСтрока = СтрЗаменить(МногоСтрок,Символы.ПС,Символы.НПП);
КолСтрок2 = СтрЧислоСтрок(ОднаСтрока);
```

```
Сообщить(ОднаСтрока);
Сообщить("В этой строке " + КолСтрок2 + " строки");
```

Листинг 2.5.27

У Вас должен получиться следующий результат:

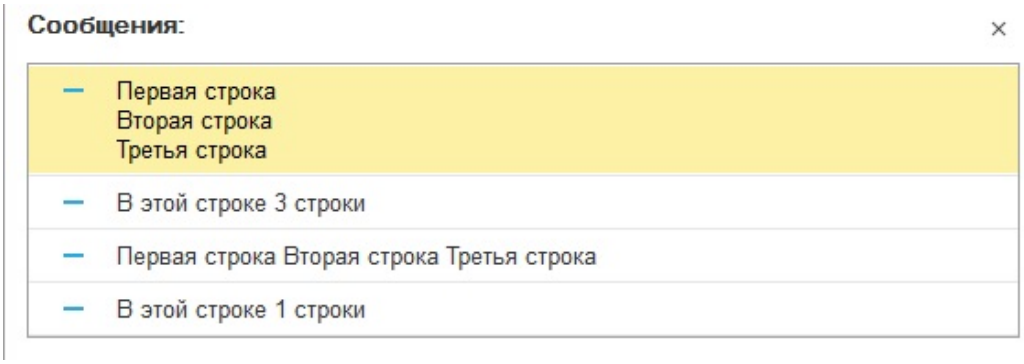


Рис. 2.5.23

Эта функция простая, и у Вас не должно возникнуть вопросов.

Функция СтрПолучитьСтроку

Следующая функция ***СтрПолучитьСтроку***. Эта функция получает из многострочной строки строку с заданным номером.

Она имеет следующий синтаксис:

СтрПолучитьСтроку(<Строка>,<Номер строки>)

Где:

«Строка» - это заданная строка;

«Номер строки» - номер строки, которая будет возвращена (нумерация строк ведется с 1).

Если номер будет больше количества строк или равен нулю, то ничего не вернется, при этом ошибки тоже не будет. Функция вернет пустую строку.

Сделайте в Вашей обработке следующий пример:

```
МногоСтрок = "Первая строка
             |Вторая строка
             |Третья строка";
ВтораяСтрока = СтрПолучитьСтроку(МногоСтрок,2);
Сообщить(ВтораяСтрока);
```

Листинг 2.5.28

В данном примере мы задаем многострочную строку. А потом получаем вторую строку в ней.

Посмотрите самостоятельно, как будет работать этот код.

Небольшое задание: поразбивайте, как мы это делали прежде, ФИО на строки, и получите какую-нибудь строку с помощью этой функции. Номер строки можете задавать интерактивно с помощью функции *ВвестиЧисло*. Сделайте так, чтобы пользователь мог вводить только целые числа.

Функции НРег, ВРег, ТРег

Теперь рассмотрим три очень похожие функции – ***НРег, ВРег, ТРег.***

Первая функция приводит все символы в нижний регистр, вторая в верхний регистр, третья в титульный регистр, т.е. первый символ слова будет верхним.

Задаются они очень просто:

НРег(<Строка>)

ВРег(<Строка>)

ТРег(<Строка>)

Сделайте следующий пример:

```
ФИО = "Иванов Петр Григорьевич";
ФИОВРег = ВРег(ФИО);
ФИОНРег = НРег(ФИО);
ФИОТРег = ТРег(ФИО);

Сообщить(ФИОВРег);
Сообщить(ФИОНРег);
Сообщить(ФИОТРег);
```

Листинг 2.5.29

Результат должен быть следующим:

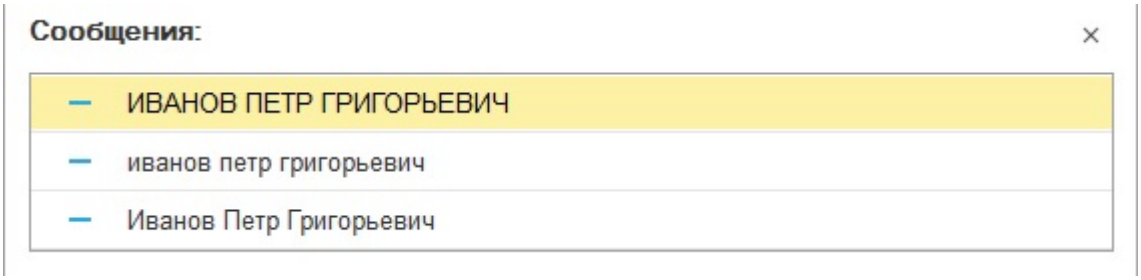


Рис. 2.5.24

Функции простые. Теперь небольшое задание. В предыдущем примере, где несколько строк, приведите первую строку в верхний регистр, вторую - в нижний регистр, третью - в титульный регистр.

Функция ПустаяСтрока

И самая последняя функция, которую мы рассмотрим, будет ***ПустаяСтрока***.

Она имеет один параметр - саму строку, и возвращает *Истина*, если строка пустая, и возвращает *Ложь*, если строка содержит символы.

Пустую строку можно задать, написав сдвоенные кавычки : "", пробела между ними быть не должно.

Сделайте следующий пример:

```
ФИО = "Иванов Петр Григорьевич";
ФИО2 = "";
ПризнакПустСтр1 = ПустаяСтрока(ФИО);
ПризнакПустСтр2 = ПустаяСтрока(ФИО2);

Сообщить(ПризнакПустСтр1);
Сообщить(ПризнакПустСтр2);
```

Листинг 2.5.30

И посмотрите, что у нас получается.

Резюме

Мы рассмотрели все нужные функции для работы со строками. Практика показывает, что знание этих функций позволит Вам решить большинство задач со строками, которые возникают при работе с программами 1С.

Часть 6. Переменные типа Дата

Итак, в этой части нашей главы мы разберем переменные типа ***Дата***.

Создайте новую обработку, в которой Вы будете проводить все манипуляции с переменными типа Дата.

Из первой части мы знаем, что переменные вводятся либо в виде 'ГГГГММДДччммсс', либо в виде 'ГГГГММДД' (см. стр. 56). Сейчас мы рассмотрим, как их вводить интерактивно и в дальнейшем в этой части главы все переменные типа *Дата* будем вводить интерактивно.

Функция ВвестиДату

Первым делом мы рассмотрим функцию ***ВвестиДату***. Она возвращает *Истина*, если введена дата, или *Ложь*, если нет.

Данная функция имеет следующий синтаксис:

ВвестиДату(<Дата>, <Подсказка>, <ЧастьДаты>)

Где:

«Дата» - это переменная, в которую будет записана дата;

«Подсказка» – текстовая строка - интерактивная подсказка на форме;

«Часть даты» – задает то, в каком виде будет записана дата.

Разберем последний параметр подробно.

Он задается в виде системного перечисления *ЧастьДаты*, у которого имеются три значения: *Время*, *Дата* и *ДатаВремя*.

ЧастьДаты.Время

ЧастьДаты.Дата

ЧастьДаты.ДатаВремя

Для лучшего понимания выполните следующий пример:

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)
    Перем НашаДата;
    ВвестиДату(НашаДата,"Введите дату", ЧастиДаты.Дата);
    Сообщить(НашаДата);
КонецПроцедуры
```

Листинг 2.6.1

Разберем данный код.

Сначала мы задаем переменную. Потом пишем функцию для интерактивного ввода даты, где первым параметром задаем вновь созданную дату, вторым - заголовок интерактивного окна, третьим – в каком виде будет задаваться дата пользователем.

Запустите обработку и введите любую дату.

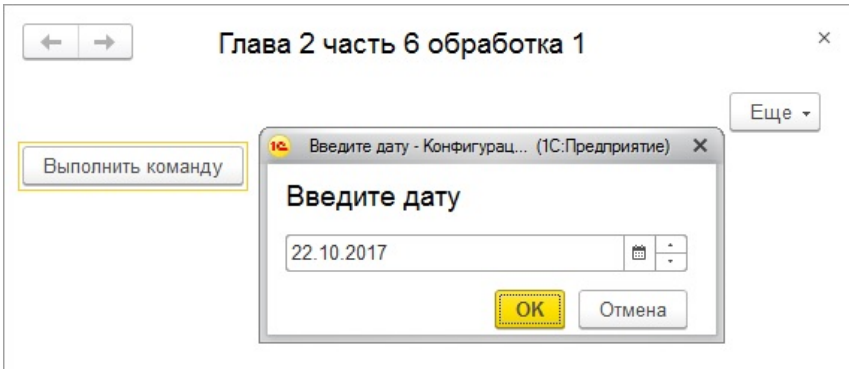


Рис. 2.6.1

Изменим значение третьего параметра на *ЧастиДаты.ДатаВремя*, и посмотрим, в каком формате Вы сможете вводить дату (самостоятельно попробуйте со значением *ЧастиДаты.Время*).

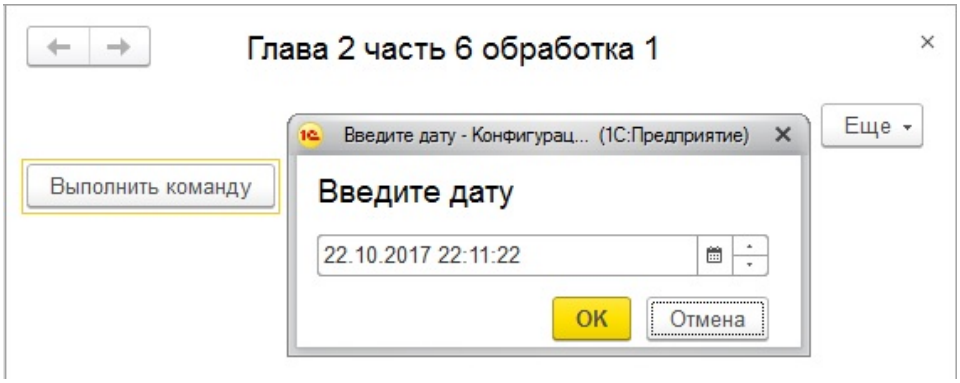


Рис. 2.6.2

Нажимаем кнопку *«ОК»*, и выходит следующее сообщение:

Рис. 2.6.3

Это не очень красиво и читаемо. Так получилось, потому что функция *Сообщить* конвертировала дату в строку, один к одному как есть. Чтобы такого не происходило, мы разберем функцию, которая будет конвертировать тип *Дата* в тип *Строка* по нужным нам правилам.

Функция Формат

Рассмотрим функцию ***Формат***. Данная функция может конвертировать в строку не только даты, но и числа, и булево. Но мы изучим ее работу только с типом *Дата*.

У данной функции следующий синтаксис:

Формат(<Значение>, <ФорматнаяСтрока>)

Где:

«Значение» – это значение нашей переменной;

«ФорматнаяСтрока» – строковое значение, которое включает в себя параметры вывода на экран.

Рассмотрим функцию применительно к датам. Для этого сделайте следующий пример:

```
Перем НашаДата;

ВвестиДату(НашаДата,"Введите дату", ЧастиДаты.ДатаВремя);
НашаДатаСтрока = Формат(НашаДата,"ДФ = ""дд.ММ.гггг""");
Сообщить(НашаДатаСтрока);
```

Листинг 2.6.2

Посмотрите, как дата вышла в окно сообщений:

Рис. 2.6.4

Разберем форматную строку применительно к датам. Как Вы видите, данная форматная строка имеет тип «Строка».

У любой форматной строки в данной функции должно быть имя. В рассматриваемом случае оно - **ДФ**. И значение - **"дд.ММ.гггг"**. Поскольку значение должно быть в кавычках, и мы знаем из предыдущей части, что кавычки внутри строки задаются двойными кавычками, то мы ставим двойные кавычки вокруг значения. Если бы мы вводили значение интерактивно, то оно имело бы следующий вид ДФ="дд.ММ.гггг".

Рассмотрим имена форматной строки для даты.

ДФ – формат даты. Данный формат можно компоновать из множества различных значений. Все значения:

- д (d) - день месяца (цифрами) без лидирующего нуля;
- дд (dd) - день месяца (цифрами) с лидирующим нулем;
- ддд (ddd) - краткое название дня недели *);
- дддд (dddd) - полное название дня недели *);
- М (M) - номер месяца (цифрами) без лидирующего нуля;
- ММ (MM) - номер месяца (цифрами) с лидирующим нулем;
- ММММ (MMMM) - краткое название месяца *);
- ММММ (MMMM) - полное название месяца *);
- к (q) - номер квартала в году;
- г (y) - номер года без века и лидирующего нуля;
- гг (yy) - номер года без века с лидирующим нулем;
- гггг (yyyy) - номер года с веком;
- ч (h) - час в 12-часовом варианте без лидирующих нулей;
- чч (hh) - час в 12-часовом варианте с лидирующим нулем;
- Ч (H) - час в 24-часовом варианте без лидирующих нулей;
- ЧЧ (HH) - час в 24-часовом варианте с лидирующим нулем;
- м (m) - минута без лидирующего нуля;
- мм (mm) - минута с лидирующим нулем;
- с (s) - секунда без лидирующего нуля;
- сс (ss) - секунда с лидирующим нулем;
- вв (tt) - отображение половины дня AM/PM (действительно только для языков конфигурирования, поддерживающих 12-часовой вариант представления времени).

Регистр в данном написании важен!

Запоминать эти значения не нужно, их можно найти в синтаксис-помощнике функции «Формат» (для того чтобы перейти в синтаксис-помощник нужной функции, необходимо установить курсор на эту функцию и нажать комбинацию клавиш Ctrl+F1, в появившемся списке выбрать пункт «Глобальный контекст/Функции форматирования/Формат»).

Теперь о том, как компоновать формат даты из них. Делается это очень просто.

Можем написать вот так:

```
Формат(НашаДата,"ДФ = ""дд- ММММ-гггг """);
```

```
Формат(НашаДата,"ДФ = ""ММММ-гггг """);
```

Или даже вот так:

```
Формат(НашаДата,"ДФ = ""ММММ-гггг/дд(гг) """);
```

Т.е. в этих кавычках значения Вы можете скомпоновать любую дату или время так, как Вам нравится, но не рекомендую использовать слова и буквы, чтобы не получился такой казус:

```
НашаДатаСтрока = Формат(НашаДата,"ДФ = ""День: дд, месяц: ММММ""");
```

Листинг 2.6.3

Запускаем. Вот как некрасиво вышло:

Рис. 2.6.5

Поэтому используйте только спецсимволы.

Рассмотрим следующее имя - **ДЛФ**, это локальный формат даты, который указывает вариант отображения частей даты. Их всего пять значений:

Д – дата цифрами

ДД – дата с месяцем прописью

ДВ – дата с временем

В - время

ДДВ – дата с месяцем прописью и временем.

В этом варианте нет таких широких возможностей для творчества. Задается все просто.

Например: ДЛФ = ДД.

Сделайте следующий пример:

```
Перем НашаДата;

ВвестиДату(НашаДата,"Введите дату", ЧастиДаты.ДатаВремя);
НашаДатаСтрока = Формат(НашаДата,"ДЛФ = Д");
Сообщить(НашаДатаСтрока);
НашаДатаСтрока = Формат(НашаДата,"ДЛФ = ДД");
Сообщить(НашаДатаСтрока);
НашаДатаСтрока = Формат(НашаДата,"ДЛФ = ДВ");
Сообщить(НашаДатаСтрока);
НашаДатаСтрока = Формат(НашаДата,"ДЛФ = В");
Сообщить(НашаДатаСтрока);
НашаДатаСтрока = Формат(НашаДата,"ДЛФ = ДДВ");
Сообщить(НашаДатаСтрока);
```

Листинг 2.6.4

Введем следующее значение:

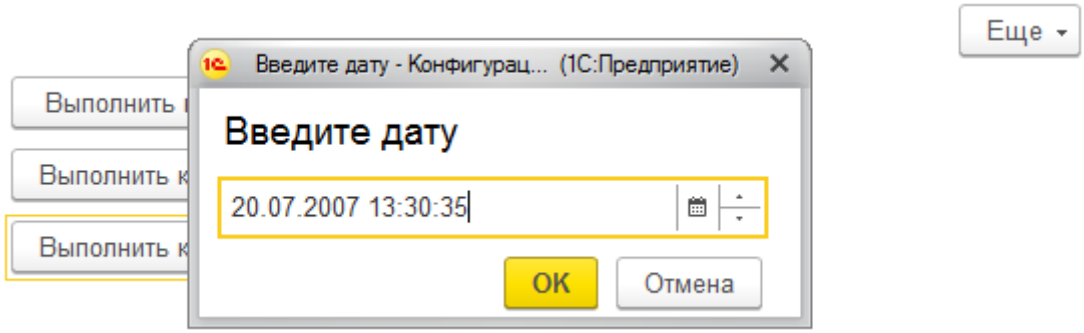


Рис. 2.6.6

И получим такой результат:

Рис. 2.6.7

Конструктор форматной строки

В платформе 8.3 есть возможность не набирать форматную строку вручную, а использовать для этих целей *конструктор форматной строки*. Чтобы создать форматную строку при помощи конструктора форматной строки, нужно во втором параметре функции *Формат* написать две кавычки и поставить между ними курсор (см. рис. 2.6.8).

```
Перем НашаДата;

ВвестиДату(НашаДата,"Введите дату", ЧастиДаты.ДатаВремя);
НашаДатаСтрока = Формат(НашаДата,"");
```

Рис. 2.6.8

После этого вызвать контекстное меню, нажав правую клавишу мышки, где выбираем пункт «Конструктор форматной строки» (см. рис. 2.6.9).

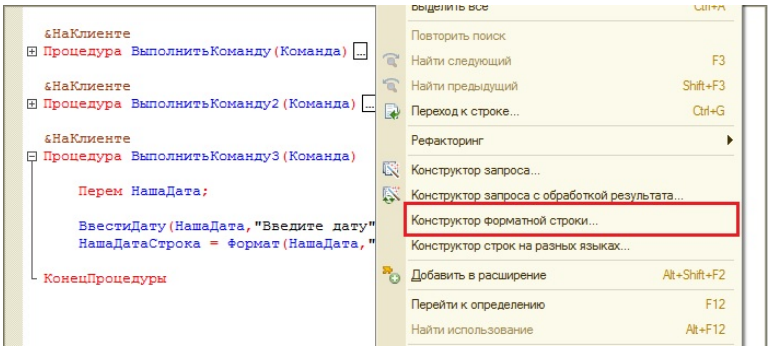


Рис. 2.6.9

В открывшемся конструкторе нас интересует закладка «Дата», в которой Вы можете выбрать как свободный формат даты, так и локальный (см. рис. 2.6.10).

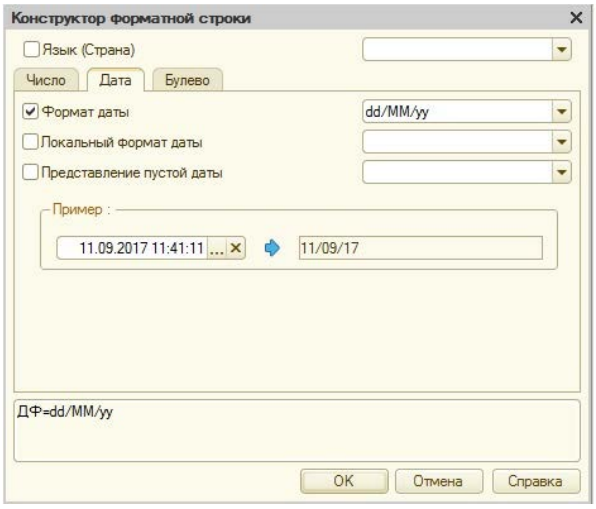


Рис. 2.6.10

После нажатия кнопки «ОК» нужный формат даты встанет в форматную строку (см. рис. 2.6.11).

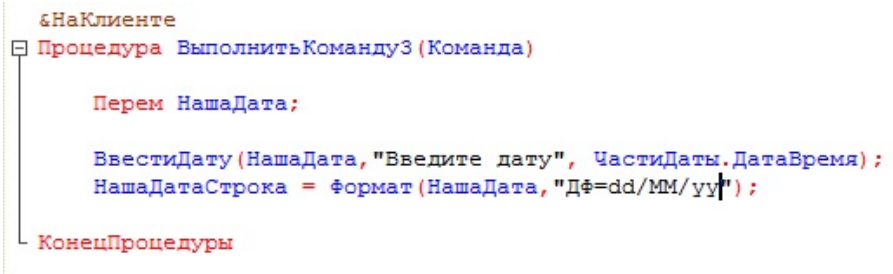


Рис. 2.6.11

Итак, мы рассмотрели, как нам красиво выводить даты на экран, следующим шагом перейдем непосредственно к функциям, работающим с датами.

Вычитание дат

Рассмотрим интересную особенность типа *Дата* - это способность вычитания дат. Даты можно вычитать одну из другой. Результатом будет точная разница между датами в секундах.

Рассмотрим пример:

```
&НаКлиенте
Процедура ВыполнитьКоманду4(Команда)
    Перем Дата1, Дата2;

    ВвестиДату(Дата1,"Введите первую дату", ЧастиДаты.ДатаВремя);
    ВвестиДату(Дата2,"Введите вторую дату", ЧастиДаты.ДатаВремя);

    РазницаДат = Дата1 - Дата2;

    Сообщить("Между " + Формат(Дата1,"ДЛФ=DT") + "
             |и " + Формат(Дата2,"ДЛФ=DT") + "
             | "  + РазницаДат + " секунд");

КонецПроцедуры
```

Листинг 2.6.5

Выведем результат на экран.

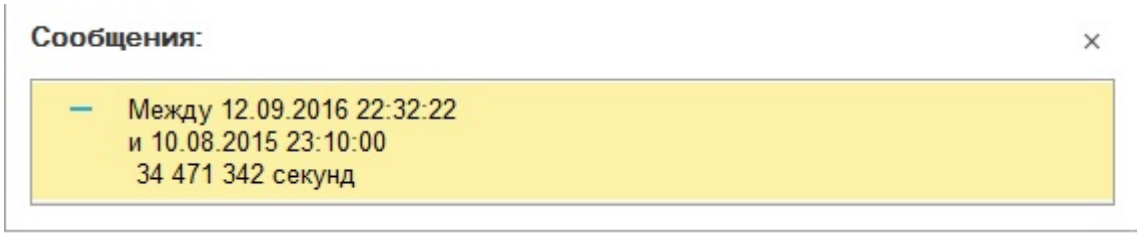


Рис. 2.6.12

Понятно, что количество секунд - это не информативные данные, а как узнать ровное количество дней между датами, мы узнаем немного позже. Наберитесь терпения.

Функции Год, Месяц, День, Час, Минута, Секунда

Рассмотрим ряд похожих функций для работы с типом дата. Это функции:

Год(<Дата>), Месяц(<Дата>), День(<Дата>),

Час(<Дата>), Минута(<Дата>), Секунда(<Дата>).

У данных функций один параметр – дата, а возвращают они число, которое соответствует году, месяцу, дню, часу, минуте и секунде соответственно. Рассмотрим пример для наглядности.

```
&НаКлиенте
Процедура ВыполнитьКоманду5(Команда)

    Перем НашаДата;

    ВвестиДату(НашаДата,"Введите дату", ЧастиДаты.ДатаВремя);

    ПеремГод  = Год(НашаДата);
    ПеремМес  = Месяц(НашаДата);
    ПеремДень = День(НашаДата);
    ПеремЧас  = Час(НашаДата);
    ПеремМин  = Минута(НашаДата);
    ПеремСек  = Секунда(НашаДата);

    Сообщить("В дате " + Формат(НашаДата,"ДЛФ=DT") + "
             |Год     - "  + ПеремГод + "
             |Месяц   - "  + ПеремМес + "
             |День    - "  + ПеремДень+ "
             |Час     - "  + ПеремЧас + "
             |Минута  - "  + ПеремМин + "
             |Секунда - "  + ПеремСек);
КонецПроцедуры
```

Листинг 2.6.6

На выходе должен получиться следующий результат:

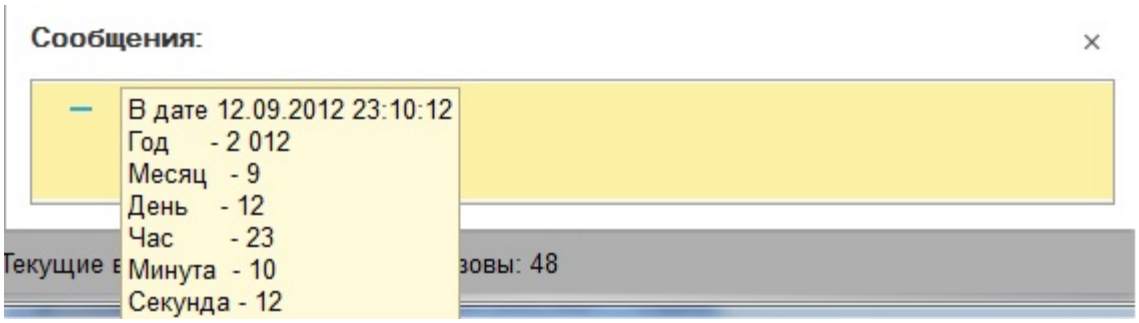


Рис. 2.6.13

Функции НачалоГода, НачалоДня, НачалоКвартала, НачалоМесяца, НачалоНедели, НачалоЧаса, НачалоМинуты

Теперь рассмотрим еще ряд похожих друг на друга функций. Это:

НачалоГода(<Дата>), НачалоКвартала(<Дата>),

НачалоМесяца(<Дата>), НачалоНедели(<Дата>),

НачалоДня(<Дата>), НачалоЧаса(<Дата>),

НачалоМинуты(<Дата>)

Все они возвращают дату и время начала периода, именем которого они называются. Т.е. если мы берем эту дату 12.09.2012, то функция *НачалоГода* возвратит дату *01.01.2012*, а функция *НачалоНедели* возвратит дату *10.09.2012*. Чтобы было наглядно, мы рассмотрим очень простой пример:

```
    Перем НашаДата;

    ВвестиДату(НашаДата,"Введите дату", ЧастиДаты.ДатаВремя);

    ДатаНачГода     = НачалоГода(НашаДата);
    ДатаНачКвартала = НачалоКвартала(НашаДата);
    ДатаНачМесяца   = НачалоМесяца(НашаДата);
    ДатаНачНедели   = НачалоНедели(НашаДата);
    ДатаНачДня      = НачалоДня(НашаДата);
    ДатаНачЧаса     = НачалоЧаса(НашаДата);
    ДатаНачМинуты   = НачалоМинуты(НашаДата);

    НашаДатаФормат = Формат(НашаДата,"ДЛФ=DT");

    Сообщить("Мы ввели дату " + НашаДатаФормат);
    Сообщить("НачалоГода( " + НашаДатаФормат + " ) = " + ДатаНачГода);
    Сообщить("НачалоКвартала( " + НашаДатаФормат + " ) = " +
ДатаНачКвартала);
    Сообщить("НачалоМесяца( " + НашаДатаФормат + " ) = " + ДатаНачМесяца);
    Сообщить("НачалоНедели( " + НашаДатаФормат + " ) = " + ДатаНачНедели);
    Сообщить("НачалоДня( " + НашаДатаФормат + " ) = " + ДатаНачДня);
    Сообщить("НачалоЧаса( " + НашаДатаФормат + " ) = " + ДатаНачЧаса);
    Сообщить("НачалоМинуты( " + НашаДатаФормат + " ) = " + ДатаНачМинуты);
```

Листинг 2.6.7

Посмотрим, какой результат на выходе.

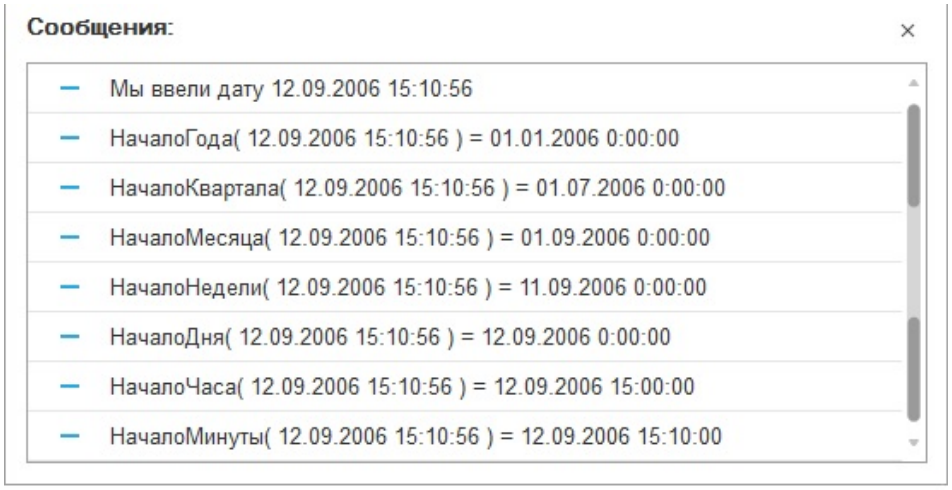


Рис. 2.6.14

Еще один пример. Есть у нас две даты, и нужно получить количество дней между ними, причем сразу целое. Можно, конечно, вычесть даты, потом поделить 3600, умноженное на 24, и округлить до целого. Но я не рекомендую Вам это делать, т.к. разница между датами

12 августа 2012 23:59:59, и 13 августа 2012 07:33:40 будет меньше одного дня, и при округлении или выделении целого получится 0.

Поэтому лучше сделать все проще. Получить начало дня первой даты, потом начало дня второй даты и вычесть первую из второй.

Посмотрим пример, демонстрирующий это:

```
Перем Дата1, Дата2;

ВвестиДату(Дата1,"Введите первую дату", ЧастиДаты.ДатаВремя);
ВвестиДату(Дата2,"Введите вторую дату", ЧастиДаты.ДатаВремя);

Дата1НачалоДня = НачалоДня(Дата1);
Дата2НачалоДня = НачалоДня(Дата2);

КолДней = (Дата1НачалоДня - Дата2НачалоДня)/(3600*24);

Сообщить("Между " + Формат(Дата1,"ДЛФ=DT") + " и
         |" + Формат(Дата2,"ДЛФ=DT") + " " + КолДней + " дней");
```

Листинг 2.6.8

Результат должен получиться следующим:

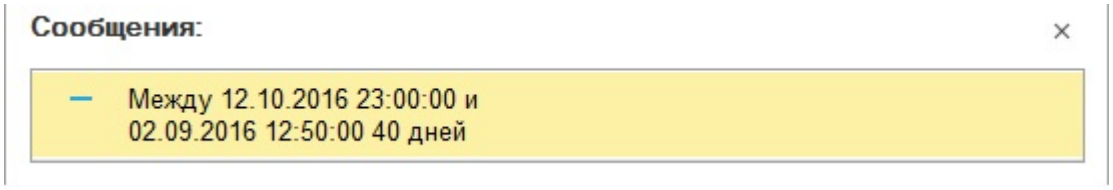


Рис. 2.6.15

Функции КонецГода, КонецДня, КонецКвартала, КонецМесяца, КонецНедели, КонецЧаса, КонецМинуты

Теперь рассмотрим противоположные функции, это:

КонецГода(<Дата>), КонецКвартала(<Дата>),

КонецМесяца(<Дата>), КонецНедели(<Дата>),

КонецДня(<Дата>), КонецЧаса(<Дата>) и

КонецМинуты(<Дата>).

Все они возвращают дату и время конца периода, именем которого они называются. Т.е. если мы берем эту дату 12.09.2012 12.12.02, то функция *КонецГода* возвратит дату 31.12.2012 23.59, а функция *КонецНедели* возвратит дату 16.09.2012 23.59.59. Рассмотрим очень простой пример:

```
&НаКлиенте
Процедура ВыполнитьКоманду7(Команда)
    Перем НашаДата;

    ВвестиДату(НашаДата,"ВведитеДату",ЧастиДаты.ДатаВремя);

    ДатаКонецГода       = КонецГода(НашаДата);
    ДатаКонецКвартала = КонецКвартала(НашаДата);
    ДатаКонецМесяца    = КонецМесяца(НашаДата);
    ДатаКонецНедели    = КонецНедели(НашаДата);
    ДатаКонецДня       = КонецДня(НашаДата);
    ДатаКонецЧаса       = КонецЧаса(НашаДата);
    ДатаКонецМинуты    = КонецМинуты(НашаДата);

    НашаДатаФормат      = Формат(НашаДата,"ДЛФ = ДДВ");

    Сообщить("Мы ввели дату " + НашаДатаФормат);
    Сообщить("КонецГода("+ НашаДата + ") = "     + ДатаКонецГода);
    Сообщить("КонецКвартала("+ НашаДата + ") = " + ДатаКонецКвартала);
    Сообщить("КонецМесяца(" + НашаДата + ")  = " + ДатаКонецМесяца);
    Сообщить("КонецНедели(" + НашаДата + ")  = " + ДатаКонецНедели);
    Сообщить("КонецДня(" + НашаДата + ")     = " + ДатаКонецДня);
    Сообщить("КонецЧаса(" + НашаДата + ")    = " + ДатаКонецЧаса);
    Сообщить("КонецМинуты(" + НашаДата + ")  = " + ДатаКонецМинуты);

КонецПроцедуры
```

Листинг 2.6.9

Рис. 2.6.16

Для тренировки попробуйте разницу дат сделать с помощью функции *КонецДня*.

Функции ДеньГода, ДеньНедели, НеделяГода

Рассмотрим еще три похожие функции, это:

ДеньГода(<Дата>)

ДеньНедели(<Дата>)

НеделяГода(<Дата>)

У них один параметр, и они возвращают соответственно номер дня года, номер дня недели и номер недели года указанной даты.

Рассмотрим простой пример:

```
&НаКлиенте
Процедура ВыполнитьКоманду8(Команда)
    Перем НашаДата;

    ВвестиДату(НашаДата,"Введите дату",ЧастиДаты.Дата);

    НомерДняГода = ДеньГода(НашаДата);
    НомерДняНедели = ДеньНедели(НашаДата);
    НомерНеделиГода = НеделяГода(НашаДата);

    Сообщить("День " + Формат(НашаДата,"ДЛФ = ДД") + " является
             |" + НомерДняГода + " днем в году.
             |" + НомерДняНедели + " днем в недели.
             |Неделя этого дня: " + НомерНеделиГода + " неделя в году.");
КонецПроцедуры
```

Листинг 2.6.10

Выводим все это на экран.

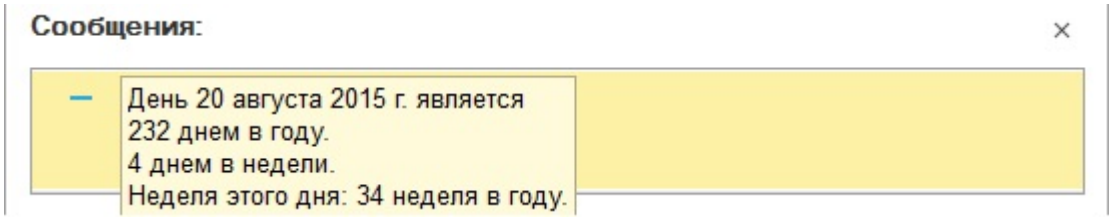


Рис. 2.6.17

Функция ДобавитьМесяц

Разберем очень важную функцию для работы с датами - это функция ***Добавить месяц***.

Она имеет следующий синтаксис:

ДобавитьМесяц(<Дата>,<Количество мМесяцев>)

Дата - это заданная дата.

Количество месяцев - этот параметр имеет тип число – количество месяцев, которые прибавляются или отнимаются (если число отрицательное) от заданной даты.

Рассмотрим пример. Пользователь задает некоторую дату, и нам надо получить начало следующего месяца после этой даты, начало следующего квартала и начало следующего года.

```
&НаКлиенте
Процедура ВыполнитьКоманду9(Команда)
    Перем НашаДата;

    ВвестиДату(НашаДата,"Введите дату",ЧастиДаты.Дата);

    НачалоТекущегоМесяца = НачалоМесяца(НашаДата);
    НачалоСледующегоМесяца = ДобавитьМесяц(НачалоТекущегоМесяца,1);

    НачалоТекущегоКвартала = НачалоКвартала(НашаДата);
    НачалоСледующегоКвартала = ДобавитьМесяц(НачалоТекущегоКвартала,3);

    НачалоТекущегоГода       = НачалоГода(НашаДата);
    НачалоСледующегоГода     = ДобавитьМесяц(НачалоТекущегоГода,12);

    Сообщить("Вы ввели следующую дату: "
             + Формат(НашаДата,"ДЛФ = ДД"));
    Сообщить("Начало следующего месяца от этой даты: "
             + Формат(НачалоСледующегоМесяца,"ДЛФ = ДД"));
    Сообщить("Начало следующего квартала от этой даты: "
             + Формат(НачалоСледующегоКвартала,"ДЛФ = ДД"));
    Сообщить("Начало следующего года от этой даты: "
             +  Формат(НачалоСледующегоГода,"ДЛФ = ДД") + ".");
КонецПроцедуры
```

Листинг 2.6.11

И выводим все это на экран.

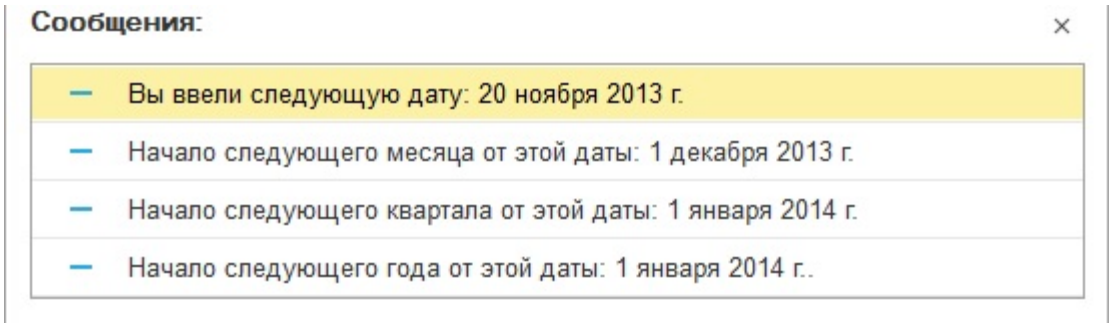


Рис. 2.6.18

Обращаю Ваше внимание, что данный код можно было написать короче, здесь я написал так, чтобы Вам было более понятно.

Вот короткий вариан:

```
    Перем НашаДата;

    ВвестиДату(НашаДата,"Введите дату",ЧастиДаты.Дата);

    НачалоСледующегоМесяца = ДобавитьМесяц(НачалоМесяца(НашаДата),1);
    НачалоСледующегоКвартала = ДобавитьМесяц(НачалоКвартала(НашаДата),3);
    НачалоСледующегоГода     = ДобавитьМесяц(НачалоГода(НашаДата),12);
```

```
Сообщить("Вы ввели следующую дату: "
         + Формат(НашаДата,"ДЛФ = ДД"));
Сообщить("Начало следующего месяца от этой даты: "
         + Формат(НачалоСледующегоМесяца,"ДЛФ = ДД"));
Сообщить("Начало следующего квартала от этой даты: "
         + Формат(НачалоСледующегоКвартала,"ДЛФ = ДД"));
Сообщить("Начало следующего года от этой даты: "
         +  Формат(НачалоСледующегоГода,"ДЛФ = ДД") + ".");
```

Листинг 2.6.12

Теперь для тренировки получите начало предыдущего месяца, предыдущего квартала, предыдущего года от введенной даты.

Функция ТекущаяДата()

И, под конец, простая функция – ***ТекущаяДата()***. У нее нет никаких параметров. Функция возвращает текущую системную дату на компьютере.

Рассмотрим простой пример:

```
ДатаТекущая = ТекущаяДата();
Сообщить("На данный момент текущая дата " +
          Формат(ДатаТекущая, "ДЛФ = ДДВ"));
```

Листинг 2.6.13

В данном примере мы задаем переменной значение текущей даты и выводим это на экран. Теперь можете самостоятельно поработать с данной датой, вывести в различных форматах.

Резюме

На этом наше знакомство с типом *Дата* закончилось. В данной части Вы узнали все необходимое для работы в программе 1С с типом *Дата.* Тип *Дата* Вам будет встречаться очень часто, поскольку «1С:Предприятие» - программа для автоматизации учетных систем, а они все работают с датами. Поэтому если Вам не понятна работа с данным типом, то лучше углубленно изучить его, используя дополнительные материалы.

Часть 7. Функции преобразования и общие функции для примитивных типов

В последней части нашего урока мы разберем функции преобразования типов и общие функции для примитивных типов. Так же, как и в предыдущих уроках, создайте новую обработку, на форме которой Вы будете проводить все манипуляции с преобразованием типов.

Функции преобразования

В данной части мы рассмотрим только три вида преобразования: из строки в число, из числа в строку, чисел и строк в дату.

Преобразование из числа в строку

Делается это достаточно просто:

Строка(1.212)

Рассмотрим пример:

```
&НаКлиенте
Процедура ПреобразованиеИзЧислаВСтроку(Команда)
    Перем ПеремЧисло;

    Надпись = "Наше число - ";

    ПеремЧисло = 1.22112;
    ПеремСтрока = Строка(ПеремЧисло);

    НадписьСЧислом = Надпись + ПеремСтрока;

    Сообщить(НадписьСЧислом)
КонецПроцедуры
```

Листинг 2.7.1

В данном примере мы сначала задаем переменную и присваиваем строковое значение переменной *Надпись* и числовое значение переменной *ПеремЧисло*. Далее, преобразуем число в строку. Складываем две строки. Выводим в окно сообщений.

Результат должен быть следующим:

Рис. 2.7.1

Все просто. Но пытливый читатель спросит меня: зачем нам нужна данная функция преобразования, если в предыдущих примерах мы просто складывали строку с числом, и получалась строка? Да, это верно, так и есть, и если мы переделаем наш пример по-другому:

```
Перем ПеремЧисло;

Надпись = "Наше число - ";

ПеремЧисло = 1.22112;

НадписьСЧислом = Надпись + ПеремЧисло;

Сообщить(НадписьСЧислом)
```

Листинг 2.7.2

то он выдаст точно такой же результат.

Но есть моменты, когда без использования данного преобразования не обойтись. Например, когда мы используем два числа, которые необходимо сначала сложить, прежде чем вывести, можно это сделать через отдельную переменную, а можно и сложив внутри оператора «Строка».

Сделайте следующий пример:

```
&НаКлиенте
Процедура ПреобразованиеИзЧислаВСтроку2(Команда)

    Перем ПеремЧисло1,ПеремЧисло2;

    Надпись = "Наше число - ";
    ПеремЧисло1 = 1.22112;
    ПеремЧисло2 = 1.213444;
    НадписьСЧислом = Надпись + Строка(ПеремЧисло1+ПеремЧисло2);

    Сообщить(НадписьСЧислом);

КонецПроцедуры
```

Листинг 2.7.3

Рис. 2.7.2

Потом сложите их без оператора *Строка*, и посмотрите, что получится.

Преобразование из строки в число

Рассмотрим преобразование из строки в число. Оно довольно редко применимо, но все же я Вам о нем расскажу, т.к. бывают иногда случаи, когда без данного преобразования не обойтись.

Выполняется оно просто: пишется слово *Число* и внутри в скобках строка в виде числа.

Рассмотрим пример:

```
&НаКлиенте
Процедура ПреобразованиеИзСтрокиВЧисло(Команда)
    Перем ПеремСтрока1, ПеремСтрока2;

    ПеремСтрока1 = "45";
    ПеремСтрока2 = "34";

    ПеремЧисло1 = Число(ПеремСтрока1);
    ПеремЧисло2 = Число(ПеремСтрока2);

    Сообщить(ПеремСтрока1 + " + "+ ПеремСтрока2 + " = " + Строка(ПеремЧисло1
+ ПеремЧисло2));
КонецПроцедуры
```

Листинг 2.7.4

Объясним данный код: задаем две переменные и присваиваем им строковые значения в виде цифр. Преобразуем их в числа. Потом в сообщении их складываем и выводим в окно.

Рис. 2.7.3

В реальности на моей практике такая задача стояла, и не раз, когда приходилось обрабатывать различные выгрузки из Excel, DBF, XML. Но это уже предмет немножко другого разговора.

Преобразование в дату

Для преобразования в дату используется функция *Дата*. У нее есть два синтаксиса: для строк и для чисел.

Рассмотрим синтаксис для строк. Для того, чтобы преобразовать строку в дату, она должна иметь следующий вид (его мы разбирали в ч 5.): «ГГГГММДДЧЧММСС».

Это может выглядеть так:

Дата("20120922200202");

Синтаксис для чисел будет следующим:

Дата(Год,Месяц,День,Час, Минута,Секунда).

Обязателен в данном формате только первый параметр. Потому можно написать и в таком виде:

Дата(Год,Месяц, День)

Или

Дата(Год,Месяц)

Или

Дата(Год)

Сделайте следующий пример:

```
&НаКлиенте
Процедура ПреобразованиеВДату(Команда)
    Дата1 = Дата("20120909110012");
    Дата2 = Дата(2012,09,09,11,0,12);
    Дата3 = Дата(2012,09,09);

    Сообщить(Формат(дата1,"ДЛФ = ДДВ"));
    Сообщить(Формат(дата2,"ДЛФ = ДДВ"));
    Сообщить(Формат(дата3,"ДЛФ = ДДВ"));
КонецПроцедуры
```

Листинг 2.7.5

В данном примере мы задаем даты в двух разных видах и выводим это все в окно сообщений. Поэкспериментируйте самостоятельно с различными способами задания дат.

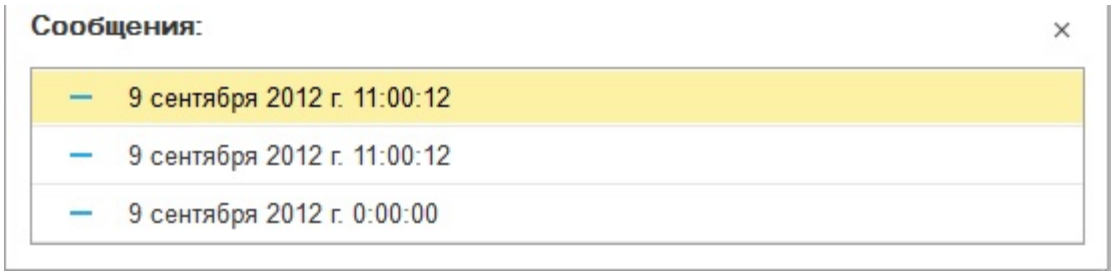


Рис. 2.7.4

И напоследок в этом уроке мы изучим общие функции, которые работают для всех переменных примитивного типа.

Функции Мин, Макс

Начнем с функции ***Мин***, данная функция возвращает минимальное значение из ряда параметров. Параметры могут быть как одного типа, так и разных типов. Если параметры будут разных типов, то они будут преобразовываться в тип первого параметра, если преобразование будет невозможным, то сгенерируется ошибка.

Число возвращается минимальное, строка - самая маленькая по длине, дата – самая ранняя, булево – ложь.

```
&НаКлиенте
Процедура ФункцияМин(Команда)
    ЧислоМин = Мин(12.11,221,34556,0.11,-2222);

    СтрокаМин= Мин("ааа","аа","ууууу","ццццццццц");

    ДатаМин  =
Мин('20120122090934','20110122090934','19990122090934','20240522090934');

    БулевоМин = Мин(Истина, Ложь, Ложь,Истина);

    Сообщить("Минимальное число  =" + ЧислоМин);
    Сообщить("Минимальная строка =" + СтрокаМин);
    Сообщить("Минимальное дата   =" + Формат(ДатаМин,"ДЛФ = ДДВ"));
    Сообщить("Минимальное булево =" + БулевоМин);
КонецПроцедуры
```

Листинг 2.7.6

В данном примере мы получаем минимальные значения чисел, строк, дат и булево. Поэкспериментируйте самостоятельно, задавая в данные функции значения разных типов.

Рис. 2.7.5

Следующая функция – это функция ***Макс***. Она возвращает максимальное значение из ряда параметров. Параметры аналогичны параметрам функции *Мин*.

Самостоятельно повторите последний пример, только с функцией *Макс*, с примерами из листинга 1.7.6 должен получиться результат, как на рисунке 1.7.6.

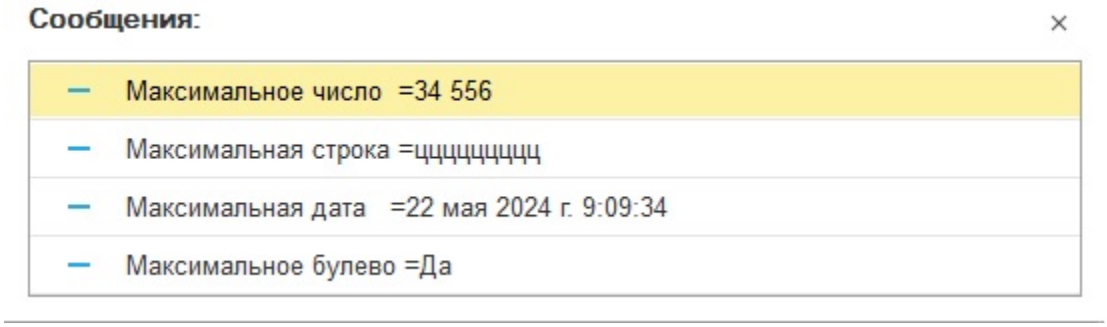


Рис. 2.7.6

Резюме

В этой части Вы научились преобразовывать строки в числа и наоборот, а также получать тип *Дата*, используя строковые и числовые типы. Функция *Мин* и *Макс* будет полезна Вам при работе с датами, поскольку часто приходится из нескольких дат выбирать самую раннюю или самую позднюю дату.

Глава 3. Основные операторы

Третья глава у нас очень объемная по знаниям, в ней мы научимся работать с конструкцией *Если*, разберем два вида цикла, будем пробовать делать собственные процедуры и функции и освоим такой нужный оператор как *Попытка*.

Часть 1. Условие: Если…Тогда…ИначеЕсли

Начнем мы с очень необходимого оператора, без которого просто никуда. Это оператор условия ***Если…Тогда.***

Данный оператор управляет выполнением программы в зависимости от логического выражения (их может быть одно или несколько).

Простейший вариант данного оператора будет следующим:

Если <Условие> тогда

[Операция1];

……..

[ОперацияN];

……..

КонецЕсли;

Рассмотрим подробнее. Для того чтобы начали выполняться операции внутри конструкции *Если…Тогда*, нужно, чтобы *Условие* принимало значение *Истина*. Если *Условие* принимает значение *Ложь*, то программа не будет заходить внутрь конструкции. Все просто. Теперь пример.

Создайте самостоятельно обработку, форму и в обработчике команды формы напишем следующий код:

```
&НаКлиенте
Процедура ВыполнениеКоманды(Команда)
    Перем Число1;

    ВвестиЧисло(Число1,"ВведитеЧисло");

    Если Число1 > 0 тогда
        Сообщить("Число " + Число1);
        Сообщить("Больше нуля");
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.1.1

В данном коде мы указали, что если введенное число больше нуля, то выводим сообщение. В противном случае ничего не делаем.

Сохраните обработку и посмотрите, как она работает.

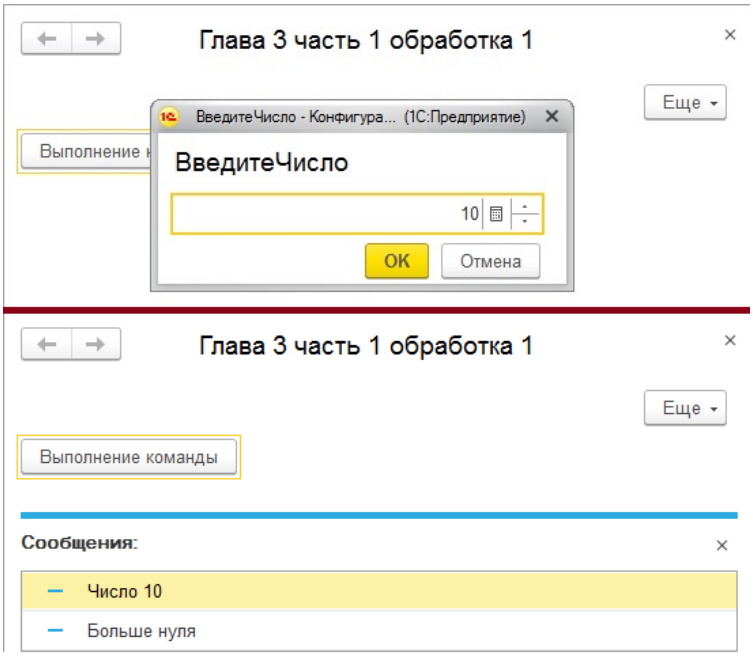


Рис. 3.1.1

Это самый примитивный вариант использования данного оператора.

Условие для данного оператора может быть каким угодно, главное, чтобы возвращалось значение типа Булево.

Например, может быть такой вариант:

Если Число1 > 1 и Число1 < 10 тогда….

Или такой вариант:

Условие = Число1 > 1 и Число1 < 10;

Если Условие тогда…

И так далее, более подробно про работу с булевыми операторами смотрите главу 2, часть 3 (см. стр. 63).

Продолжаем изучать дальше конструкцию «Если…». Что если нам надо выполнить какие-нибудь операции, когда выполняется наше условие, и другие операции, когда оно не выполняется? Для этого нам помогает оператор условия *Иначе*.

Рассмотрим схему:

Если <Условие> тогда

[Операция1];

…….

[ОперацияN];

Иначе

[Операция_2];

…….

[Операция_N]

КонецЕсли;

В данном случае выполнение программы после оператора *Иначе* будет, когда *Условие* примет значение *Ложь*.

Доработайте Ваш пример:

```
&НаКлиенте
Процедура ВыполнениеКоманды2(Команда)
    Перем Число1;

    ВвестиЧисло(Число1,"ВведитеЧисло");

    Если Число1 > 0 тогда
        Сообщить("Число " + Число1);
        Сообщить("Больше нуля");
    иначе
        Сообщить("Число " + Число1);
        Сообщить("Равно нулю или меньше нуля");
    КонецЕсли;

КонецПроцедуры
```

Листинг 3.1.2

В вышеприведенном коде мы вводим некоторое число в программу, после, как в предыдущем примере, пишем условие, что если это число больше нуля, то выводим соответствующее сообщение. Но также пишем оператор *Иначе*, после которого сообщаем, что число меньше или равно нулю. Запустите данную обработку и посмотрите, как она работает.

Опять усложним текущий пример. Что если нам необходимо вывести сообщение, когда число равно нулю, или, к примеру, равно 10?

Для этого мы используем условную конструкцию ***ИначеЕсли*** …***Тогда***.

Рассмотрим схему:

Если <Условие> тогда

[Операция1];

…….

[ОперацияN];

ИначеЕсли <Условие2> тогда

[Операция_2];

…….

[Операция_2N];

ИначеЕсли <Условие3> тогда

[Операция_2];

…….

[Операция_2N];

.

Иначе

[Операция_N];

…….

[Операция_NN]

КонецЕсли;

Операторов *ИначеЕсли…Тогда* внутри условия может быть бесконечное множество, оператор *Иначе* всегда должен быть один и идти после всех операторов *ИначеЕсли…Тогда*.

Теперь усложните Ваш пример:

```
&НаКлиенте
Процедура ВыполнениеКоманды3(Команда)
    Перем Число1;

    ВвестиЧисло(Число1,"ВведитеЧисло");

    Если Число1 >= 1 тогда
        Сообщить("Число " + Число1);
        Сообщить("Больше или равно единицы");
    ИначеЕсли Число1 >0 и Число1 < 1 тогда
        Сообщить("Число " + Число1);
        Сообщить("Меньше 1, но больше 0");
    ИначеЕсли Число1 = 0 тогда
```

```
        Сообщить("Число " + Число1);
        Сообщить("Равно нулю");
    иначе
        Сообщить("Число " + Число1);
        Сообщить("Меньше нуля");
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.1.3

В данном примере мы проверяем, больше или равно единицы введенное нами число, если да, то сообщаем об этом. Потом проверяем, входит ли оно в интервал от 0 до 1, если да, то сообщаем. Следующим шагом проверяем, равно ли оно нулю, если да, то сообщаем. И если число не удовлетворяет всем вышеприведенным условиям, т.е. число меньше нуля, выводим сообщение об этом.

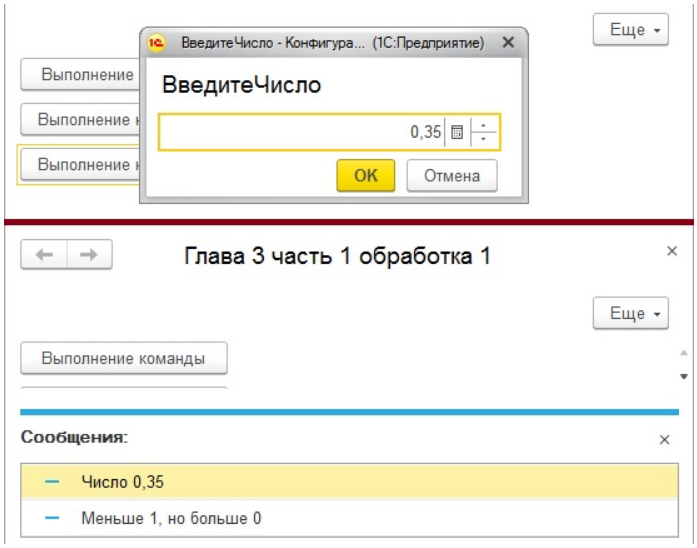


Рис. 3.1.2

Обращаю Ваше внимание, что условия *ИначеЕсли* будут выполняться только в том случае, когда не выполнились все предыдущие условия. Подправьте для примера Ваш код:

```
    Перем Число1;
    ВвестиЧисло(Число1,"ВведитеЧисло");

    Если Число1 > 0 тогда
        Сообщить("Число " + Число1);
        Сообщить("Больше или равно единицы");
    ИначеЕсли Число1 >0 и Число1 < 1 тогда
        Сообщить("Число " + Число1);
        Сообщить("Меньше 1, но больше 0");
    ИначеЕсли Число1 = 0 тогда
        Сообщить("Число " + Число1);
        Сообщить("Равно нулю");
    иначе
        Сообщить("Число " + Число1);
        Сообщить("Меньше нуля");
    КонецЕсли;
```

Листинг 3.1.4

Если мы изменим первое условие, что *Число1* больше 0, то программа никогда не зайдет во второе условие. Так как, к примеру, число *0.6* больше нуля, и выполнится код под этим условием, после чего происходит выход из оператора.

Обратите внимание на один момент, если в *Условии* идет сразу переменная типа булево, то не надо писать так:

Если НашеУсловие = Истина тогда…

Если НашеУсловие = Ложь тогда…

Достаточно просто написать условие, например:

Если НашеУсловие тогда…

Или

Если Не НашеУсловие тогда….

Теперь переделайте предыдущий пример. Создайте переменную типа булево с нужным условием и подставляйте ее в конструкцию *Если…Тогда*.

Закончим разбирать данный оператор. Попробуйте сами усложнить последний пример (листинг 3.1.4), добавив различные дополнительные сравнения.

Теперь изучим похожий оператор, но несколько упрощенный – *Вычислить выражение по условию*. Он имеет следующий синтаксис:

?(<Условие>,<Выражение1>,<Выражение2>)

Где:

«Условие» – это логическое выражение, в зависимости от результата которого будет возвращено либо *Выражение1*, либо *Выражение2*;

«*Выражение1»* – возвращается, если логическое выражение *Истина*;

«Выражение2» – возвращается, если логическое выражение *Ложь*;

Рассмотрим пример. Вводим некоторое число, и надо вывести сообщение, больше оно нуля или меньше.

```
&НаКлиенте
Процедура ВыполнениеКоманды4(Команда)
    Перем Число1;

    ВвестиЧисло(Число1,"Введите число");

    Сообщение = ?(Число1 > 0,"Число " + Строка(Число1) + " больше 0",
                             "Число " + Строка(Число1) + " меньше или равно
0");
```

```
    Сообщить(Сообщение);
КонецПроцедуры
```

Листинг 3.1.5

В изучаемом примере первый параметр оператора «?» это наше условие: *Число1 >0*, второй параметр – строка с текстом, что наше число больше нуля, оно будет возвращаться, если первый параметр будет *Истина*. И третий параметр – строка с текстом, что наше число ноль или меньше нуля, оно будет возвращаться, если первый параметр будет *Ложь*.

Посмотрите, как она работает.

Сделайте еще один пример: пользователь вводит некоторое число, нужно извлечь из него корень, для этого оно всегда должно быть положительным.

```
&НаКлиенте
Процедура ВыполнениеКоманды5(Команда)
    Перем Число1;

    ВвестиЧисло(Число1,"Введите число");
    ЧислоДляКорня = ?(Число1 < 0, Число1*(-1),Число1);
    Корень = Окр(Sqrt(ЧислоДляКорня),5);

    Сообщить("Корень из модуля числа " + Строка(Число1) + " равен " +
Строка(Корень));
КонецПроцедуры
```

Листинг 3.1.6

Разберите данный пример самостоятельно.

Резюме

В этой части Вы изучили один из основных операторов языка программирования 1С. Если Вам не понятно его действие, то смотрите, как работает программа в отладке. Вы должны хорошо владеть данным оператором, чтобы перейти к дальнейшему изучению языка программирования 1С.

Часть 2. Цикл: Для…Цикл

Итак, начнем изучать циклы. Что такое вообще *Цикл*? ***Цикл*** – действие, выполняемое определенное количество раз. Под действием понимается один или несколько операторов.

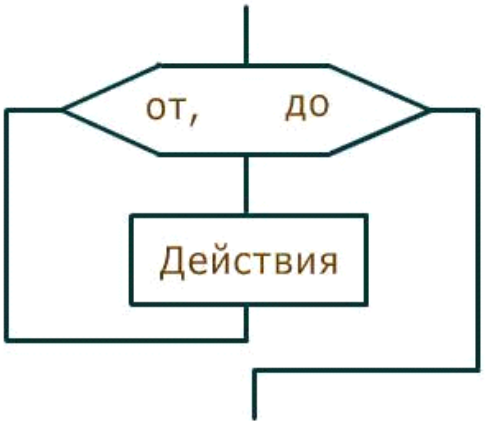


Рис. 3.2.1

Изучите рисунок 3.2.1, он как раз применим к осваиваемой теме. Пока идет перечисление какого-нибудь параметра от одного значения до другого, то выполняются определенные действия. Как только параметр достигает максимального значения, мы выходим из цикла и продолжаем выполнение программы.

Первый цикл который мы изучим, будет цикл ***Для…Цикл.***

Данный оператор имеет следующий синтаксис (пока мы его рассмотрим в самом общем виде, а углублять будем по мере изучения):

Для ИмяПеременной = Выражение1 по Выражение2 цикл

//операторы

КонецЦикла.

Где:

«ИмяПеременной» – это переменная счетчик цикла, его значение увеличивается на единицу при каждом повторении цикла;

«Выражение1» – числовое выражение, оно задает начальное значение, которое присваивается счетчику цикла при первом обходе. *Может быть любое целое число;*

«Выражение2» – максимальное значение счетчика цикла. Когда переменная *ИмяПеременной* становится больше *Выражение2*, программа выходит из цикла, т.е. операторы внутри конструкции *Для …Цикл* больше не выполняются. *Может быть любое целое число;*

«Цикл» – ключевое слово, которое открывает тело цикла;

«КонецЦикла» – ключевое слово, которое закрывает тело цикла;

Итерация – каждый обход цикла.

Все, что находится между ключевыми словами *Цикл* и *КонецЦикла,* будет выполняться, пока *ИмяПеременной* находится в интервале от *Выражение1* до *Выражение2*.

Обращаю внимание, что *Выражение1* всегда должно быть меньше или равно *Выражение2.*

Создайте самостоятельно обработку, форму, и в обработчике команды формы напишите следующий код:

```
&НаКлиенте
Процедура ВыполнитьКоманду1(Команда)

    Для  н = 1 по 10 цикл;
        Сообщить("н = " + н);
    КонецЦикла;

КонецПроцедуры
```

Листинг 3.2.1

В данном примере мы создали цикл *Для…Цикл*, где счетчик цикла называется *«н»*, и начальное его значение равно *1*, а конечное его значение равно *10*.

И при каждом обходе цикла, выводим сообщение о том, чему равно *«н»*.

Сохраните обработку, запустите и посмотрите, что у Вас выйдет в окно сообщений.

Циклы можно вкладывать друг в друга. Сделайте следующий пример:

```
&НаКлиенте
Процедура ВыполнитьКоманду2(Команда)
    Для  н = 1 по 5 цикл

        Для к = 1 по 3 Цикл
            Сообщить("н.к = " + н + "." + к );
        КонецЦикла;

    КонецЦикла;
КонецПроцедуры
```

Листинг 3.2.2

Как работает такой цикл? Сначала выполняется внешний цикл, т.е. переменной *«н»* присваивается значение *1*, потом полностью выполняется внутренний цикл, т.е. значение *«к»* проходит от *1* до *7*, после этого управление передается опять на внешний цикл.

Сохраните обработку и посмотрите, что получится.

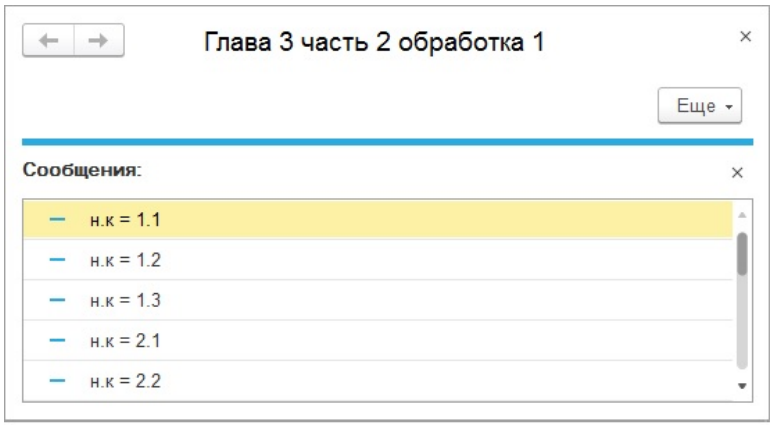


Рис. 3.2.2

Если Вам трудно понять очередность внешнего и внутреннего цикла, поставьте самостоятельно сообщения перед внутренним циклом, и после него, тогда все встанет на свои места.

Значения *Выражение1* и *Выражение2* могут задаваться пользователем в виде переменных.

Сделайте следующий пример:

```
&НаКлиенте
Процедура ВыполнитьКоманду3(Команда)
    Перем Выражение1, Выражение2;

    ВвестиЧисло(Выражение1,"Введите выражение1",2,0);
    ВвестиЧисло(Выражение2,"Введите выражение2 (больше выражение1)",2,0);

    Если Выражение1 > Выражение2 тогда

        Предупреждение("Выражение1 должно быть больше выражение2");

    иначе

        Для н = Выражение1 по Выражение2 цикл

            Сообщить(" н = " + н);

        КонецЦикла;

    КонецЕсли;
КонецПроцедуры
```

Листинг 3.2.3

Разберите его самостоятельно и посмотрите, как он работает.

Продолжим изучение цикла *Для..Цикл.* И углубим нашу схему, введя оператор *Прервать.*

Данный оператор прерывает выполнение цикла в любой его точке. После выполнения данной команды управление передается оператору, следующему после ключевого слова *КонецЦикла*.

С данным оператором конструкция цикла будет иметь следующий вид:

Для ИмяПеременной = Выражение1 по Выражение2 цикл

//операторы

Прервать.

//операторы

КонецЦикла.

Сделайте пример, который нам объяснит использование данного оператора.

```
&НаКлиенте
Процедура ВыполнитьКоманду4(Команда)

    Для н = 1 по 10 цикл
        к = 1/н;
        Сообщить("к = " + Окр(к,3));
        Прервать;
    КонецЦикла;

КонецПроцедуры
```

Листинг 3.2.4

Сохраните обработку и посмотрите, как она работает. Должен быть следующий результат:

Рис. 3.2.3

Как видно из результата выполнения данной обработки, произошел только первый шаг, дальше цикл не стал продолжаться. Как правило, данный оператор ставят внутри определенного условия.

Смотрим еще один пример:

```
&НаКлиенте
Процедура ВыполнитьКоманду5(Команда)

    Для н = - 10 по 10 цикл
        Если н = 0 тогда
            Прервать;
        КонецЕсли;
        к = 1/н;
        Сообщить("к = " + Окр(к,3));
    КонецЦикла;

КонецПроцедуры
```

Листинг 3.2.5

В этом примере наш цикл следует от *-10* до *10*, в том случае, когда счетчик цикла равняется нулю, мы выходим из цикла, чтобы не сгенерировалась исключительная ситуация.

Напоследок, мы разберем оператор *Продолжить*. Еще углубим схему конструкции.

Для ИмяПеременной = Выражение1 по Выражение2 цикл

//операторы

Продолжить;

//операторы

Прервать.

//операторы

КонецЦикла.

Оператор *Продолжить* передает управление в начало цикла, после этого происходит вычисление и проверка условий выполнения цикла. Все, что следует за данным оператором, на этой итерации цикла выполняться не будет.

Операторы Прервать и Продолжить применимы во всех циклах платформы 1С!

Сделайте следующий пример, и посмотрите результат его работы:

```
&НаКлиенте
Процедура ВыполнитьКоманду6(Команда)
    Для н = 1 по 10 цикл
        к = 1/н;
        р = 1/(Pow(н,2));
        Сообщить("1/н   = " + Окр(к,3));
    КонецЦикла;
КонецПроцедуры
```

Листинг 3.2.6

А теперь подправьте наш пример:

```
&НаКлиенте
Процедура ВыполнитьКоманду6(Команда)

    Для н = 1 по 10 цикл
        к = 1/н;
        р = 1/(Pow(н,2));
        Сообщить("1/н   = " + Окр(к,3));
        Продолжить;
        Сообщить("--------------------");
        Сообщить("1/н^2 = " + Окр(р,3));
    КонецЦикла;

КонецПроцедуры
```

Листинг 3.2.7

Выполните обработку. Как видите, сообщение о делении единицы на *«н» в квадрате* не выходит, потому что оператор *Продолжить* передал управление в начало цикла.

Также оператор *Продолжить* можно выполнять внутри условия.

Сделайте следующий пример:

```
&НаКлиенте
Процедура ВыполнитьКоманду7(Команда)

    Для н = - 10 по 10 цикл
        Если н = 0 тогда
            Продолжить;;
        КонецЕсли;
        к = 1/н;
        Сообщить("1/н = " + Окр(к,3));
    КонецЦикла;

КонецПроцедуры
```

Листинг 3.2.8

В данном примере при *«н»* равном нулю, мы не прерываем цикл, как это делали в предыдущем уроке, а просто начинаем его заново, чтобы не сгенерировалась ошибка.

Только что мы с Вами изучили самый простой вид цикла, который есть в языке программирования 1С. Он простой, но основополагающий, научитесь хорошо его понимать и с легкостью им пользоваться. Только после этого приступайте к изучению следующих частей данного урока.

Часть 3. Цикл: Пока…Цикл. Рекурсия переменных

Рассмотрим еще один цикл, это цикл с условием. Принцип его работы мы разберем на следующей схеме:

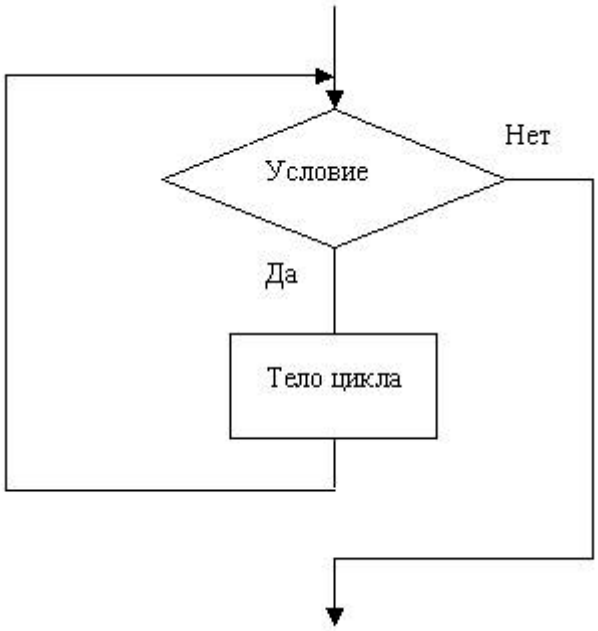


Рис. 3.3.1

Посмотрите на данную схему: перед телом цикла задается какое-нибудь условие, пока данное условие истинно, выполняется тело цикла, в противном случае программа выходит из цикла. Рассмотрим синтаксис данного оператора в программе 1С.

Пока <*Условие*> Цикл

//операторы

Продолжить;

//операторы

Прервать;

КонецЦикла;

Разберем данный оператор. Ключевое слово *Пока* задает некоторое условие, оно должно иметь тип булево, после данного условия следует ключевое слово *Цикл*, открывающее тело цикла, а закрывает тело цикла ключевое слово *КонецЦикла*.

Операторы *Продолжить* и *Прервать* обладают теми же функциями, как и в предыдущем цикле *Для…Цикл*. Это переход в начало цикла и выход из цикла соответственно.

Перед тем, как рассматривать примеры цикла *Пока...Цикл*, мы изучим ***Рекурсию переменных***.

Рекурсия – определение нового значения параметров при помощи старого значения, используя единый идентификатор.

Посмотрим, как это выглядит.

и = 0;

и = и + 1;

В этом случае переменная «и» увеличивает сама себя на 1.

Сделаем самый элементарный пример, и на нем же изучим рекурсию и цикл *Пока…Цикл.* Пример таков: нам необходимо вывести ряд чисел от *0* до *100* с шагом *5 (0,5,10,15 и т.д.)*.

Циклом *Для…Цикл*, который мы изучали в предыдущей части, сделать этот пример можно: нужно при помощи оператора % определять, кратен ли текущий счетчик цикла 5 или нет, и исходя из этого выводить сообщение. Но гораздо проще и понятнее будет сделать данную задачу, используя цикл *Пока...Цикл*.

Создайте самостоятельно обработку, форму и в обработчике команды формы напишем следующий код:

```
&НаКлиенте
Процедура ВыполнитьКоманду1(Команда)
    н = 0;
    Шаг = 5;

    Пока н <= 100 Цикл
        Сообщить("н =" + н);
        н = н + Шаг;
    КонецЦикла;
КонецПроцедуры
```

Листинг 3.3.1

Разберем данный пример. Сначала задаем начальное значение выводимой переменной, потом шаг, с которым она будет выходить на экран.

После этого мы задаем условие цикла: что он будет выполняться, пока переменная *«н»* меньше или равна ста. То есть если переменная *«н»* меньше или равна *100*, то программа будет заходить в тело цикла и выполнять операторы в цикле. Как только данное условие перестанет выполняться, то будут исполняться операторы после ключевого слова *КонецЦикла*. Выводим сообщение чему равно *«н»*. Последний шаг – выполняем рекурсию переменной, увеличивая ее на шаг равный 5.

Сохраните обработку, запустите и посмотрите, как работает данный код.

Рекурсивно можно вычитать, умножать и делить переменные (для типа число). Рекурсивный вызов возможен также для строк и дат.

Еще один пример на рекурсию. Нам надо получить *«н»* факториал.

5! = 1*2*3*4*5

```
&НаКлиенте
Процедура ВычислениеФакторила(Команда)
    Перем н;
    ВвелиЧисло = ВвестиЧисло(н,"Введите число",1,0);

    Если ВвелиЧисло тогда
        Если н > 0 тогда
            НачалоФакториала = 1;
            Итерация = 0;
            Пока Итерация < н цикл
                Итерация = Итерация + 1;
                НачалоФакториала = НачалоФакториала*Итерация;
            КонецЦикла;
            Сообщить(Строка(н) + "! = " +
                        Строка(НачалоФакториала));
        КонецЕсли;
    КонецЕсли;

КонецПроцедуры
```

Листинг 3.3.2

Разберем его. Получаем интерактивно некоторое целое число, проверяем, ввели мы его или нет, записываем введенное число в переменную *«н»*.

Если число ввели, и если оно больше нуля, задаем переменную *НачалоФакториала*, которая равна единице и которая будет рекурсивно увеличиваться, и переменную *Итерация*.

Определяем наш цикл, он будет работать пока переменная *Итерация* меньше переменной *«н»*. Т.е. если мы *«н»* задали равным 5, то цикл будет выполняться при следующих значениях переменной *Итерация* : *0,1,2,3,4.*

Сразу же в начале тела цикла увеличиваем переменную *Итерация* на *1*.

И последний шаг: внутри цикла рекурсивно вызываем переменную *Факториал,* умножая ее на переменную *Итерация*.

Т.е. в первом шаге цикла: *НачалоФакториала = 1 * 1*.

Во втором шаге цикла: *НачалоФакториала = 1 * 2*.

В третьем шаге цикла: *НачалоФакториала = 2 * 3*.

И так далее.

В конце выводим сообщение.

Разберите данный пример, если он Вам не понятен. Разберите его в отладке, посмотрите, как изменяются переменные.

Самостоятельно измените пример так, чтобы *Итерация* начиналась с *1*, поменяйте условие цикла или место рекурсии переменной *Итерация*, чтобы пример работал правильно.

Сделайте еще один пример на цикл *Пока…Цикл*.

Мы имеем две даты, вторая дата старше первой, надо вывести все даты в промежутке между ними.

```
&НаКлиенте
Процедура ВыводДат(Команда)

    Перем ДатаНач, ДатаКон;

    ДатаНачВведена = ВвестиДату(ДатаНач,"Введите начальную дату",
                                ЧастиДаты.Дата);
    ДатаКонВведена = ВвестиДату(ДатаКон,"Введите конечную дату",
                                ЧастиДаты.Дата);

    Если ДатаНачВведена и ДатаКонВведена тогда
        Если ДатаНач < ДатаКон тогда
            Пока ДатаНач <= ДатаКон цикл
                Сообщить(Формат(ДатаНач,"ДЛФ = ДД"));
                ДатаНач = ДатаНач + 3600*24;
            КонецЦикла;
        иначе
            Предупреждение("Начальная дата должна быть раньше конечной
даты");
        КонецЕсли;
    иначе
        Предупреждение("Вы не ввели одну из дат");
    КонецЕсли;

КонецПроцедуры
```

Листинг 3.3.3

Разберем данный пример.

Первым делом мы задаем и интерактивно вводим переменные. Для того чтобы исключить ситуацию, когда пользователь не ввел переменную, мы делаем проверку. Если Вы забыли, то напоминаю, что функции *ВвестиДату, ВвестиСтроку, ВвестиЧисло* возвращают *Истину*, если пользователь ввел строку, число или дату, и возвращают *Ложь*, если нет.

В том случае, если обе переменные *ДатаНачВведена* и *ДатаКонВведена* принимают значение *Истина*, то мы заходим в условие, а если нет, то выдаем пользователю предупреждение.

Следующим шагом мы проверяем то, что дата начальная должна быть всегда меньше даты конечной, иначе также выдаем пользователю сообщение.

После того как проверка прошла, начинает выполняться цикл. В условии цикла мы проверяем то, что дата начальная меньше или равна дате конечной. Если условие истинно, то заходим в тело цикла. В теле цикла мы первым делом выводим на экран начальную дату. А потом - внимание! - мы делаем то, что называется рекурсией переменных (о чем я вам рассказывал в начале этой части главы), но только применимо к датам.

После данных манипуляций, *ДатаНач* увеличивается на один день. Опять проверяем наше условие, и если *ДатаНач* все еще меньше или равно *ДатаКон*, то идем в цикл, иначе выходим из него.

Запустите обработку и посмотрите, как она работает.

На этом мы закончим изучение циклов. Всего в языке программирования 1С существует три вида циклов. Два из них мы изучили в этом уроке. Третий цикл будем изучать в седьмой главе, когда будем проходить универсальные коллекции значений. Пока изучите и научитесь работать с циклом *Пока…Цикл*, изучите и освойте рекурсию переменных. Поскольку в практической работе Вам часто придется сталкиваться с данными вопросами.

Часть 4. Работа с процедурами и функциями

В данной части изучим работу с процедурами и функциями. Что такое процедура?

Процедура - это идентификатор, выполняющий некоторые действия, определенные пользователем или системные. Говоря простым языком: если мы пропишем внутри нашего кода некоторую процедуру (не важно, нами написанную или процедуру программы 1С), то когда выполнение программы дойдет до этой строки, тогда будут выполнены определенные действия. Рассмотрим уже известные нам процедуры *Сообщить* и *Предупреждение*. Из предыдущих примеров Вы видели, что при выполнении процедуры *Сообщить* в окно сообщений выходит текст, введенный в качестве первого параметра данной процедуры. А при выполнении процедуры *Предупреждение* появляется всплывающее окно с описанием текста, заданного в качестве параметра данной процедуры.

Что такое функция?

Функция - это идентификатор, выполняющий некоторые действия, определенные пользователем или системные, и возвращающий значение определенного типа. Как видите, функции отличаются от процедур только тем, что возвращают определенное значение. К примеру, функция *СтрДлина* вычисляет длину строки, которая передается в нее в качестве параметра, и возвращает это значение пользователю в виде числа. Или функция *ТекущаяДата* определяет системную дату компьютера и возвращает это значение в виде даты.

В языке программирования 1С разработчик или внедренец может самостоятельно разрабатывать любые процедуры и функции, которые необходимы для выполнения алгоритма программы.

У многих может возникнуть вопрос: для чего необходимо самим писать процедуры или функции? Во-первых, Вы упростите свой код, если он будет описан с помощью процедур, такой код будет хорошо читаем и понятен для следующего разбора, причем, возможно, и Вами. Во-вторых, чтобы не писать много раз одно и то же в коде. Если у Вас один и тот же код повторяется несколько раз, то лучше его написать в процедуре или функции, а потом просто использовать данную процедуру или функцию в нужных местах. Так Вы не только сделаете хорошо читаемый код, но и подстрахуете себя от возможных ошибок. Понятно, что одно дело исправить ошибку в одном месте внутри процедуры или функции, а другое дело в 10 местах: Вы можете исправить в 9 местах, а в 10-м забыть или пропустить, в результате качество Вашей работы будет страдать.

Это была теоретическая часть, сейчас мы изучим синтаксис процедур в самом простом для начинающих варианте.

```
Процедура ИмяПроцедуры()

//оператор

КонецПроцедуры
```

Разберем этот синтаксис. Ключевое слово *Процедура* идет всегда перед названием. После данного ключевого слова всегда идет название процедуры. Все названия процедур внутри одного

модуля должны быть уникальны. Поскольку в данном примере синтаксиса процедура представлена без параметров (их мы разберем позже), то просто пишем левую и правую скобку вместе. После этих скобок идут операторы, которые описывают те или иные действия внутри процедуры (для удобного чтения я рекомендую начинать писать их со следующей строки и делать отступ с помощью клавиши «*Tab*»). Ключевое слово *КонецПроцедуры* завершает код процедуры. Что бы Вы ни написали после этого ключевого слова, все это к этой процедуре не будет иметь никакого отношения.

Так же в предыдущих листингах Вы должны были обратить внимание на конструкцию *&НаКлиенте*. Это так называемая *директива компиляции*, с её помощью программист может задать, где будет выполняться его код, на сервере или на клиенте. Более подробно работу этой директивы мы разберем в пятой главе, посвященной работе с управляемыми формами. Пока же просто устанавливайте эту директиву перед началом всех процедур и функций, которые Вы используете в модуле формы. На данном этапе обучения Вам не стоит задумываться о её сути.

Создайте пример. В этом примере пользователь вводит два числа, и оба раза вычисляется квадратный корень из данных чисел. Если число меньше нуля, то пользователю выходит целый ряд сообщений. Вывод этого ряда сообщений мы и создадим в процедуре.

Создайте обработку, форму и в модуле формы напишите следующую процедуру:

```
&НаКлиенте
Процедура СообщениеОНеправильномЧисле()
    Сообщить("Введенное число меньше нуля");
    Сообщить("Вычисление не будет произведено");
    Сообщить("Задайте другие числа");
КонецПроцедуры
```

Листинг 3.4.1

А в обработчике команды формы нижеприведенный код:

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)
    Перем А, Б;
    Если ВвестиЧисло(А,"Введите число А") тогда
        Если А >= 0 тогда
            КореньА = Sqrt(А);
            Сообщить("Корень из числа А = " + Окр(КореньА,5));
        иначе
            СообщениеОНеправильномЧисле();
        КонецЕсли;
    КонецЕсли;
    Если ВвестиЧисло(Б,"Введите число Б") тогда
        Если Б >= 0 тогда
            КореньБ = Sqrt(Б);
            Сообщить("Корень из числа Б = " + Окр(КореньБ,5));
        иначе
            СообщениеОНеправильномЧисле();
        КонецЕсли;
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.2

Разберем данный пример.

Мы вводим некоторое первое число *А*, проверяем на то, что оно введено. Я сразу написал функцию *ВвестиЧисло* как параметр оператора *Если*, чтобы оптимизировать код программы. Потом мы проверяем больше или равно нуля наше число, если да - то вычисляем корень, если нет - пишем вновь созданную процедуру, которая выводит целый ряд сообщений. Точно такие же действия проделываем с числом *Б*.

Запустите обработку и посмотрите, как работает данный код.

Сейчас пытливый читатель спросит: ведь эти два отрезка кода, когда вводим первое число и вычисляем корень, а потом второе число и вычисляем корень, очень похожи. Нельзя ли и их каким-нибудь образом сделать в виде процедуры? Да, можно.

Напишите в модуле формы следующую процедуру:

```
&НаКлиенте
Процедура ИнтерактивноВВестиКореньИВычислить()
    Перем А;

    Если ВвестиЧисло(А,"Введите число") тогда
        Если А >= 0 тогда
            КореньА = Sqrt(А);
            Сообщить("Корень из числа " +
                     А + " = " + Окр(КореньА,5));
        иначе
            Сообщить("Введенное число меньше нуля");
            Сообщить("Вычисление не будет произведено");
            Сообщить("Задайте другие числа");
        КонецЕсли;
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.3

А в обработчике команды код:

```
&НаКлиенте
Процедура ВыполнитьКоманду2(Команда)
    ИнтерактивноВВестиКореньИВычислить();
    ИнтерактивноВВестиКореньИВычислить();
КонецПроцедуры
```

Листинг 3.4.4

Посмотрите, код будет выполняться точно так же, как и в предыдущей обработке. Но написание кода в обработчике команды заметно упростилось.

Посмотрев на это, пытливый читатель может задать новый вопрос: а если необходимо одно число задать интерактивно, а одно в программе, как мы поступим в этом случае, как это число передать в процедуру?

Сделать это можно двумя способами. Первый: сделать эту переменную глобальной и передавать в процедуру. О глобальных и локальных переменных мы узнаем в четвертой главе. Либо передать эту переменную в качестве параметра.

А для этого мы рассмотрим наш более усложненный синтаксис. Обратите внимание: все, что касается параметров, справедливо и для процедур, и для функций. Просто сейчас мы рассмотрим данный вопрос на примере процедур.

Процедура ИмяПроцедуры(*Параметр1*, *Параметр2*, ….,*ПараметрN*)

КонецПроцедуры

После названия процедуры внутри скобок идет описание параметров, которые будут передаваться в процедуру. Из основного кода параметры должны передаваться в той последовательности, в которой описаны в процедуре. Названия параметров внутри процедуры должны совпадать с названиями в скобках.

Переделайте Ваш пример.

Измените процедуру *ВычислитьКореньИВывести* следующим образом:

```
&НаКлиенте
Процедура ВычислитьКореньИВывести(ЧислоПодКорнем, НазваниеЧисла)
    Если ЧислоПодКорнем >= 0 тогда
        КореньА = Sqrt(ЧислоПодКорнем);
        Сообщить("Корень из числа " + НазваниеЧисла + " = " +
Окр(КореньА,5));
    иначе
        Сообщить("Число " + НазваниеЧисла + " меньше нуля");
        Сообщить("Вычисление не будет произведено");
        Сообщить("Задайте другие числа");
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.5

И создадим новую команду формы, в обработчике которой напишем следующий код:

```
&НаКлиенте
Процедура ВычислитьКорень(Команда)
    Перем А,Б;

    А = 10.11;

    ВычислитьКореньИВывести(А,"А");

    Если ВвестиЧисло(Б, "Введите число Б") Тогда
        ВычислитьКореньИВывести(Б,"Б");
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.6

Во вновь созданную процедуру мы передаем два параметра: это число, которое будет под корнем, и название данного числа. Внутри процедуры мы совершаем с этими параметрами все необходимые действия. Как видите, внутри процедуры используются те же названия переменных, что и были заданы в скобках.

Переходим к коду, где непосредственно используется наша процедура в обработчике *«ВычислитьКорень»*. Задаем две переменные, одной присваиваем число явно в коде. И передаем эту переменную в процедуру *ВычислитьКореньИВывести*, также вторым параметром передаем название нашего числа, переменная типа строка. Как понятно из примера, эти параметры нельзя путать. Если мы напишем так:

ВычислитьКореньИВывести("А",А);

то мы будем пытаться извлечь квадратный корень из строки, компилятор выдаст ошибку. Очень часто начинающие программисты путают местами параметры, что приводит к различным проблемам.

Еще один момент - это то, что количество параметров заданных в процедуре при разработке, должно соответствовать количеству параметров при использовании данной процедуры в коде.

Т.е. нельзя написать так:

ВычислитьКореньИВывести(А);

Компилятор выдаст ошибку.

С параметрами должно быть понятно, но есть один момент. Когда мы передаем параметр в процедуру, то мы передаем не просто само по себе число, строку или какой-то объект, а, по сути, передаем ссылку на данное число, строку или объект. Это значит, что если внутри процедуры мы изменим данный параметр, то поменяется и сама та переменная, которая передается в виде параметра в нашу процедуру.

Звучит несколько сложно, но один простой пример прояснит картину.

Сделаем новую процедуру *ВычислитьКореньПоМодулюИВывести*:

```
&НаКлиенте
Процедура ВычислитьКореньПоМодулюИВывести(ЧислоПодКорнем, НазваниеЧисла)

    ЧислоПодКорнем = ?(ЧислоПодКорнем >=0,ЧислоПодКорнем,ЧислоПодКорнем*(-
1));
    КореньА = Sqrt(ЧислоПодКорнем);
    Сообщить("Корень из модуля числа " +
             НазваниеЧисла + " = " +
             Окр(КореньА,5));

КонецПроцедуры
```

Листинг 3.4.7

Создадим новую команду *«ВычислитьКореньПоМодулю»* и в её обработчике напишем следующий код:

```
&НаКлиенте
Процедура ВычислитьКореньПоМодулю(Команда)
    Перем А;

    Если ВвестиЧисло(А, "Введите число А") Тогда
        Сообщить("Число А = " + А);
        ВычислитьКореньПоМодулюИВывести(А,"А");
        Сообщить("Число А = " + А);
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.8

В данном примере если *ЧислоПодКорнем* меньше нуля, то мы его рекурсивно умножаем на минус единицу.

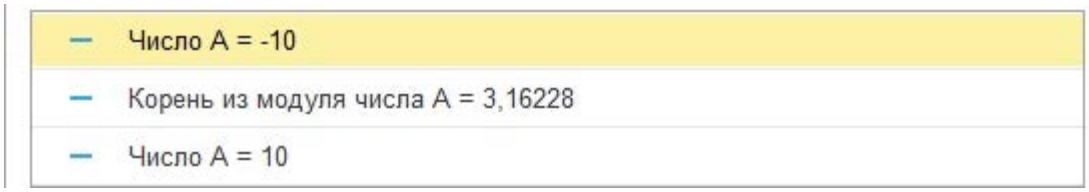


Рис. 3.4.1

Запустите данный пример. И при вводе отрицательного числа будет изменено само число, которое Вы ввели.

В каких-то случаях Вас это не будет сильно волновать, но в других случаях Вы захотите, чтобы переменная, которую передаете в процедуру или функцию, оставалась прежней. Как это сделать? Смотрим следующий синтаксис:

Процедура ИмяПроцедуры(*Знач Параметр1, Знач Параметр2, ….,Знач ПараметрN*)

КонецПроцедуры

Ключевое слово *Знач* перед параметром означает, что данный параметр передается в процедуру по значению, т.е. изменение параметра внутри процедуры никак не повлияет на значение фактического параметра, который был передан в процедуру. Ключевое слово *Знач* является необязательным, т.е. переменная может передаваться как ссылочно, так и по значению.

Теперь измените Ваш предыдущий пример. Подправьте процедуру *«ВычислитьКореньПоМодулюИВывести»*

```
Процедура ВычислитьКореньПоМодулюИВывести(Знач ЧислоПодКорнем, НазваниеЧисла)
    ЧислоПодКорнем = ?(ЧислоПодКорнем >=0,ЧислоПодКорнем,ЧислоПодКорнем*(-1));
    КореньА = Sqrt(ЧислоПодКорнем);
    Сообщить("Корень из модуля числа " +
              НазваниеЧисла + " = " +
              Окр(КореньА,5));
КонецПроцедуры
```

Листинг 3.4.9

Обработчик команды *«ВычислитьКореньПоМодулю»* не изменится.

Запустите обработку, и при вводе отрицательного числа посмотрите, изменится ли само введенное число или нет.

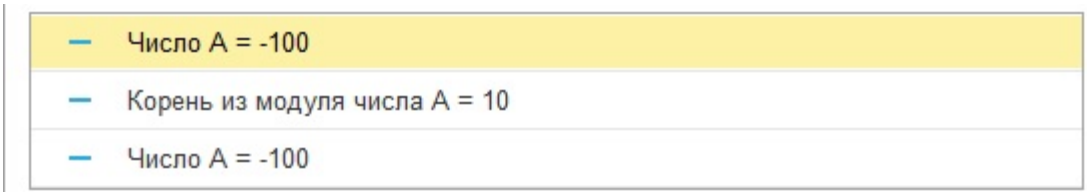


Рис. 3.4.2

Продолжим рассматривать передачу параметров в процедуры и функции. Мы уже знаем, что все параметры, которые мы передаем в процедуру или функцию, обязательны, т.е. когда Вы в коде задаете процедуру, то обязаны перечислить их все, но есть возможность создать параметр по умолчанию, тогда у Вас не будет обязанности перечислять его в процедуре.

Рассмотрим синтаксис.

Процедура ИмяПроцедуры(*Параметр1 = Умол1, Параметр2 = Умол2, ….,ПараметрN = УмолN)*

КонецПроцедуры

Разберем данный синтаксис. Такая форма описания параметра означает, что данный параметр имеет значение по умолчанию и тем самым он становится необязательным, т.е. когда мы будем писать данную процедуру в модуле, то вольны не указывать этот параметр.

Доработайте пример. В этот раз мы вычисляем корень числа и выводим его на экран с точностью до 5-го знака, но пользователь может менять точность.

Создадим новую процедуру *«ВычислитьКореньИВывестиСОкруглением»*

```
&НаКлиенте
Процедура ВычислитьКореньИВывестиСОкруглением(Знач ЧислоПодКорнем,
НазваниеЧисла, Точность = 5)

    ЧислоПодКорнем = ?(ЧислоПодКорнем >=0,ЧислоПодКорнем,ЧислоПодКорнем*(-
1));
    КореньА = Окр(Sqrt(ЧислоПодКорнем),Точность);
    Сообщить("Корень из модуля числа " + НазваниеЧисла + " = " + КореньА);

КонецПроцедуры
```

Листинг 3.4.10

Опять создадим новую команду «*ВычислитьКореньСОкруглением»*, и в её обработчике напишем следующий код:

```
&НаКлиенте
Процедура ВычислитьКореньСОкруглением(Команда)
    Перем А;

    Если ВвестиЧисло(А, "Введите число А") Тогда
        ВычислитьКореньИВывестиСОкруглением(А,"А");
        ВычислитьКореньИВывестиСОкруглением(А,"А",7);
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.11

Разберем данный пример. Мы переделали нашу процедуру *ВычислитьКореньСОкруглением*, добавив параметр *Точность*, который по умолчанию равен *5*. Его мы используем для округления числа, которое получилось при извлечении квадратного корня. После этого мы используем процедуру два раза. Первый раз без третьего параметра, а второй раз с третьим параметром.

Сохраните обработку и запустите ее. И введите любое число.

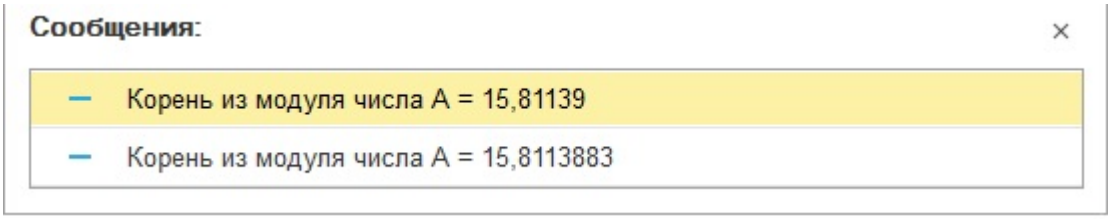


Рис. 3.4.3

Как Вы видите, в первый раз значение вышло с точностью до 5-го знака, а второй до 7-го.

Обращаю Ваше внимание, что параметр, который передается по значению, также может быть и параметром по умолчанию.

Рассмотрим следующий синтаксис:

Процедура ИмяПроцедуры(*Знач Параметр1 = Умол1, Знач Параметр2 = Умол2, …., Знач ПараметрN = УмолN*)

КонецПроцедуры

Данный синтаксис нам говорит, что параметр по умолчанию будет передаваться по значению. Сделайте следующий пример.

В этом примере, если *Точность* будет меньше *5*, то будем выводить все равно с точностью равной *5*.

Создадим новую процедуру *«ВычислитьКореньИВывестиТочностьПять»*:

```
&НаКлиенте
Процедура ВычислитьКореньИВывестиТочностьПять(Знач ЧислоПодКорнем,
НазваниеЧисла, Знач Точность = 5)

    ЧислоПодКорнем = ?(ЧислоПодКорнем >=0,ЧислоПодКорнем,ЧислоПодКорнем*(-
1));
    Точность = ?(Точность < 5, 5, Точность);
    КореньА = Sqrt(ЧислоПодКорнем);
    Сообщить("Корень из модуля числа " +
             НазваниеЧисла + " = " +
             Окр(КореньА,Точность));

КонецПроцедуры
```

Листинг 3.4.12

Опять создадим новую команду «*ВычислитьКореньСТочностьюПять*», и в её обработчике напишем следующий код:

```
&НаКлиенте
Процедура ВычислитьКореньСТочностьюПять(Команда)

    Перем А;

    Если ВвестиЧисло(А, "Введите число А") Тогда
        ВычислитьКореньИВывестиТочностьПять(А,"А");
        ВычислитьКореньИВывестиТочностьПять(А,"А",3);
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.13

Разберите данный пример самостоятельно.

Рассмотрим последний оператор, который применим просто к процедурам, а не к их параметрам. Это оператор *Возврат*.

Рассмотрим синтаксис.

Процедура ИмяПроцедуры(*Параметр1*, *Параметр2*, ….,*ПараметрN*)

///операторы

Возврат;

//операторы

КонецПроцедуры

В данном синтаксисе, параметры могут задаваться каким угодно из вышеперечисленных способов.

После того, как мы написали имя процедуры, перечислили параметры, пишем наши действия. И когда в ходе выполнения процедуры встретится ключевое слово *Возврат*, то программа тут же выйдет из нее и ее выполнение прекратится. После этого действия работать будут операторы, следующие после ключевого слова *КонецПроцедуры*.

Сделайте пример. В этом примере Вы также будете вычислять квадратный корень числа, но если число отрицательное, то будет выполнен выход из процедуры (см. рис. 3.4.4).

Создадим процедуру *«ВычислитьКореньТолькоПоложительныхЧисла»*:

```
&НаКлиенте
Процедура ВычислитьКореньТолькоПоложительныхЧисла(ЧислоПодКорнем,
НазваниеЧисла)
    Если ЧислоПодКорнем < 0 тогда
        Предупреждение("Нельзя извлекать квадратный корень из отрицательного
числа!");
        Возврат;
    КонецЕсли;
    КореньА = Sqrt(ЧислоПодКорнем);
    Сообщить("Корень из числа " + НазваниеЧисла + " = " + Окр(КореньА,5));
КонецПроцедуры
```

Листинг 3.4.14

Опять создадим новую команду «*ВычислитьКореньПоложительногоЧисла*» и в её обработчике напишем следующий код:

```
&НаКлиенте
Процедура ВычислитьКореньПоложительногоЧисла(Команда)

    Перем А;
    Если ВвестиЧисло(А, "Введите число А") Тогда
        ВычислитьКореньТолькоПоложительныхЧислел(А,"А");
    КонецЕсли;

КонецПроцедуры
```

Листинг 3.4.15

Разберем данный пример. В самом начале процедуры мы проверяем, меньше ли нуля параметр *ЧислоПодКорнем* или нет. Если меньше нуля, то выводим предупреждение и посредством оператора *Возврат* выходим из процедуры.

Сохраните и перезапустите Вашу обработку.

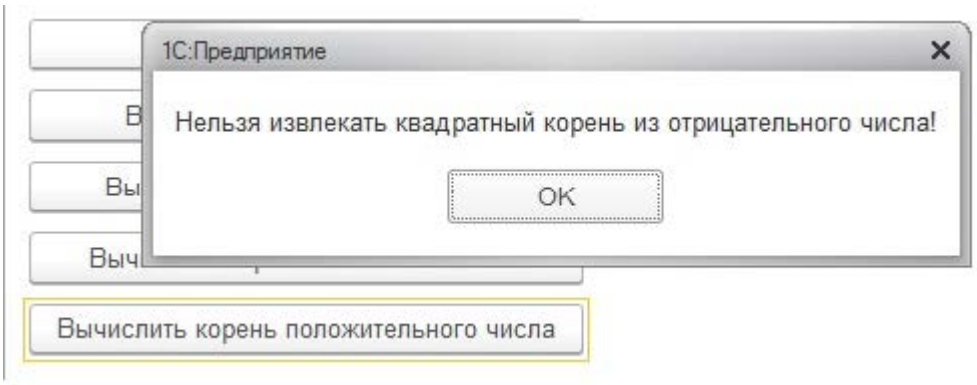


Рис. 3.4.4

Обращаю Ваше внимание: что бы Вы ни написали после слова *Возврат*, все это выполняться уже не будет. Теперь самостоятельно напишите предупреждение после ключевого слова *Возврат* и посмотрите, что получится.

Только что мы изучили с Вами работу с процедурами в языке программирования 1С. Теперь Вы самостоятельно сможете создавать любые процедуры, какие захотите.

Рассмотрим синтаксис написания функции. Как я уже рассказывал, все, что мы изучили касательно параметров на примере процедур, абсолютно справедливо и для функций.

Итак, синтаксис функций.

Функция ИмяФункции(*<Параметры>)*

//операторы

Возврат ВозвращаемоеЗначение;

КонецФункции

Рассмотрим данный синтаксис. Ключевое слово *Функция* идет всегда перед названием функции. Поскольку все параметры мы разобрали на примере процедуры, не будем подробно останавливаться на этом. После параметров идут операторы, которые описывают те или иные действия внутри функции (для удобного чтения кода я рекомендую начинать писать их со следующей строки и делать отступ с помощью клавиши *«Tab»*). Основное отличие функции от процедуры в том, что ключевое слово *Возврат* не только завершает выполнение функции, но и возвращает определенное значение в то выражение, где использовалась данная функция. Ключевое слово *КонецФункции* завершает код функции. Как и в случае процедуры, все, что Вы написали после данного слова, не будет иметь к вашей функции никакого отношения.

Рассмотрим пример. Напишем функцию *Сигнум*, параметром которой является некоторое число, и она возвращает *1*, если число больше нуля; ноль, если число равно нулю; и *-1*, если число меньше нуля.

Создайте в модуле формы следующую функцию:

```
&НаКлиенте
Функция Сигнум(ЧислоА)
    Перем Сигн;

    Если ЧислоА > 0 тогда
        Сигн = 1;
    ИначеЕсли ЧислоА = 0 тогда
        Сигн = 0
    ИначеЕсли ЧислоА < 0 тогда
        сигн = -1
    КонецЕсли;

    Возврат Сигн;
КонецФункции
```

Листинг 3.4.16

Опять создадим новую команду «*ВычислитьСигнумЧисла*» и в её обработчике напишем следующий код:

```
&НаКлиенте
Процедура ВычислитьСигнумЧисла(Команда)
    Перем А;
    Если ВвестиЧисло(А, "Введите число А") Тогда
        В = Сигнум(А);
        Сообщить("Signum (" + А + ") = " + В);
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.17

Разберем данный код.

В модуле формы мы создали функцию, которая называется *Сигнум*, указываем, что у нее только один параметр, назовем его *ЧислоА*. После этого задаем переменную внутри функции, которую мы будем возвращать, назовем ее *Сигн*. Делаем сравнение, в результате которого *Сигн* становится равен либо единице, либо нулю, либо минус единице.

Смотрим основной код в обработчике команды *«ВычислитьСигнумЧисла»*. Первым делом интерактивно вводим некоторое число *А*. В следующей строке используем Вашу вновь созданную функцию: передаем в нее переменную *А*, а возвращать она будет переменную *В*, значение которой выводим в сообщении..

Сохраните обработку и посмотрите, как работает Ваша функция.

Обращаю ваше внимание, что необязательно писать один *Возврат* и только в конце функции, вполне может быть и такой вид функции:

```
&НаКлиенте
Функция л_Сигнум(ЧислоА)

    Если ЧислоА > 0 тогда
        Возврат 1;
    ИначеЕсли ЧислоА = 0 тогда
        Возврат 0;
    Иначе
        Возврат -1
    КонецЕсли;

КонецФункции
```

Листинг 3.4.18

Как видно из данного примера, после каждого результата сравнения мы возвращали нужное нам значение.

Еще одной особенностью функций является то, что необязательно какой-то переменной присваивать значение функции. Эту функцию можно использовать в качестве процедуры.

Сделаем пример, с помощью которого Вы поймете данный тезис. В этом примере мы будем вычислять квадратный корень из числа и возвращать значение в виде параметра, а сама функция будет возвращать *Истина*, если удалось вычислить, и *Ложь*, если нет.

Напишите в модуле формы следующую функцию:

```
Функция ВычислитьКвадратныйКорень(НазваниеЧислаПодКорнем, ЧислоПодКорнем,
КореньЧисла)

    Если ЧислоПодКорнем < 0 тогда
        Сообщить("Нельзя вычислять корень из отрицательного числа");
        Возврат Ложь;
    иначе
        КореньЧисла = Окр(Sqrt(ЧислоПодКорнем),5);
        Сообщить("Корень из числа " + НазваниеЧислаПодКорнем + " = " +
КореньЧисла);
        Возврат Истина;
    КонецЕсли;

КонецФункции
```

Листинг 3.4.19

Опять создадим новую команду «*ВычислитьКореньЧислаФункцией*» и в её обработчике напишем следующий код:

```
&НаКлиенте
Процедура ВычислитьКореньЧислаФункцией(Команда)
    Перем А,Б;

    Если ВвестиЧисло(А, "Введите число А") Тогда
        ВычислитьКвадратныйКорень("А",А,Б);
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.20

Разберем данный пример. Наша функция имеет три параметра: название числа под корнем, само число, которое будет под корнем, и корень из этого числа. Если число под корнем меньше нуля, то мы возвращаем *Ложь*, соответствующее сообщение и ничего не считаем, а если нет - то возвращаем *Истина*, вычисляем число и выводим соответствующее сообщение.

В основной процедуре мы задаем данную функцию в виде процедуры. Поскольку нам в этом случае не интересно, что она вернет, нам важно знать, какое будет число *Б*.

К примеру, мы можем использовать то, что возвращает эта функция для дальнейшей работы с числом *Б*, по аналогии как мы используем функцию *ВвестиЧисло*. Теперь сами измените данный пример и возведите в квадрат полученное число *Б*, если оно было вычислено.

На этом мы закончим нашу интересную и огромную часть про функции и процедуры. Теперь Вы самостоятельно сможете создавать любые процедуры и функции, которые Вам необходимы. В этой части мы изучили почти все, что необходимо для дальнейшей работы.

Остался только момент про экспорт процедур и функции. Этот вопрос мы будем проходить в четвертой главе.

Часть 5. Попытка…Исключение

В заключительной части третьей главы мы рассмотрим очень нужный и полезный оператор ***Попытка… Исключение***. В чем суть этого оператора? Как Вы уже заметили из предыдущих примеров, в ходе выполнения программы могут возникать ошибки, такие как деление на ноль, корень из отрицательного числа и прочие. Мы эти ошибки обходили конструкцией *Если...Тогда*, но не всегда возможно применить данную конструкцию, чтобы обойти исключительную ситуацию. Поэтому в языке программирования 1С существует оператор *Попытка…Исключение.*

Рассмотрим его синтаксис.

```
Попытка
//операторы попытки
Исключение
//операторы исключения.
КонецПопытки
```

Разберем данный синтаксис.

Ключевое слово *Попытка* открывает список операторов, выполнение которых может привести к исключительной ситуации, все операторы между ключевыми словами *Попытка* и *Исключение* это *Операторы попытки*.

Ключевое слово *Исключение* открывает список операторов, которые будут выполняться в том случае, если вызвана исключительная ситуация операторами попытки. Т.е. если во время выполнения программы один из операторов попытки вызвал ошибку выполнения программы (исключительную ситуацию), то выполнение данного оператора прерывается и управление передается на первый *Оператор исключения*. Причем обращаю Ваше внимание, что управление будет передано и в том случае, если исключительную ситуацию вызвали функции и процедуры, разработанные программистом и применяемые в качестве операторов попытки. Естественно, что если ошибка произошла в процедуре или функции, то ее выполнение будет прервано.

Создайте обработку, форму, команду на форме и в обработчике команды сделайте следующий пример:

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)
    Перем А;

    Если ВвестиЧисло(А,"Введите число А") тогда
        Попытка
            КореньЧислаяА = Sqrt(А);
            Сообщить("Квадратный корень числа А = " + Окр(КореньЧислаяА,5));
        Исключение
            Сообщить("Попытка извлечь корень из отрицательного числа");
        КонецПопытки;
    КонецЕсли;

КонецПроцедуры
```

Листинг 3.5.1

Разберем данный пример.

Как обычно, вводим интерактивно число, после чего пытаемся вычислить квадратный корень данного числа. Используем ключевые слова *Попытка, Исключение, КонецПопытки*. Эти ключевые слова всегда должны быть в наличии и представлены в такой очередности, как мы видим. Таким образом, в Вашем примере вычисление квадратного корня и вывод сообщения являются операторами попытки. Между словом *Исключение* и *КонецПопытки* располагаются операторы исключения. На эти операторы перейдет программа после вызова исключения. Операторы попытки, следующие после строки, когда, где была инициализирована ошибка, исполняться не будут.

Запустите вновь созданную обработку и посмотрите, как она работает.

Если Вы все сделали правильно, то увидите, что никакой ошибки не возникает, просто вышло сообщение о корне из отрицательного числа. А что если нам нужно, чтобы все-таки ошибка вышла? Для этого есть оператор ***ВызватьИсключение***.

Переделайте Ваш пример.

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)
    Перем А;

    Если ВвестиЧисло(А,"Введите число А") тогда
        Попытка
            КореньЧислаяА = Sqrt(А);
            Сообщить("Квадратный корень числа А = " + Окр(КореньЧислаяА,5));
        Исключение
            Сообщить("Попытка извлечь корень из отрицательного числа");
            ВызватьИсключение;
        КонецПопытки;
    КонецЕсли;

КонецПроцедуры
```

Листинг 3.5.2

В данном примере после операторов исключения пишем оператор *ВызватьИсключение*.

Сохраните, запустите и введите отрицательное число.

Должно выйти следующее окно:

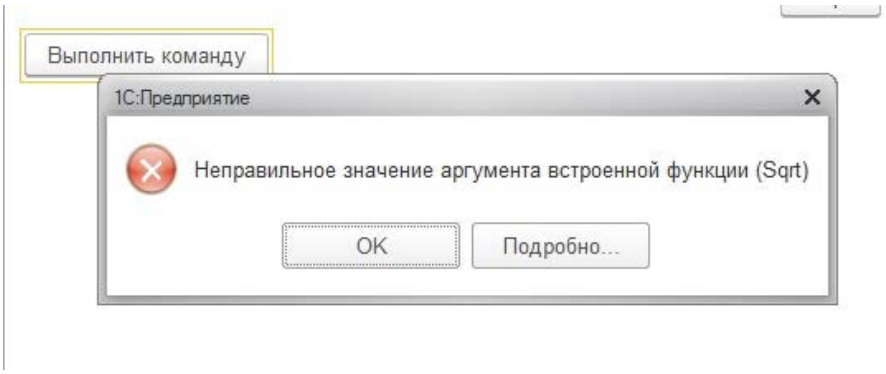


Рис. 3.5.1

Как видите, оператор исключения отработал, и сразу вышла ошибка о неправильном параметре.

Оператор *ВызватьИсключение* особенно удобно использовать на этапе отладки, когда исключительная ситуация может сложиться в различных случаях и необходимо точно знать, где она произошла и в результате чего она произошла.

Теперь сделаем пример, когда исключительная ситуация будет происходить внутри функции, которая используется в качестве оператора попытки.

Данный пример будет следующим: вычислим квадратный корень некоторого числа с округлением, потом возведем в квадрат и посчитаем погрешность: вычтем из заданного числа получившийся квадрат.

Напишите в модуле формы следующую функцию:

```
&НаКлиенте
Функция ВычеслитьИОкруглитьКорень(ЧислоПодКорнем,Точность = 5)

    КореньЧисла = Окр(Sqrt(ЧислоПодКорнем),Точность);

    Возврат КореньЧисла;

КонецФункции
```

Листинг 3.5.3

Создадим команду *«ПолучитьПогрешностьОкругления»* и в обработчике команды напишем следующий код:

```
&НаКлиенте
Процедура ПолучитьПогрешностьОкругления(Команда)
    Перем А;

    Если ВвестиЧисло(А,"Введите число А") тогда
        Попытка
            А2 = Pow(ВычеслитьИОкруглитьКорень(А,10),2);
```

```
            ПогрешностьОкругления = А - А2;
            Сообщить("Погрешность окуругления = " + ПогрешностьОкругления);
        Исключение
            Сообщить("Ошибка при вычислении");
        КонецПопытки;
    КонецЕсли;

КонецПроцедуры
```

Листинг 3.5.4

Сохраните и запустите Вашу обработку.

Если сделано все правильно, то Вы увидите, что при введении положительного числа выходит результат, а в случае отрицательного числа – сообщение об ошибке.

Разберите данный пример самостоятельно.

Сейчас мы рассмотрели оператор *Попытка…Исключение*. Теперь Вы сможете избегать неприятных ситуаций с возникновением ошибок. Особенно желательно использовать данный оператор в критических моментах выполнения кода, например, при записи чего-либо в базу.

Резюме

На этом третья глава закончилась. В ней Вы научились работать с условиями, а также освоили два из трех операторов цикла. После данного урока Вы сможете создавать собственные процедуры и функции, а также использовать оператор *Попытка…Исключение* в своей практике программирования. Удачи в изучении языка!

Глава 4. Основы конфигурирования

В предыдущих главах мы освоили элементарные азы программирования, научились работать с примитивными переменными (строка, число, дата и булево), изучили циклы, условия, функции, процедуры и попытки. Это все те основы, которые Вам необходимы для старта в интересную и, главное, денежную профессию программиста 1С.

В этой главе мы рассмотрим только основы конфигурирования программ 1С, поскольку доскональное изучение всех метаданных не является целью данной книги. А цель данной книги - научить Вас хорошо программировать, рассказать об основных объектах языка программирования и обучить основным приемам работы со встроенным языком 1С. Поэтому мы не будем сильно углубляться в конфигурирование, а изучим только основные понятия, которые нам будут необходимы для дальнейшего изучения языка программирования 1С.

Часть 1. Основы конфигурации

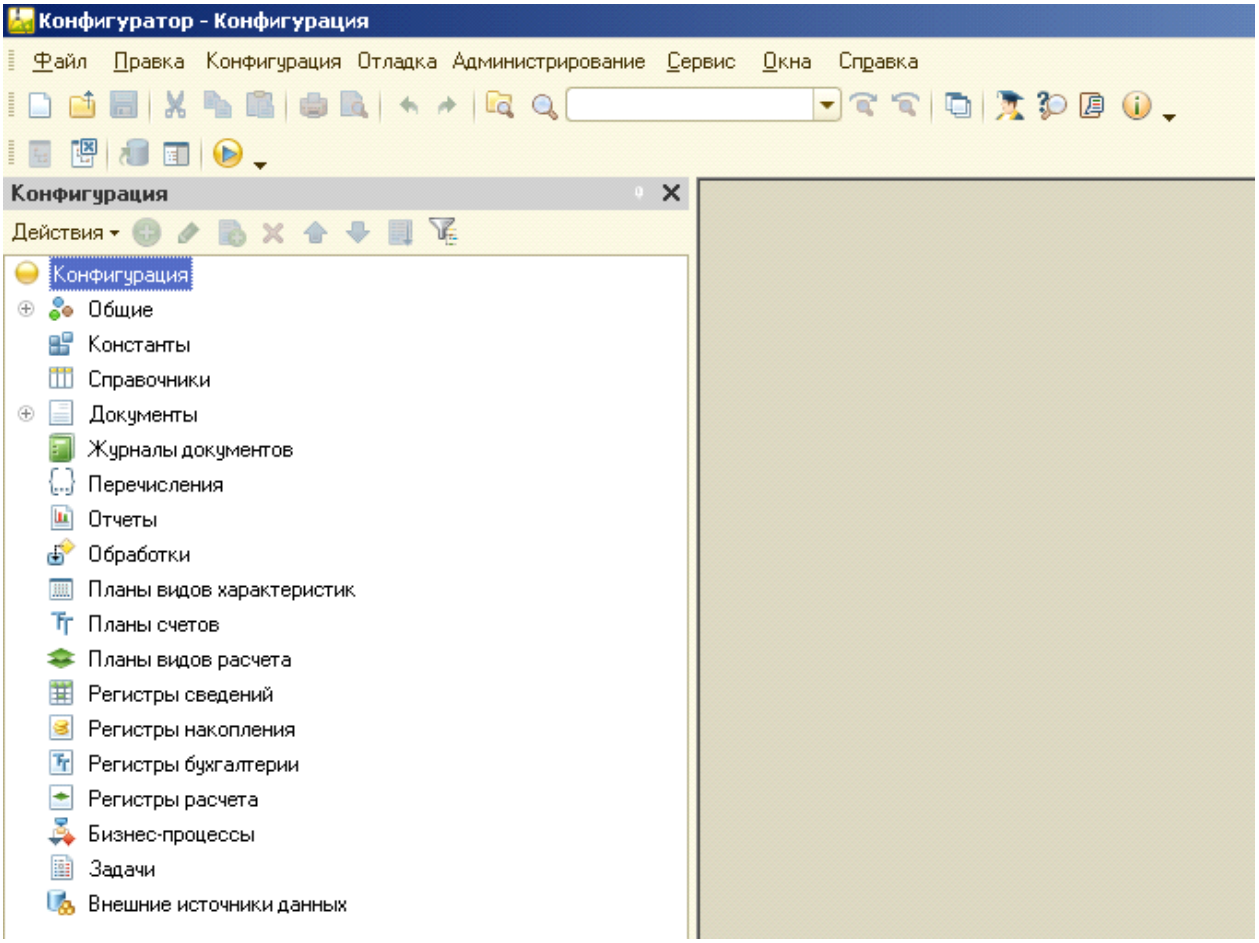


Рис. 4.1.1

Итак, первая часть нашей главы - это основы конфигурации. Что такое *Конфигурация*? ***Конфигурация*** - это единство различных взаимосвязанных составных частей (учетные данные, формы вывода, печатные формы, роли, интерфейсы и пр.), которое является моделью предметной области, необходимой для реализации прикладных функций, для которых разрабатывалось данное программное обеспечение.

Например, есть некоторая прикладная задача (а «1С:Предприятие» создано только для реализации прикладных задач), суть которой - это автоматизация бухгалтерского учета, или торгового учета, или любого другого учета, необходимого пользователю. Под эту задачу разрабатывается конфигурация, которая реализовывает в 1С модель данного учета.

К примеру, если в учете требуется хранить информацию об автомобилях, то разрабатывается справочник *«Автомобили»*, реквизитами которого являются марки автомобилей, вид автомобиля, его госномер и т.п. Этот справочник отображает реальный предмет учета. Так и во всем остальном.

На рисунке 4.1.1. приведена пустая конфигурация. Такая конфигурация бывает обычно перед началом какой-либо разработки. В реальном случае любая конфигурация всегда содержит какие-нибудь *Метаданные*. Что такое *Метаданные*? ***Метаданные*** - это объекты конфигурации. К примеру, конкретный справочник *«Автомобили»* это объект метаданных. Документ *«Счет на оплату»* – тоже объект метаданных.

У каждого объекта метаданных есть свой *прототип*, который определяет основные свойства и методы этого объекта метаданных. Как вы заметили из рисунка 4.1.1, конфигурация имеет древовидную структуру, где сама конфигурация это, по сути, корень дерева, а ветки «Справочники», «Документы», «Константы» и т.д. - это прототипы метаданных. Соответственно в рамках нужных прототипов и создаются определенные объекты метаданных. Например, в ветке «Справочники» будут созданы все справочники, т.е. объекты, необходимые для хранения постоянной или условной информации. У всех объектов, созданных в ветке «Справочники», будет примерно одинаковый набор свойств и методов (в зависимости от параметров конкретного объекта метаданных).

В этой части, для примера, мы разработаем небольшую конфигурацию по учету автотранспорта.

А для этого нам потребуется изучить основные прототипы *метаданных*.

Константы

И первый прототип, который мы изучим, это - *Константы*.

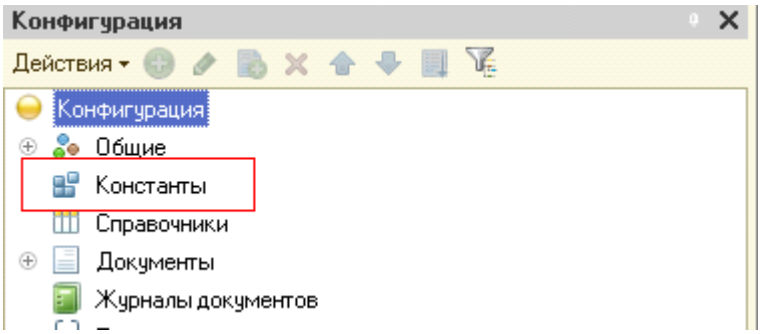


Рис. 4.1.2

Если Вы изучали другие языки программирования, то у Вас будет легкое недоумение от того, что константы в 1С не несут в себе такую же функциональную нагрузку, как в языках *Паскаль, С* и т.п. В конфигурации 1С ***константы*** хранят в себе постоянную или условно постоянную информацию, которая или никогда не изменяется, или изменяется очень редко. Это может быть, к примеру, название организации или ИНН организации и т.п. Основное отличие констант 1С в том, что пользователь в принципе может изменять данные константы по своему усмотрению. Для работы с константами в конфигурации имеется ветвь *«Константы»*. Она выделена на рисунке 4.1.2.

Для того чтобы добавить константу, необходимо щелкнуть по ней правой кнопкой мышки и выбрать пункт "*Добавить*".

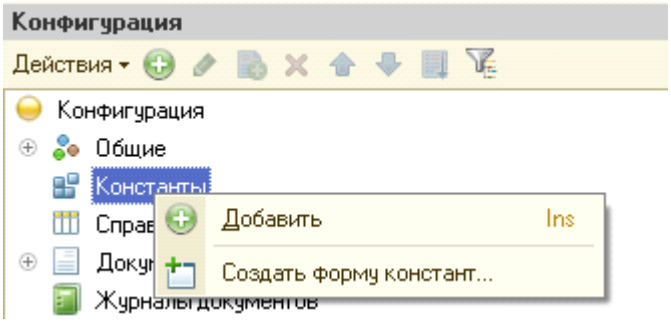


Рис. 4.1.3

Откроется окно *"Свойства константы"* (или палитра свойств констант).

Рис. 4.1.4

В этой форме вводим *Имя константы* (назовем ее *НазваниеОрганизации*) и *Синоним* – аналог имени, который будет отображаться пользователю. *Тип константы* (мы пока знаем только 4 типа) - выберем тип *Строка*. *Длина* - это длина строки, сделаем ее равной ста.

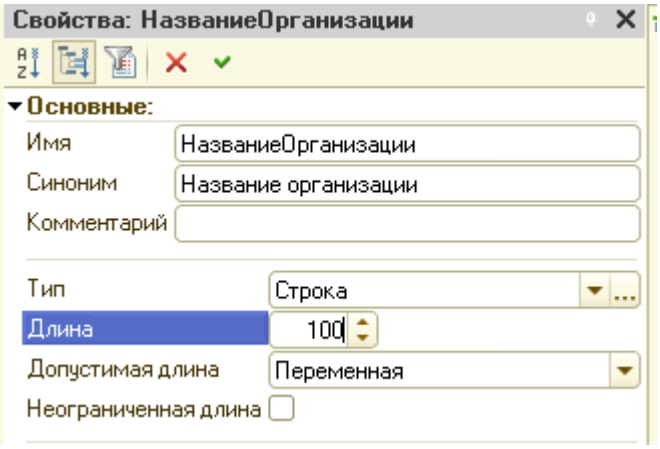


Рис. 4.1.5

Допустимая длина может быть либо фиксированной, либо переменной. Переменная отличается от фиксированной тем, что если в первом случае Вы ввели при длине 100 символов строку из 20 символов, то в базе будет храниться строка из 20 символов, а в фиксированном случае будут храниться все 100 символов, остальные 80 заполнятся пробелами.

Вот мы создали первую константу. Создайте самостоятельно следующие константы: *ДатаНачалаРаботыОрганизации* (тип *Дата*, состав даты *Дата*, см. рис. 4.1.6), *ПроцентНалогаНаПрибыль* (тип *Число* с точностью до одной сотой).

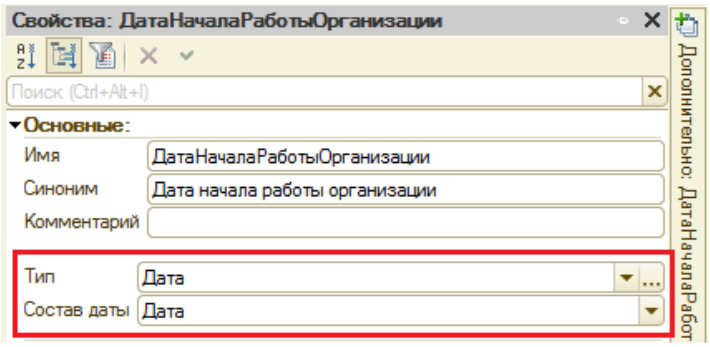


Рис. 4.1.6

В итоге у Вас должен получиться следующий набор констант:

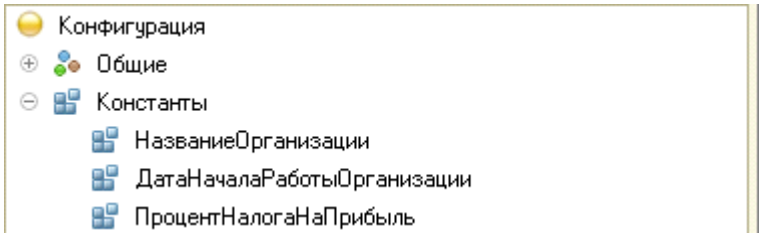


Рис. 4.1.7

Для констант можно создать собственную форму.

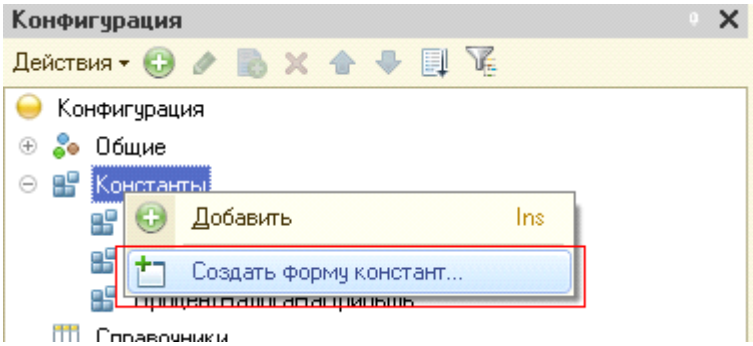


Рис. 4.1.8

В открывшемся конструкторе формы оставляем все без изменения и нажимаем кнопку *«Готово»*. Вышла форма констант, которая появилась в ветке *«Общие»-«Общие формы»*.

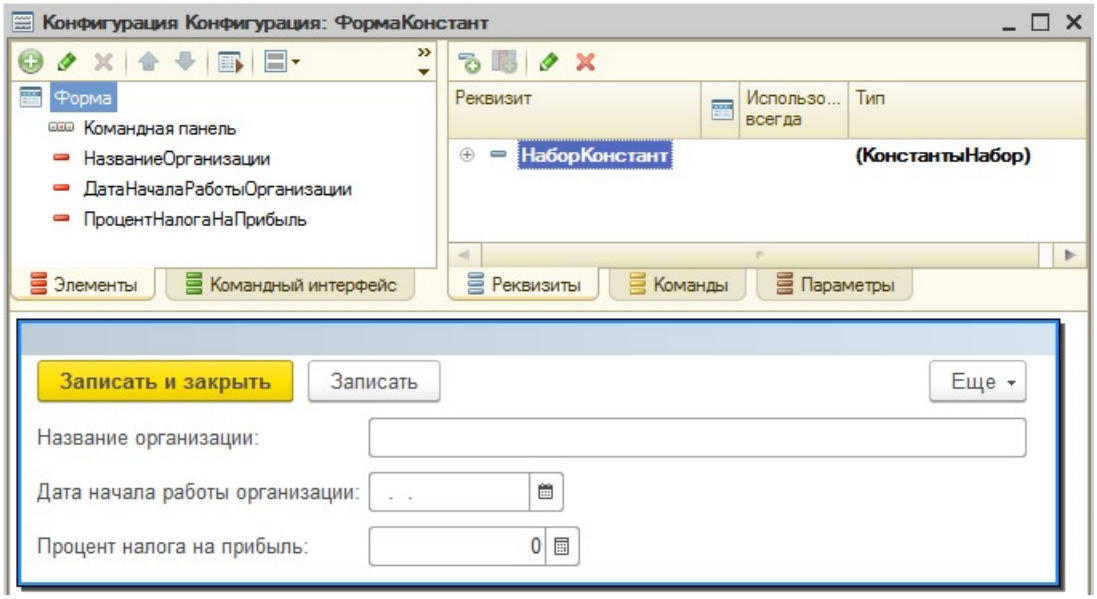


Рис. 4.1.9

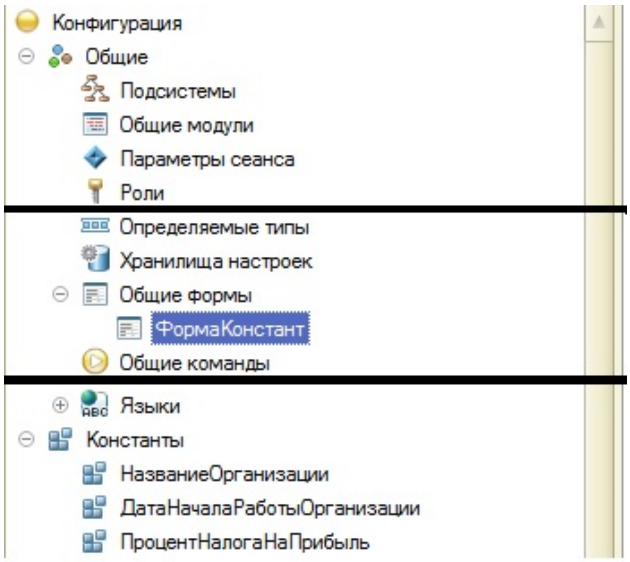


Рис. 4.1.10

Более подробно работу с формами мы рассмотрим в пятой и шестой главе.

После того, как Вы создадите константы и форму констант, у окна конфигурации появится знак * (см. рис. 4.1.11), это значит, что Ваша конфигурация отредактирована, но не сохранена. Чтобы ее сохранить, нужно нажать на кнопку «Сохранить» (см. рис. 4.1.12).

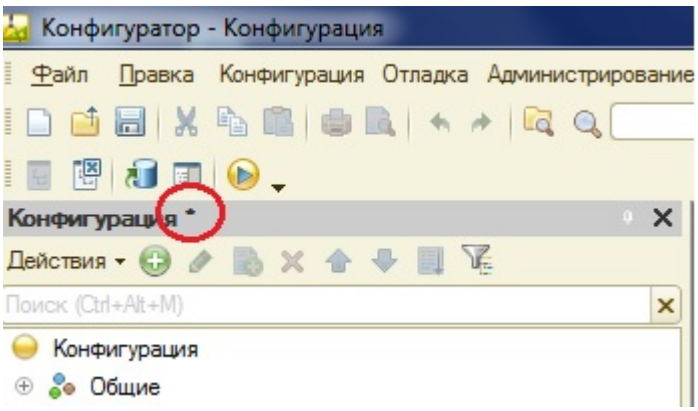


Рис. 4.1.11

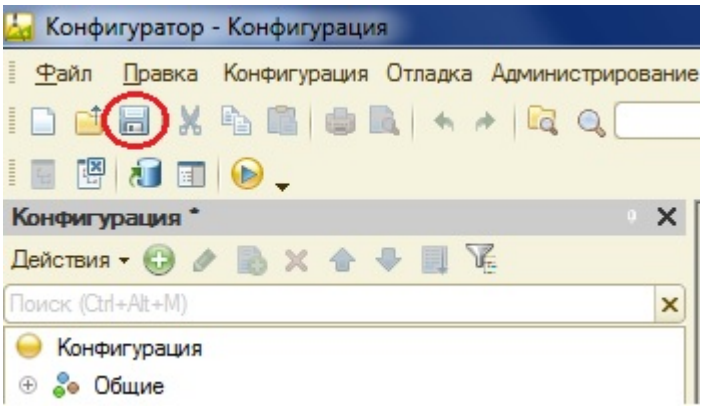


Рис. 4.1.12

После сохранения, у окна конфигурации появится знак «!» (см. рис. 4.1.13). Это значит, что Ваша конфигурация сохранена, но не загружена в базу.

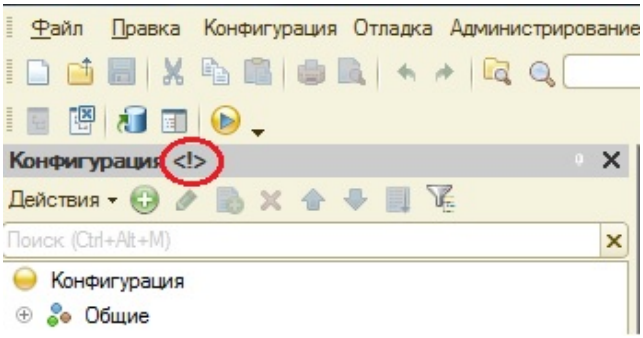


Рис. 4.1.13

Почему так? Сделаем небольшое отступление. В базе 1С хранятся две конфигурации (даже три, если база на поддержке). Это конфигурация базы данных, т.е. те наборы метаданных, с которыми непосредственно работает пользователь. И просто конфигурация, с которой работает разработчик, назовем ее «конфигурация разработчика». Разработчик (программист) не может напрямую менять конфигурацию базы данных. Он работает только с конфигурацией разработчика. После того, как программист внесет все изменения, он может или обновить конфигурацию базы данных, нажав на соответствующую кнопку (см. рис. 4.1.13), или вернуться к конфигурации базы данных, нажав на пункт меню «Вернуться к конфигурации базы данных» (см. рис. 4.1.14), тогда все его изменения затрутся, и конфигурация разработчика будет идентична конфигурации базы данных.

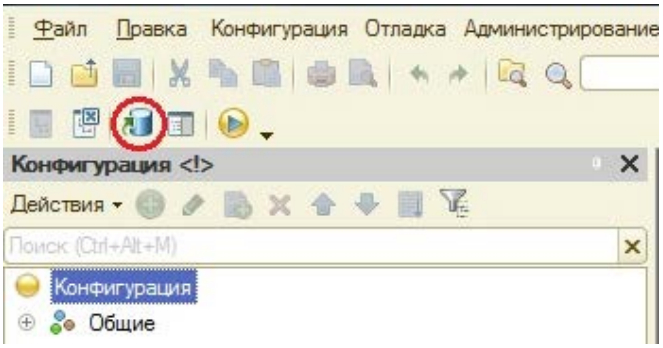


Рис. 4.1.14

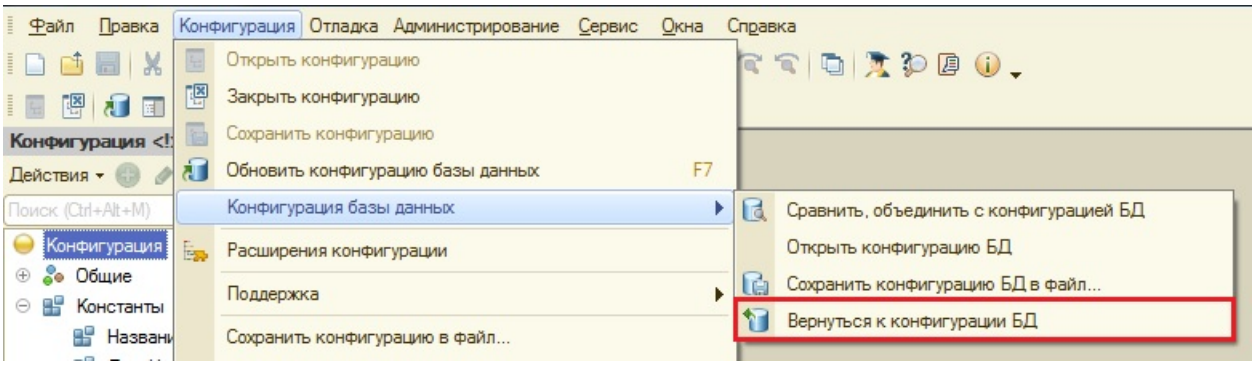


Рис. 4.1.15

Мы обновим базу данных (рис. 4.1.14) и запустим «1С:Предприятие», нажав на кнопку «Начать отладку» (см. рис. 4.1.16).

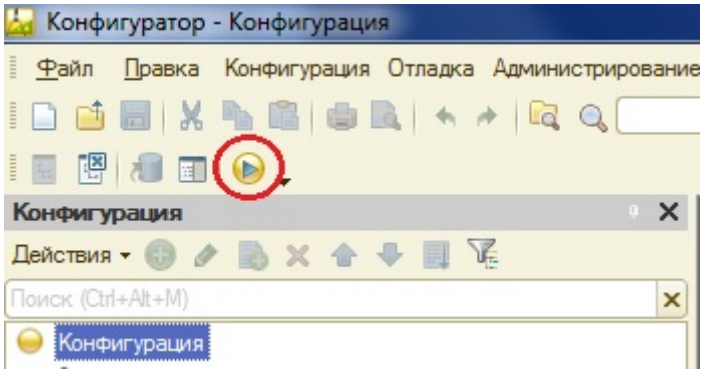


Рис. 4.1.16

В верхней панели у Вас появится меню «Сервис», раскрыв которое, Вы увидите пункт «Форма констант» (см. рис. 4.1.17).

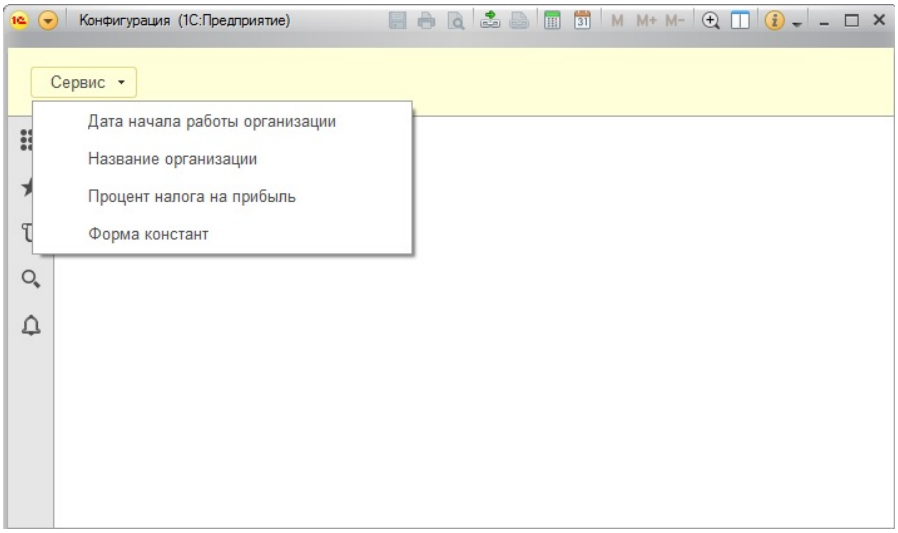


Рис. 4.1.17

После нажатия на этот пункт откроется форма, которую мы создали ранее (см. рис. 4.1.18).

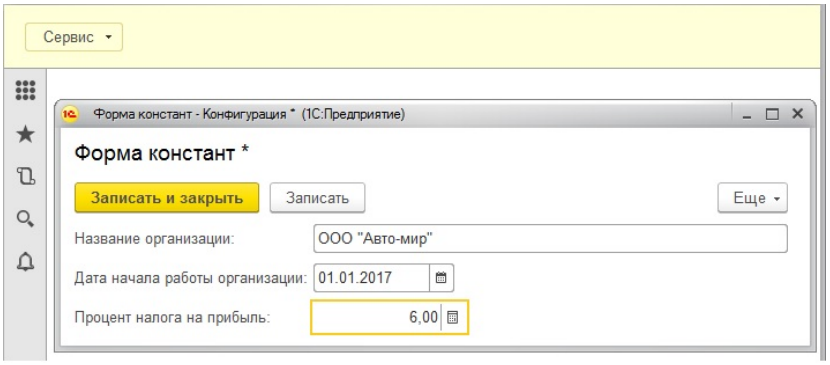


Рис. 4.1.18

Введите в неё какие-нибудь данные и нажмите на кнопку «Записать», тем самым Ваши константы запишутся в базу данных. Почему так произошло, мы изучим позже, когда будем проходить объектные типы (шестая глава).

Справочники

Изучим следующий прототип объектов конфигурации - это *Справочники*.

Справочники используются для работы с постоянной или условно постоянной информацией, но, в отличие от констант, данная информация может содержать множество значений. К примеру, если мы разрабатываем конфигурацию для учета автотранспорта, то такой информацией может быть марка автомобилей, информация о самих автомобилях, информация о водителе автомобиля и т.п.

По умолчанию у любого справочника имеется два реквизита - это *Код* и *Наименование*. *Это так называемые **стандартные реквизиты**, в действительности их больше, но мы пока остановимся на этих двух.* Иногда стандартных реквизитов бывает достаточно, но чаще всего прикладная задача требует хранения вспомогательной информации. Для этого создаются реквизиты справочника, позволяющие хранить любую дополнительную информацию об элементе справочника.

Применительно к автомобилям это может быть госномер, основной водитель, вид коробки передач (автоматическая или ручная) и т.п. Еще новая возможность справочников (в 1С 7.7. этого нет) - это возможность создавать табличные части, в которых хранится однотипная информация, количество которой может быть изменчивым. Например, для автомобиля это может быть комплектация: какой-то автомобиль может идти с подушкой безопасности и стеклоподъемниками, а какой-то просто со стеклоподъемниками.

Реквизитов и табличных частей может быть неограниченное количество.

Разберем поэтапно создание справочника. Первый справочник это *МаркиАвтомобилей*.

Кликаем правой кнопкой мышки на верхний элемент конфигурации *«Справочники»*. И нажимаем кнопку *«Добавить»*.

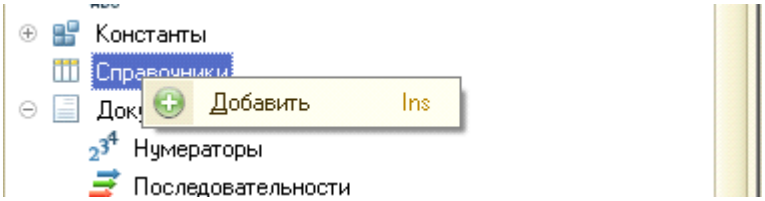


Рис. 4.1.19

Открылся конструктор справочника. Вводим название справочника *МаркиАвтомобилей* и синоним (то, что будет видеть пользователь) *Марки автомобилей*.

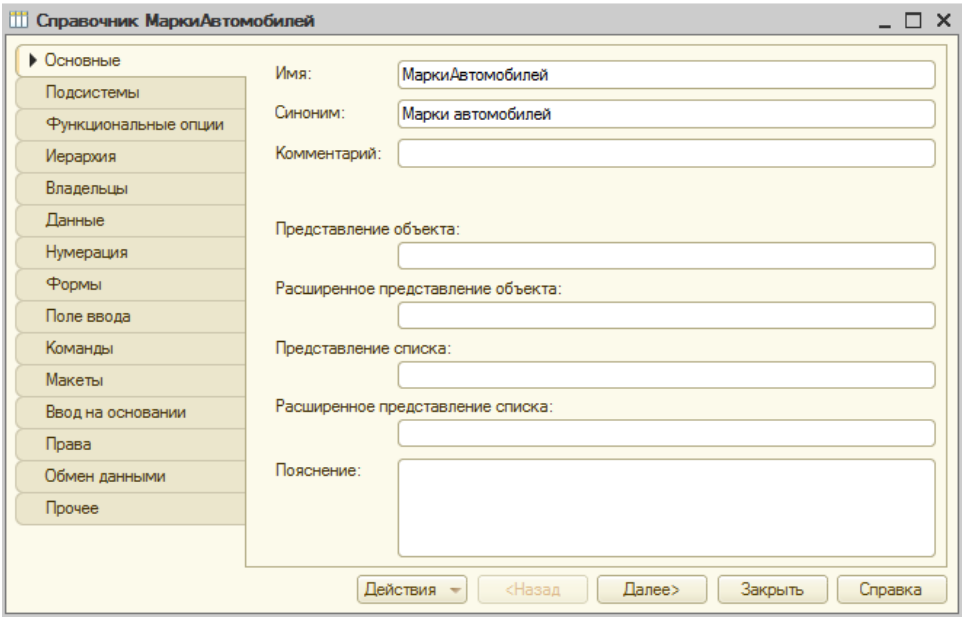


Рис. 4.1.20

В этом справочнике не будет реквизитов, поэтому сразу идем на закладку *«Данные»* и выставляем длину кода *3*, а длину наименования *10*.

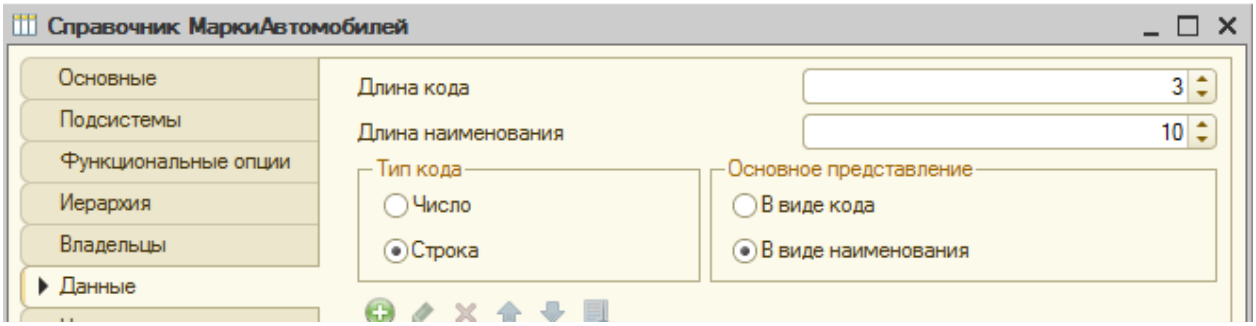


Рис. 4.1.21

На этой закладке мы можем также задать тип стандартного реквизита «Код» и основное представление этого справочника (как он будет отображаться в полях ввода). Оставим все по умолчанию и закроем конструктор этого справочника.

Точно так же с такими же значениями стандартных реквизитов самостоятельно создайте справочник *МоделиАвтомобилей*. *Подсказка: можете скопировать справочник МаркиАвтомобилей и поменять название.*

Теперь создаем справочник *Автомобили*. Точно так же кликаем правой кнопкой мышки на верхний элемент конфигурации *Справочники*, нажимаем кнопку *«Добавить»*.

Вновь открывается форма элемента справочника. Вносим название справочника: *Автомобили*, и идем на закладку *«Данные»*. Длину кода оставим ту же, что и в предыдущем

случае (3), длину наименования сделаем *50*. Теперь создаем реквизиты. Первый реквизит *ГодВыпуска*, тип *Дата* (состав даты *Дата*), второй *ГосНомер*, тип *Строка (длина 10)*.

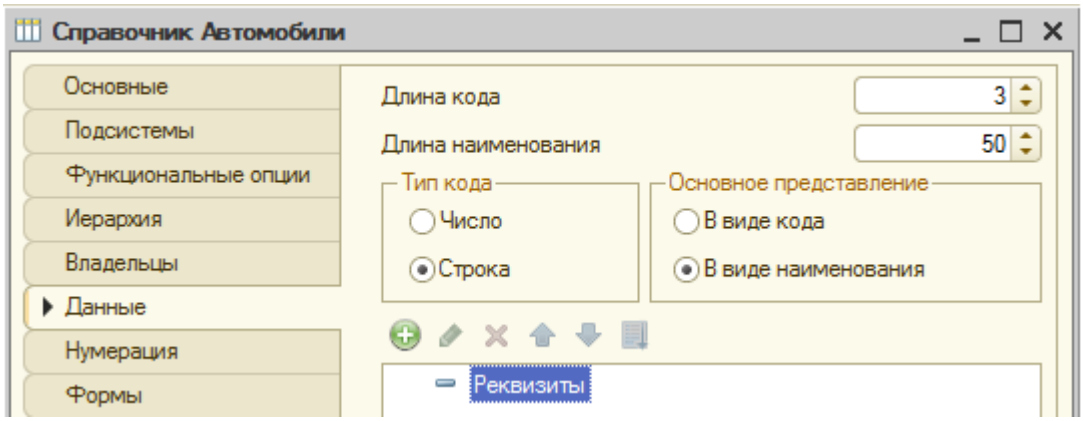


Рис. 4.1.22

И третье, самое интересное, мы создаем реквизит *Марка*, тип которого *Ссылка на справочник МаркаАвтомобиля.*

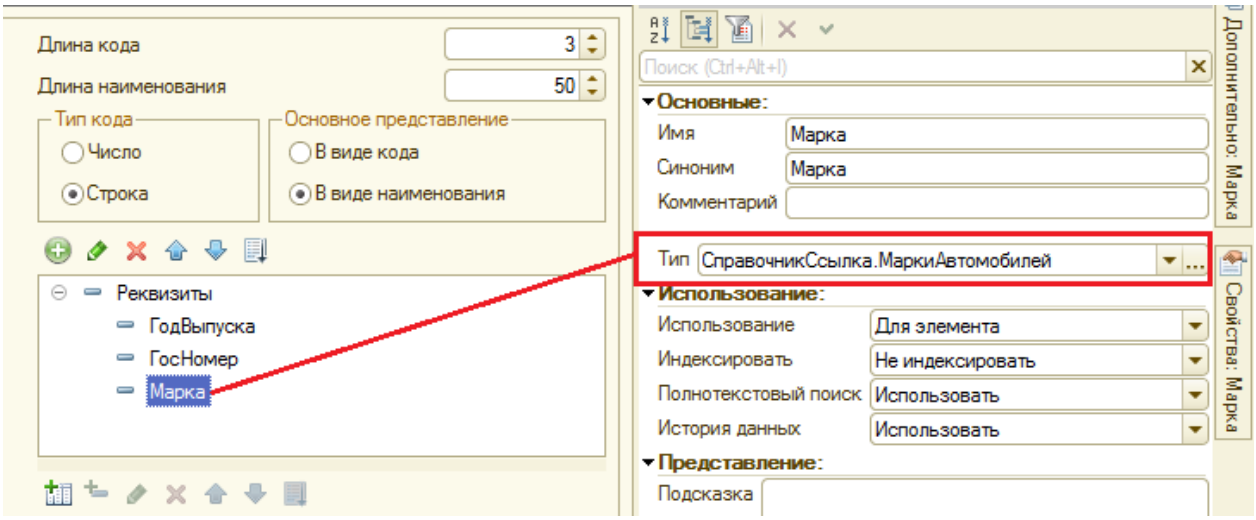


Рис. 4.1.23

Точно так же добавьте реквизит *Модель*, тип которого *Ссылка на справочник МоделиАвтомобилей*.

Более подробно ссылочные типы мы будем разбирать в следующей главе. Напоследок создаем табличную часть и назовем ее *ДополнительныеОпции*. В табличной части создаем реквизит *Опция* с типом *ссылка на справочник Опции* (создайте самостоятельно, без реквизитов, длина кода 3, длина наименования 50).

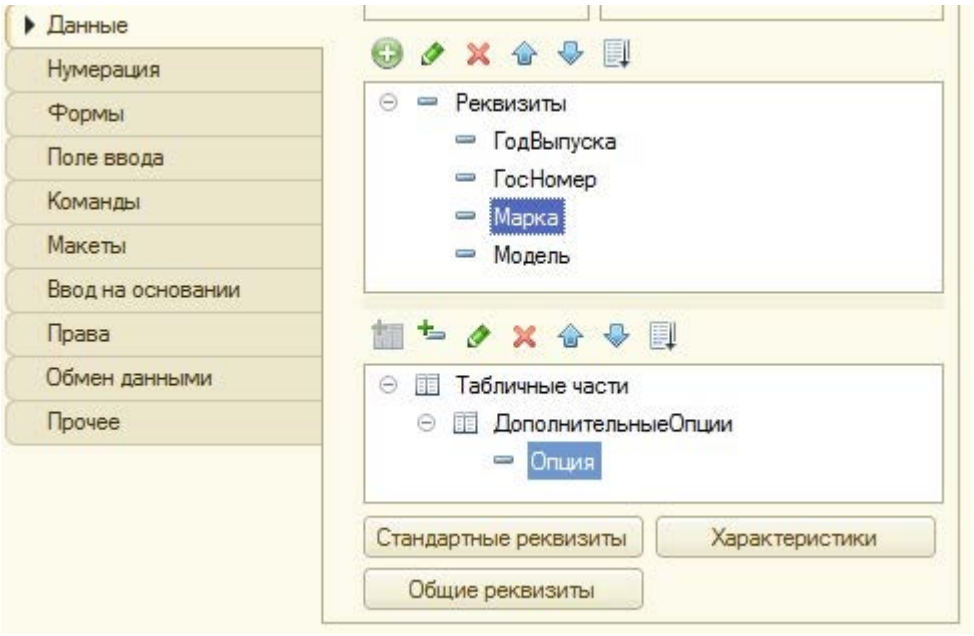


Рис. 4.1.24

И последний справочник, который создадим в этой части, это будет справочник *Гаражи*, у которого нет реквизитов, а длина кода и наименования такая же, как у справочника *Опции*.

Создайте его самостоятельно. Сохраните конфигурацию.

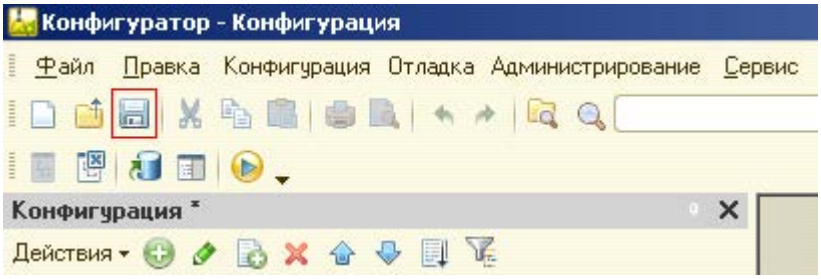


Рис. 4.1.25

Обновите базу данных.

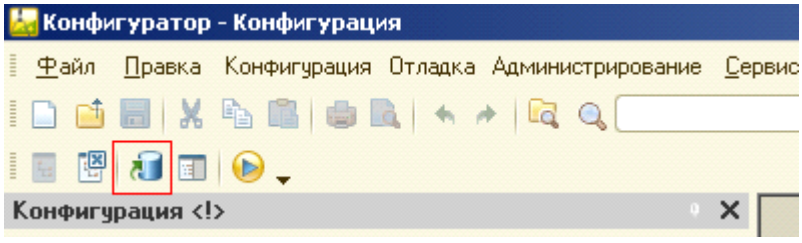


Рис. 4.1.26

Теперь посмотрим, как выглядит наш справочник. И заполним его парой элементов. Запустите «1С:Предприятие». Все справочники появились в верхней панели (см. рис. 4.1.27).

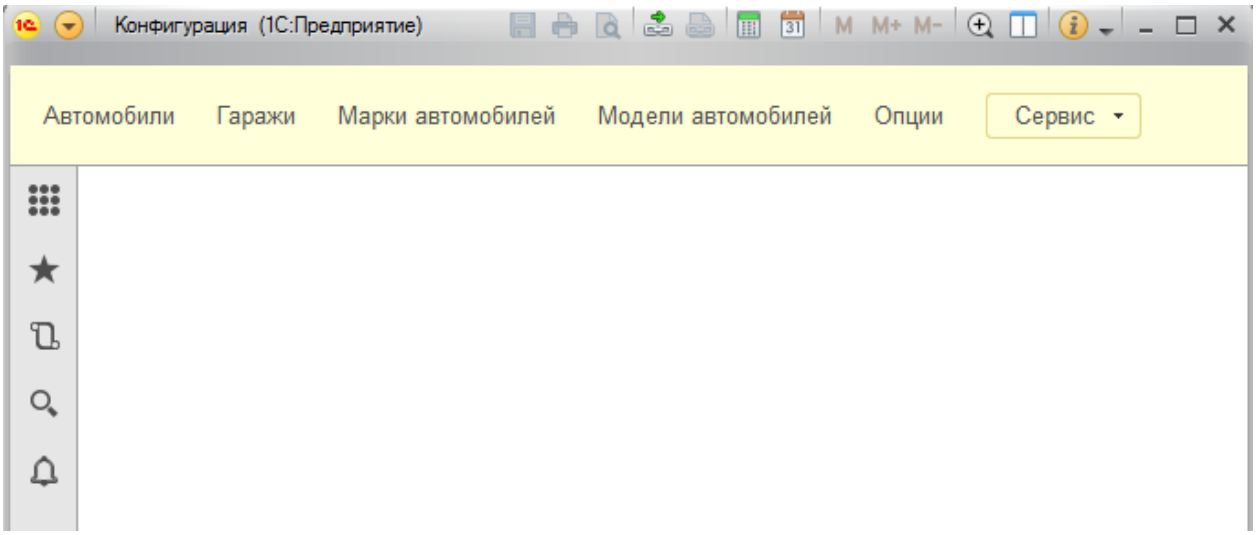


Рис. 4.1.27

Пока не создавайте никакие элементы справочников, сделаем это после того, как изучим все свойства этого прототипа.

Предопределенные элементы

У любого справочника можно создать ***Предопределенный элемент*** - это элемент, который создан в конфигураторе, и с ним можно работать как с обычным элементом. Этот элемент есть всегда, в платформе 8.2 его нельзя было пометить на удаление и удалить, в платформе 8.3. предопределенные объекты можно удалять в пользовательском режиме. Создадим два предопределенных элемента для справочника *Марки автомобилей* - *«Российские» и «Иномарки»*.

Зайдите в форму справочника, для этого кликнув правой кнопкой мышки по справочнику *Марки автомобилей,* и выберете в открывшемся меню *«Открыть предопределенные данные»*

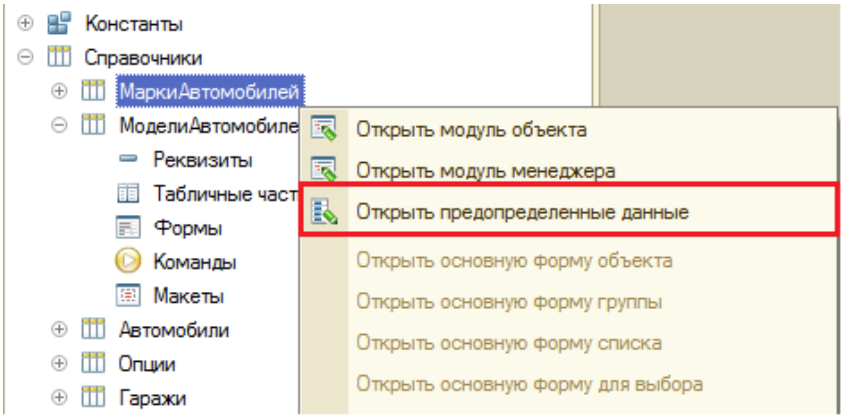


Рис. 4.1.28

В открывшемся списке добавьте новый элемент.

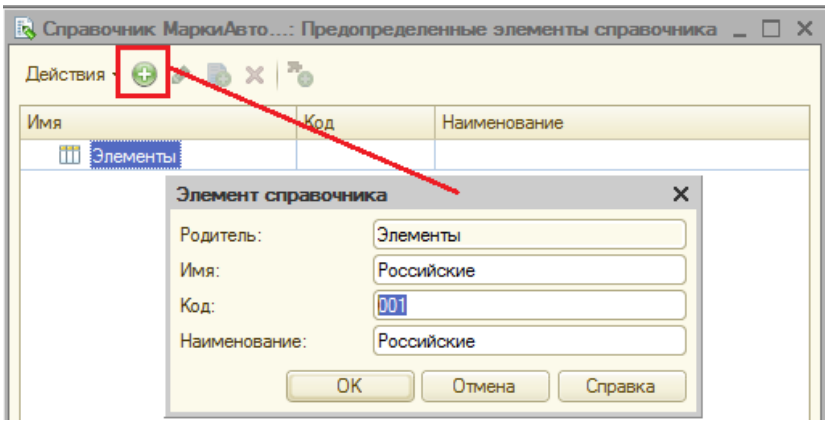


Рис. 4.1.29

Точно так же создадим элемент «Иномарка».

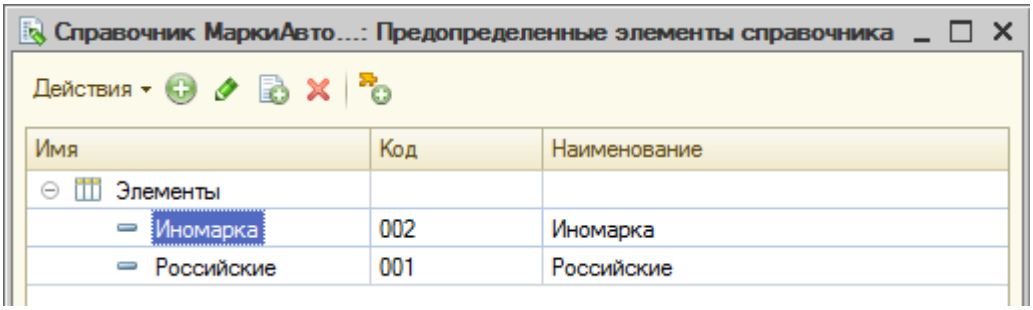


Рис. 4.1.30

Рекомендую все предопределенные элементы справочника создавать до того, как Вы начнете заполнять его, это позволит избежать проблем с нумерацией.

Предопределенные элементы Вы создали, теперь сохраним конфигурацию, обновим базу данных, запустим «1С:Предприятие» и кликнем по команде «Марки Автомобилей» в верхней панели. В результате выполнения команды откроется форма списка справочника «Марки автомобилей», где уже будут два предопределенных элемента

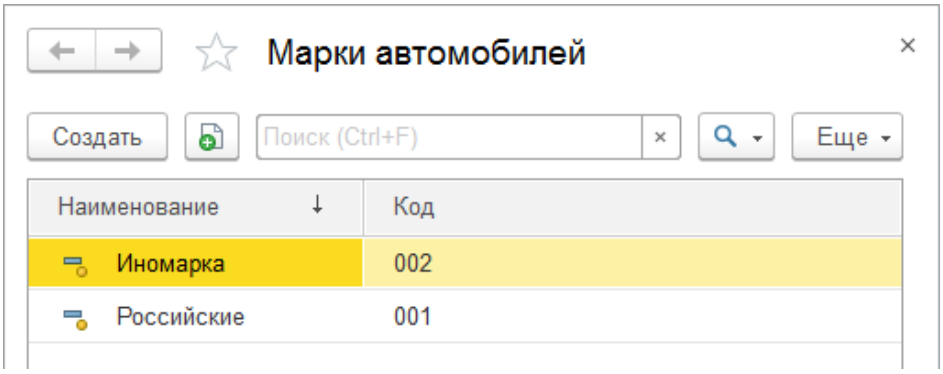


Рис. 4.1.31

Если мы создадим новый элемент в «1С:Предприятии» (см. рис. 4.1.32), то он отличаться от предопределенного будет по значку рядом с наименованием (см. рис. 4.1.33).

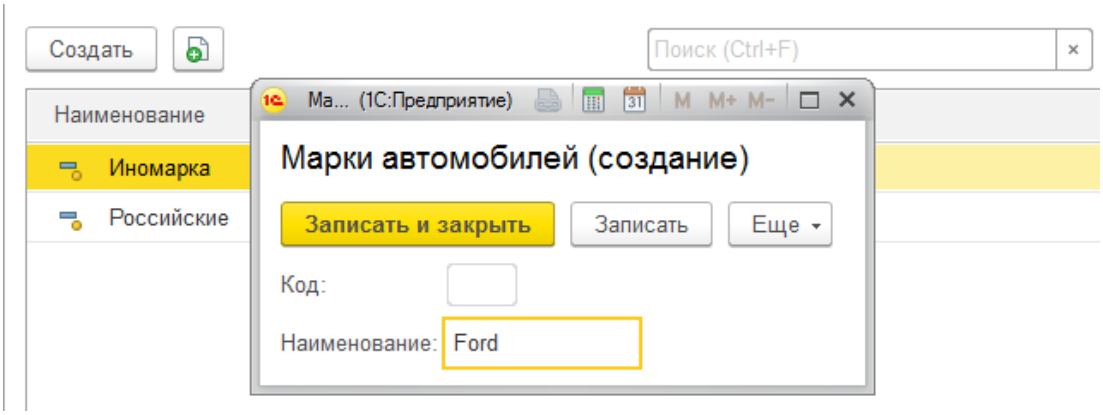


Рис. 4.1.32

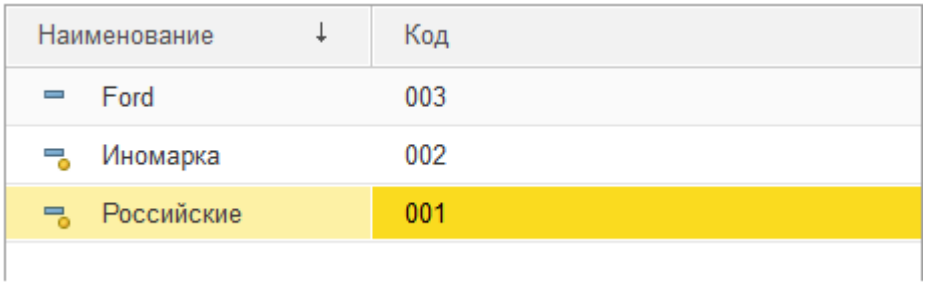


Рис. 4.1.33

Подчиненный справочник

В конфигурации 1С 8 можно разрабатывать *Подчиненные справочники*. В ***Подчиненном справочнике*** каждый его элемент имеет владельца, который является элементом или группой другого справочника. Элемент подчиненного справочника не может существовать без владельца.

Например, у нас есть два справочника *«Марки автомобилей»* и *«Модели автомобилей»*. У марки автомобиля может быть несколько разных моделей, поэтому естественно их связать между собой. В этом случае справочник *«Марки автомобилей»* будет владельцем справочника *«Модели автомобилей»*. Сделаем это.

Для этого зайдем в конструктор справочника *«Модели автомобилей»* и перейдем на закладку *«Владельцы»*.

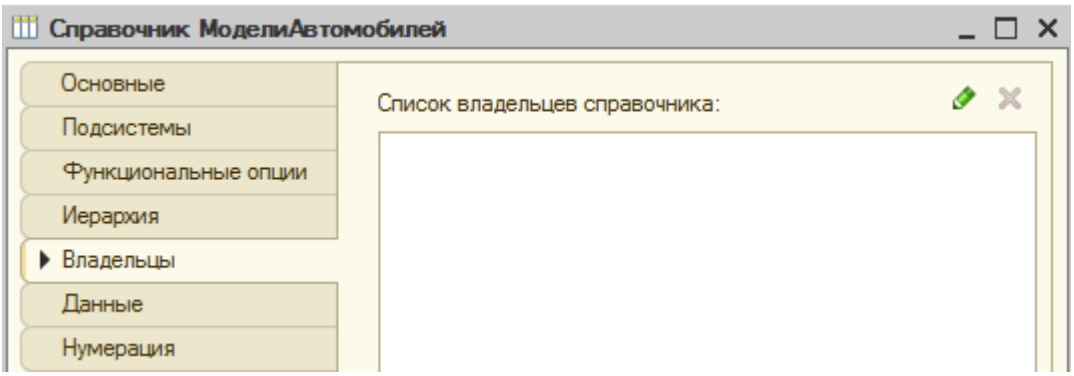


Рис. 4.1.34

Список владельцев пустой. Нажмите на карандаш, должно открыться окно выбора объектов, ставим флажок возле справочника *«Марки автомобилей»* и нажимаем *«ОК»*.

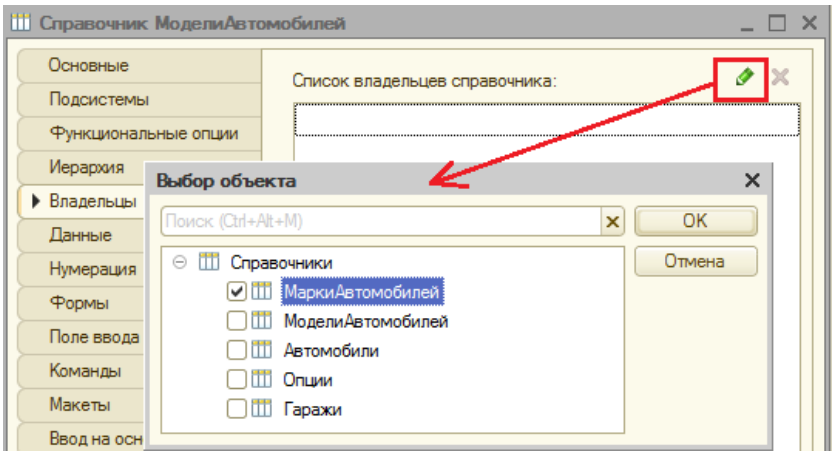


Рис. 4.1.35

Использование подчинения оставляем *Элементы*.

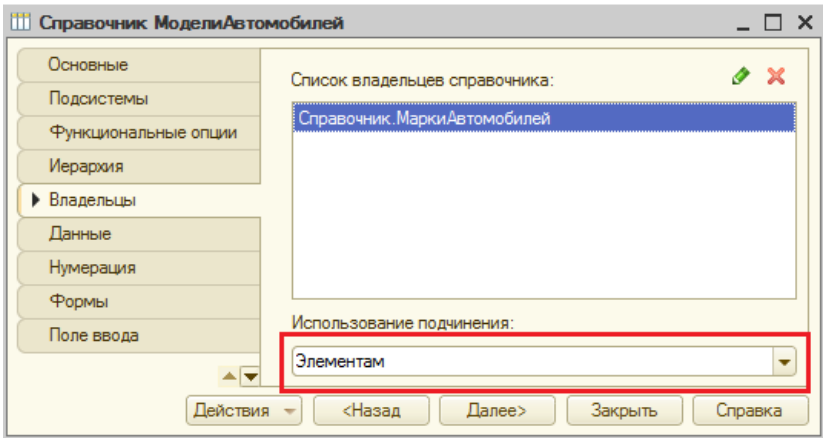


Рис. 4.1.36

Сохраните конфигурацию и запустите «1С:Предприятие». Откроем справочник *«Марки автомобилей»* и зайдем в элемент, который создали в прошлом подразделе - Ford.

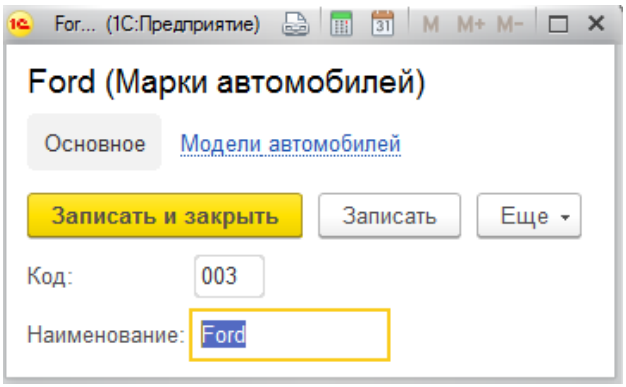


Рис. 4.1.37

Как видите, вверху элемента справочника появилась команда «Модели автомобилей». Платформа автоматически разместила эту команду в панели навигации элемента справочника. Если мы сейчас по ней кликнем, то откроется список всех моделей автомобилей, у которых владелец текущей элемент справочника «Марки автомобилей». В нашем случае он пустой, и мы добавим какую-нибудь модель автомобиля (см. рис. 4.1.38).

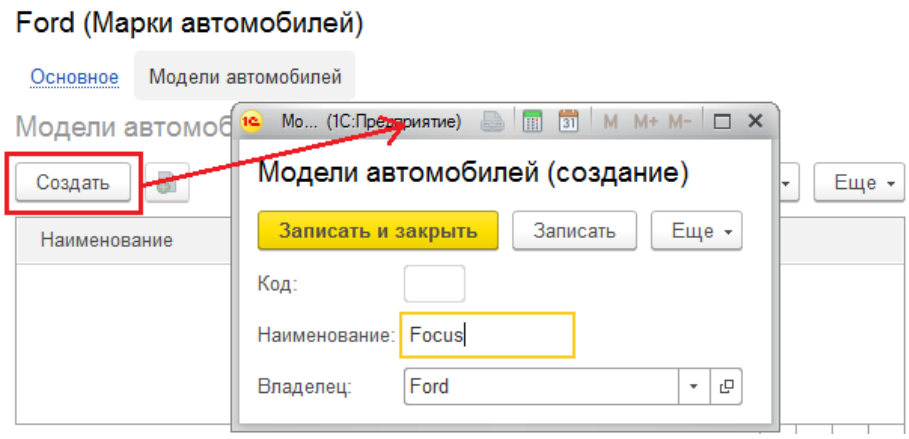


Рис. 4.1.38

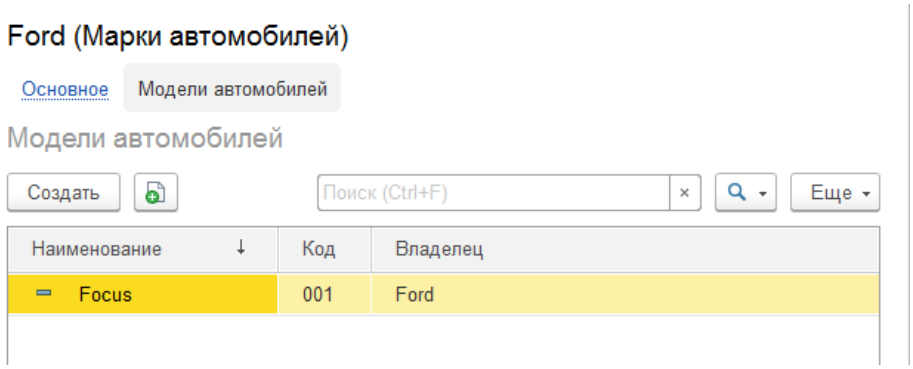


Рис. 4.1.39

Теперь создадим новую марку (пусть это будет KIA) и добавим для неё новую модель (пусть это будет Rio).

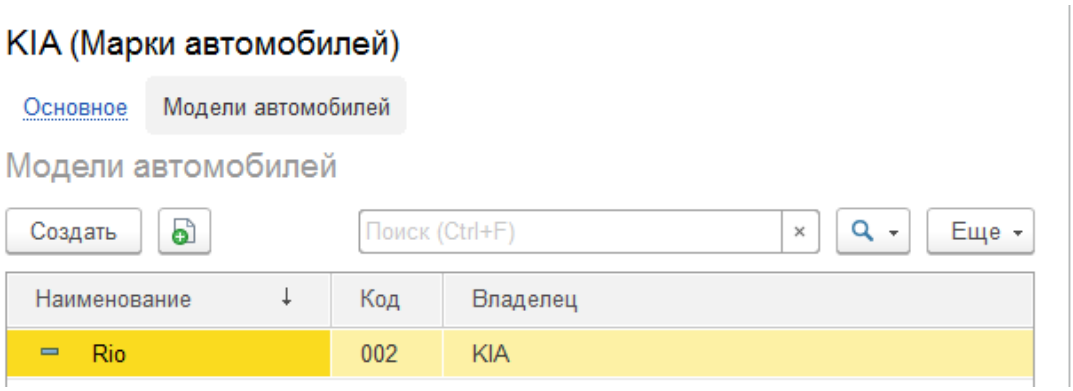


Рис. 4.1.40

А теперь попробуем создать новый элемент справочника *Автомобили*, где выберем какую-нибудь марку (пусть будет Ford), и попробуем выбрать модель.

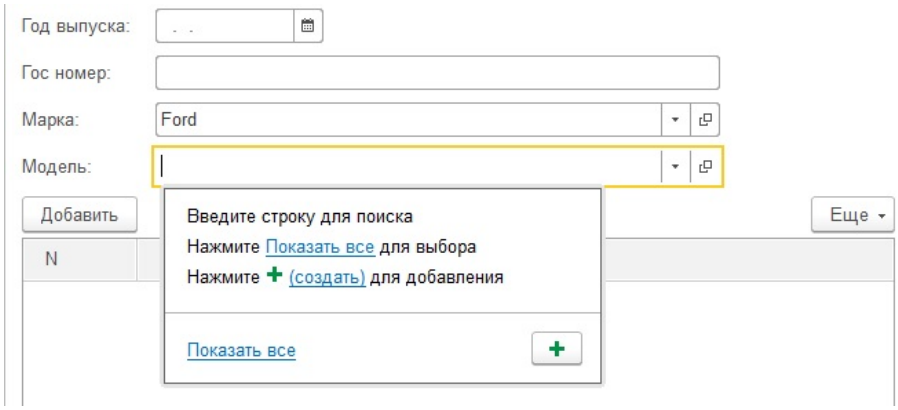


Рис. 4.1.41

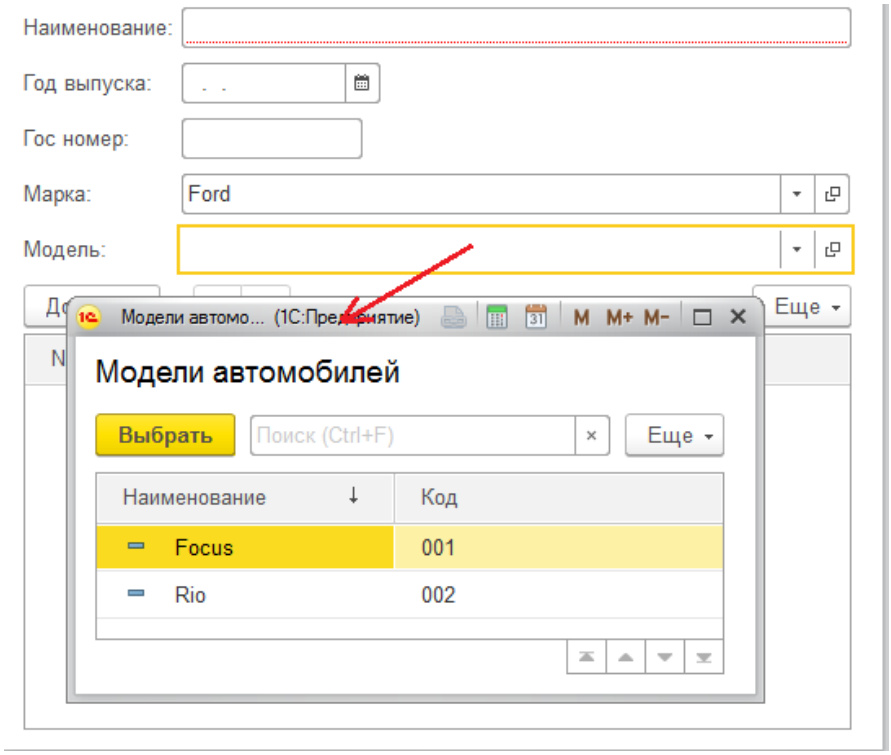


Рис. 4.1.42

Как видите, в форме выбора модели есть выбор и моделей Ford, и моделей KIA, что согласитесь, немного неудобно. Как же сделать так, чтобы в форме выбора были модели только для той марки, которая выбрана в реквизите элемента справочника. Для этого нам понадобится зайти в конфигураторе в свойства реквизита *«Модель»* справочника *«Автомобили»* (дважды кликнув левой клавишей мышки по реквизиту в дереве конфигурации). В открывшейся палитре свойств нас интересует свойство «Связи параметров выбора».

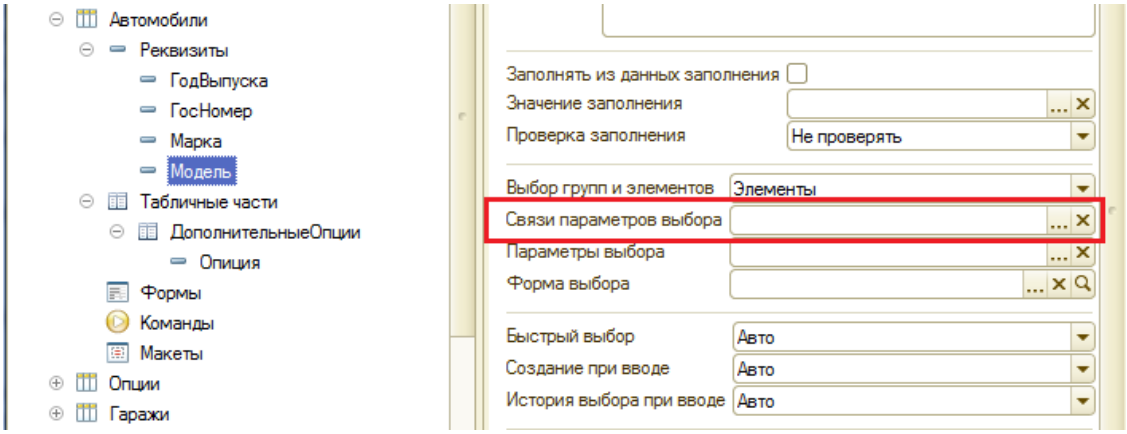


Рис. 4.1.43

При помощи этого свойства можно настроить зависимость выбираемых параметров от значений других реквизитов. Откроем форму «Связи параметров выбора», нажав на кнопку «…».

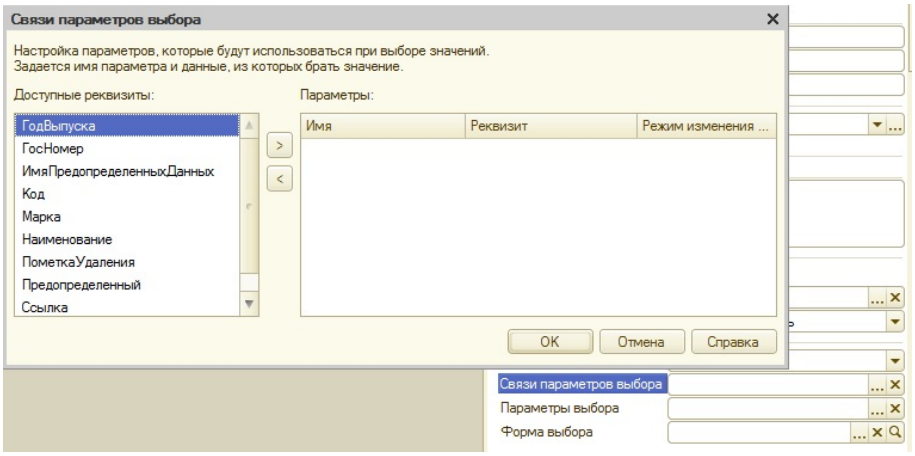


Рис. 4.1.44

Поскольку нам нужно установить связь реквизита *Модель* с реквизитом *Марка*, то перетащим этот реквизит из левой части в правую.

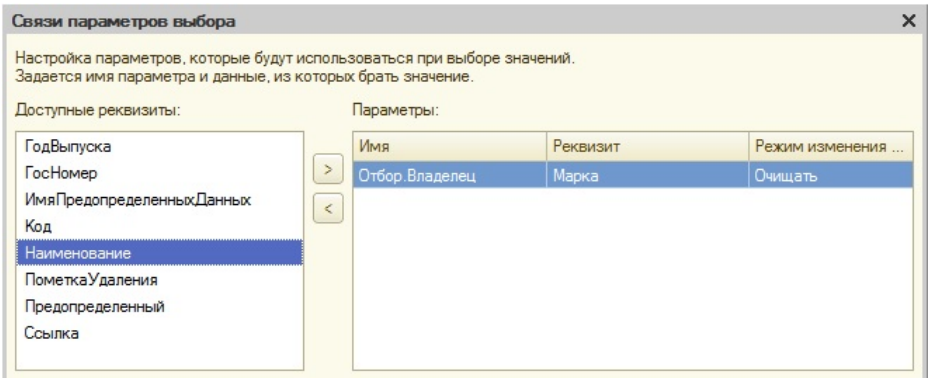


Рис. 4.1.45

Нажмем кнопку «ОК» формы связи параметров выбора.

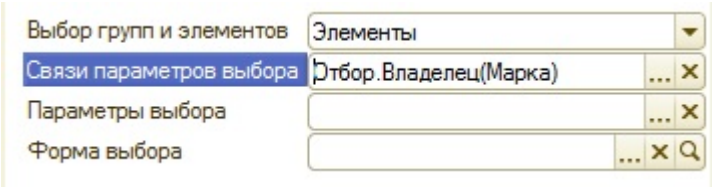


Рис. 4.1.46

Теперь сохраним конфигурацию, обновим базу данных, попробуем создать новый элемент справочника «Автомобили». Так же выберем марку Ford и попробуем выбрать модель.

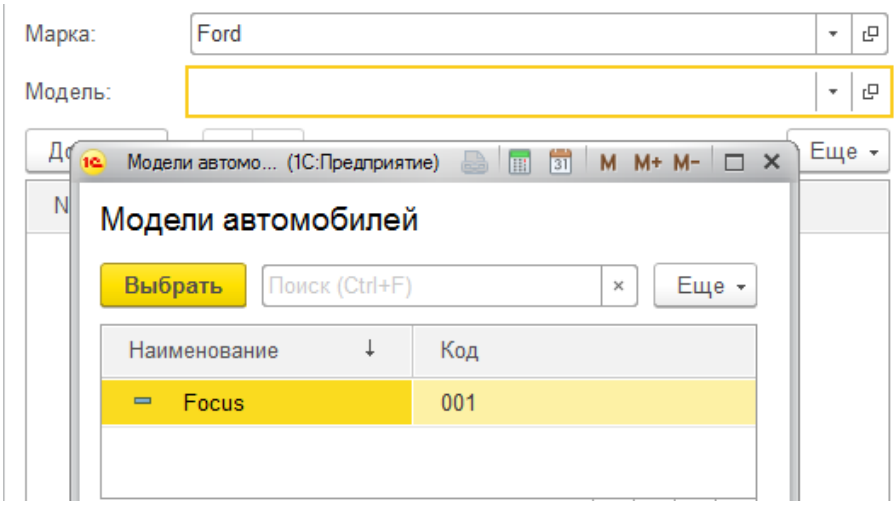


Рис. 4.1.47

Как видите, вышла только одна модель, у которой владелец марка Ford.

Иерархический справочник

Любой справочник можно сделать ***Иерархическим*** – это значит, что пользователь сможет создавать каталоги (или, говоря простым языком, папки), в которых будут содержаться элементы.

Сделаем справочник *Гаражи* иерархическим. Для этого переходим на закладку «*Иерархия*» конструктора справочника и установим флаг «*Иерархический*».

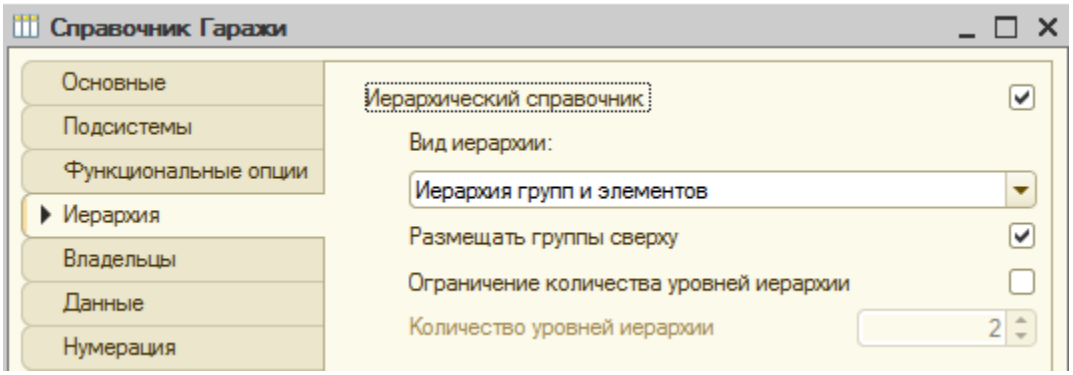


Рис. 4.1.48

Иерархия у справочников бывает двух видов: *Иерархия групп и элементов* и просто *Иерархия элементов*.

Иерархия групп - это каталоги, которые в себе содержат определенные элементы. А что такое *Иерархия элементов*? Это когда один элемент подчинен другому.

К примеру, у Вас есть большой гараж (ангар), в котором много маленьких гаражей. Вам необходимо вести учет автомобилей по большому гаражу, поскольку они могут храниться вне маленьких, а также учет по маленьким гаражам. В этом случае справочник *Гаражи* будет удобно сделать с *Иерархией элементов*. Но если Вам еще понадобится объединять гаражи по какой-то общей группе, по которой не будет вестись отдельного учета, то необходимо сделать *Иерархию групп и элементов*. Например, у Вас может быть каталог *Авто База,* в которой будут перечислены все гаражи.

Посмотрим, как выглядит иерархический справочник в «1С:Предприятии». Сохраним конфигурацию, обновим базу данных и зайдем в форму списка справочника «Гаражи».

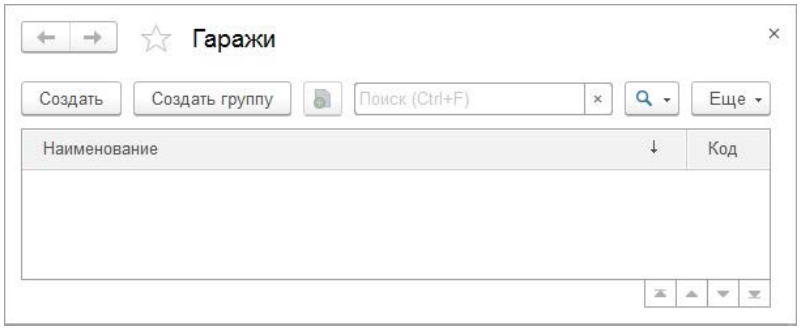


Рис. 4.1.49

Как видите, в форме списка появилось две команды. «Создать» и «Создать группу». При выполнении команды «Создать» будет открыта форма на создание нового элемента, а при выполнении команды «Создать группу» - форма на создание группы (папки).

Создадим каталог *Авто база 1*.

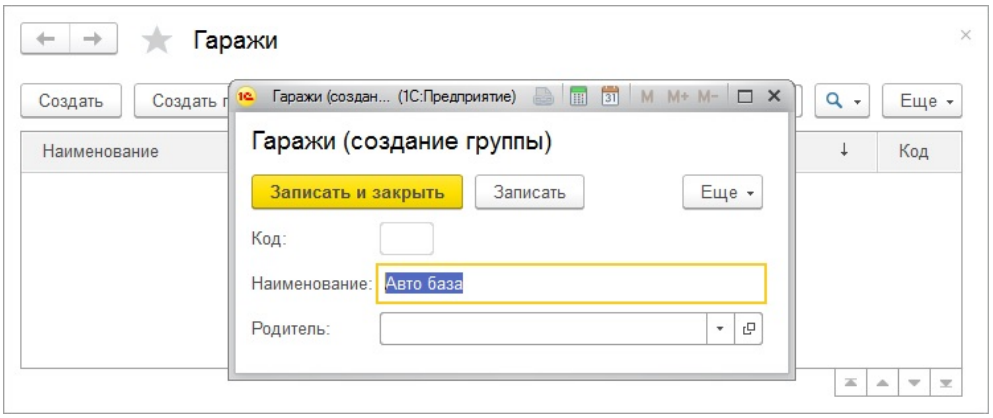


Рис. 4.1.50

Пишем наименование *«База 1»*. Родитель пустой. Нажмем кнопку «Записать и закрыть», группа будет создана. Зайдите в папку, кликнув на нее два раза, и добавьте элемент. В этот раз выполним команду «Создать».

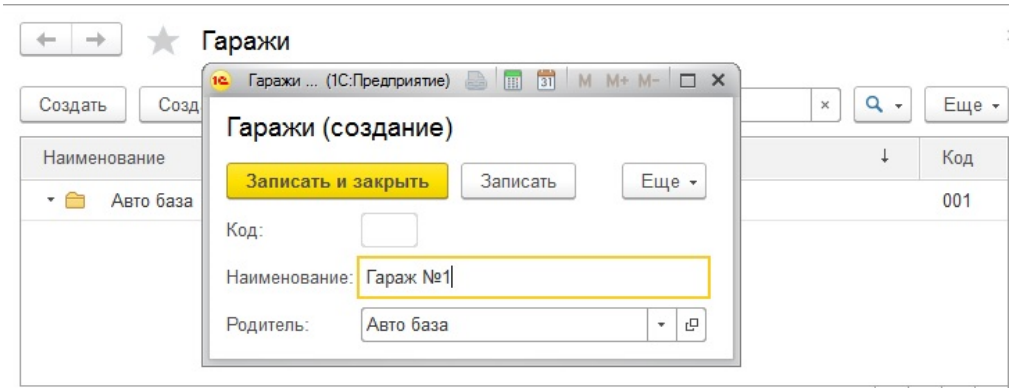


Рис. 4.1.51

Открылась форма, где уже родитель заполнен. Назовем элемент *Гараж № 1*.

Сохраним элемент.

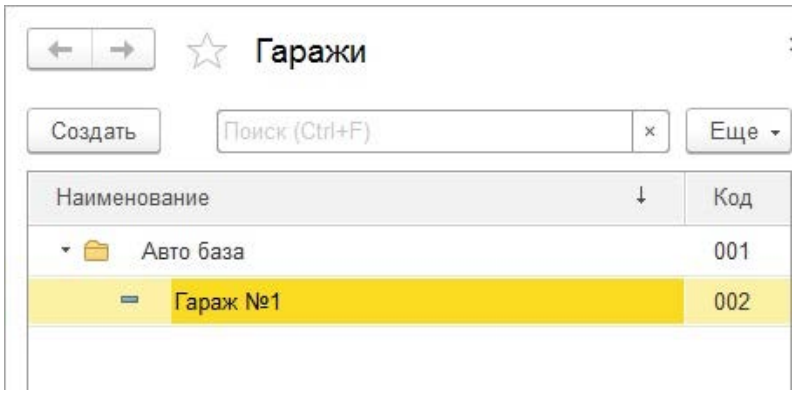


Рис. 4.1.52

Как Вы видите, данный гараж находится в группе *Авто база*.

Документы

Документы предназначены для отражения определенных событий в программе, которые имеют отношение к предметной области. Например, касательно учета автотранспорта, это могут быть события прибытия автомобиля в гараж, убытия из гаража и постановки на ремонт. Основными стандартными реквизитами документов являются *Номер документа* и *Дата документа*. Как и в случае со справочниками, остальная вспомогательная информация может храниться в реквизитах и табличных частях. Суть их такая же, как у реквизитов и табличных частей для справочников.

Основное отличие *Документов* от *Справочников* в том, что *Документ* может делать движения в регистрах накопления, сведений, бухгалтерии и расчета. Что такое *Движения документа*? *Движения документа* - это записи в регистрах накопления, сведений, бухгалтерии и расчета, которые создаются при проведении документа и уничтожаются при отмене проведения документа. Все просто. Разработчик самостоятельно настраивает, по каким регистрам будет делать движение его документ, естественно, что регистр перед этим должен быть создан.

Создадим теперь четыре документа: *Прибытие в гараж*, *Выбытие из гаража*, *Заправка топлива* и *Отчет о расходе топлива*.

Впоследствии первый будет делать соответствующие движения по регистру накопления *«Пробег автомобиля»,* а последние два по регистру накопления *«Расход топлива»,* но мы дойдем до этого, когда будем разбирать регистры накопления.

Для того чтобы создать документ, Вам необходимо кликнуть правой кнопкой мышки по пункту *«Документы»* в конфигурации и выбрать пункт *«Добавить»*.

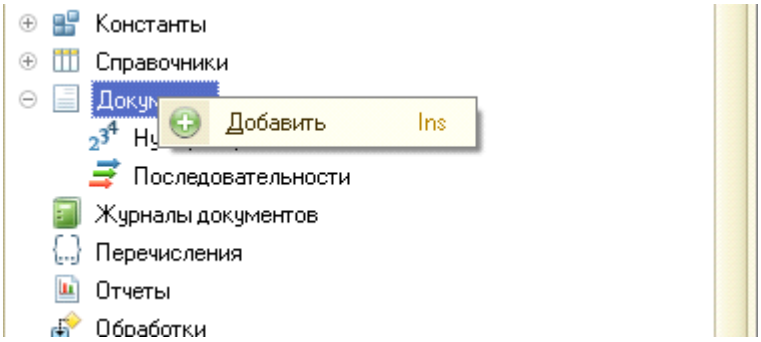


Рис. 4.1.53

Открылось меню вновь созданного документа. Назовите Ваш документ *ПрибытиеВГараж*, синоним должен появиться автоматически.

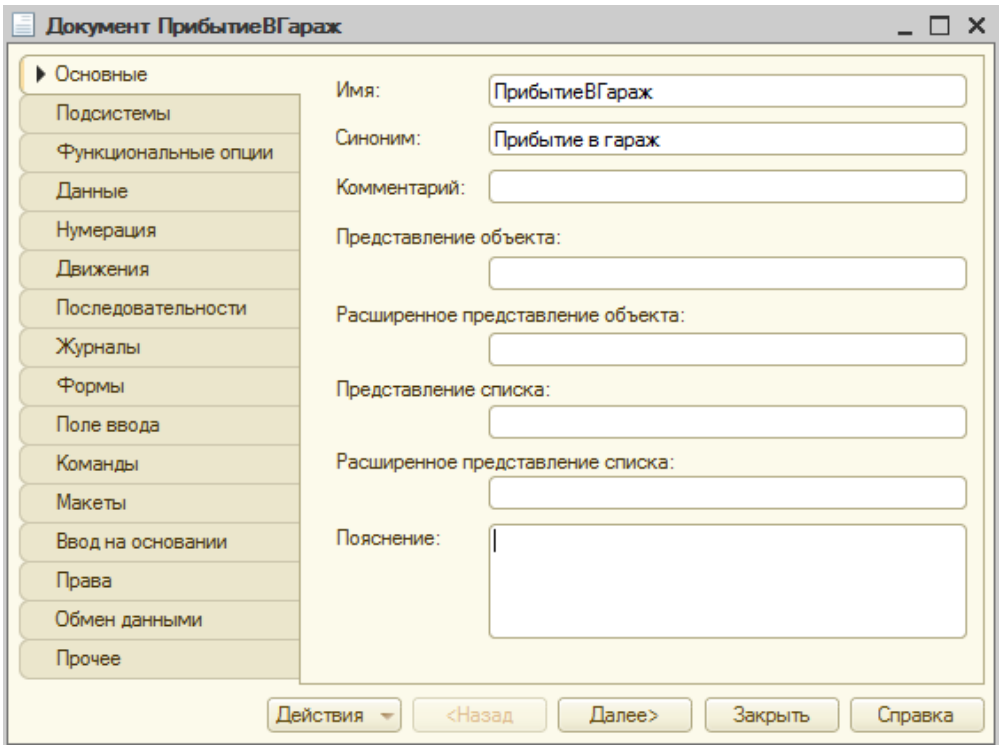


Рис. 4.1.54

Перейдите на закладку *«Данные»* и создайте три реквизита: *Автомобиль*, *Гараж* и *Дата прибытия*.

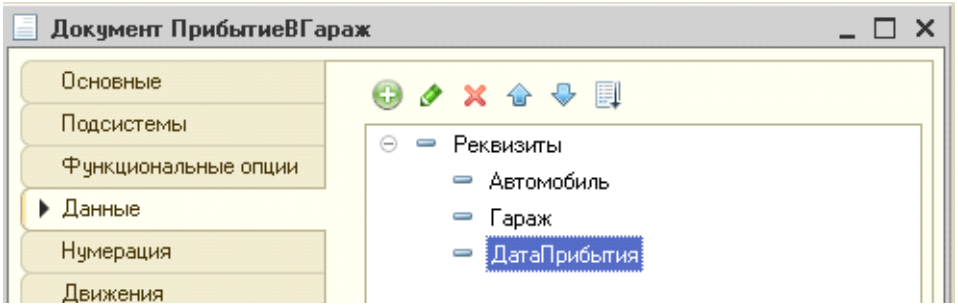


Рис. 4.1.55

Первый реквизит будет иметь тип *Ссылка на справочник Автомобили,* второй - тип С*сылка на справочник Гараж*, а третий – *Дата (состав Дата и время)*.

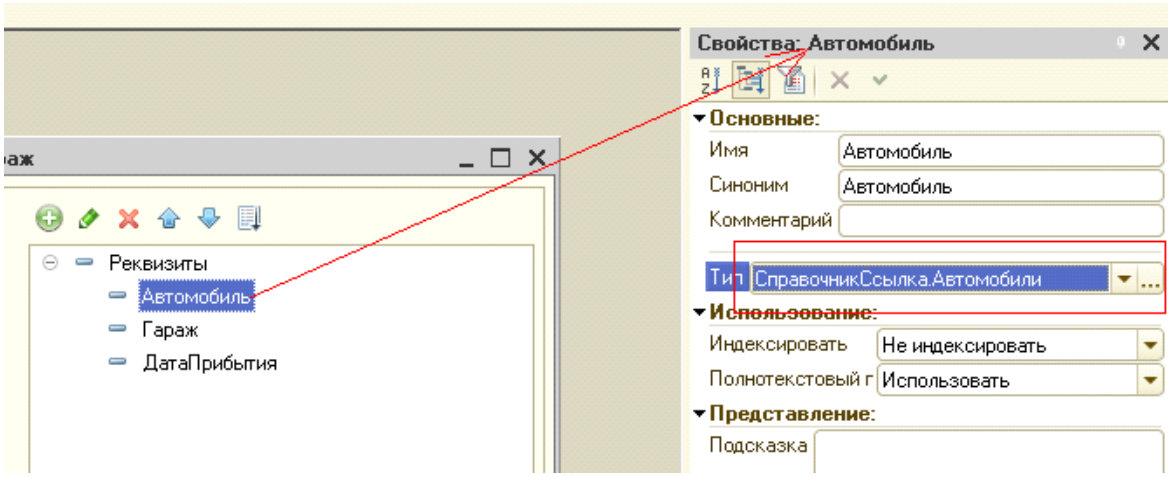


Рис. 4.1.56

Точно так же создайте документ *Выбытие из гаража*. Реквизиты – *Автомобиль, Гараж и ДатаВыбытия.* Типы реквизитов аналогичны типам в документе *Прибытие в гараж*.

Теперь создайте документ *Заправка топлива*. Поначалу действуйте все так же. Создайте два реквизита: *ДатаЗаправки* с типом *Дата* (состав даты *Дата*) и *ТипТоплива* (ссылка на справочник *ТипыТоплива*, справочник *ТипыТоплива* создайте самостоятельно, без реквизитов, длина кода 3, длина наименования 10).

Создайте табличную часть *Автомобили*, где будет два реквизита: Автомобиль (тип *Ссылка на справочник Автомобили*) и Количество (тип *Число*(10,2)). Для этого идите в закладку *Данные* и кликните правой кнопкой мышки на *Табличную часть*. После нажмите на кнопку *«Добавить».*

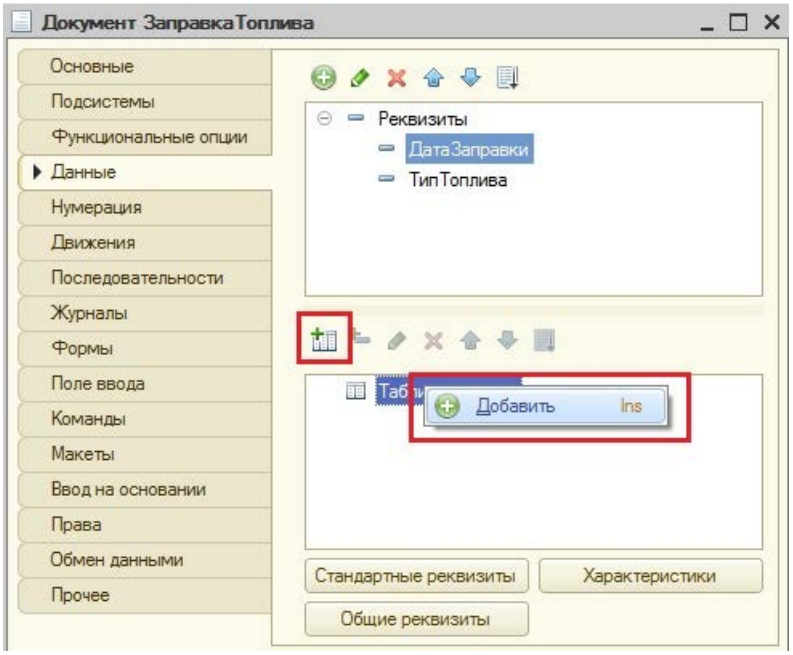


Рис. 4.1.57

Табличная часть будет создана, переименуйте ее в *«Автомобили»* (сделать это можно в палитре свойств табличной части, чтобы в неё зайти, нужно дважды кликнуть мышкой по нужной табличной части).

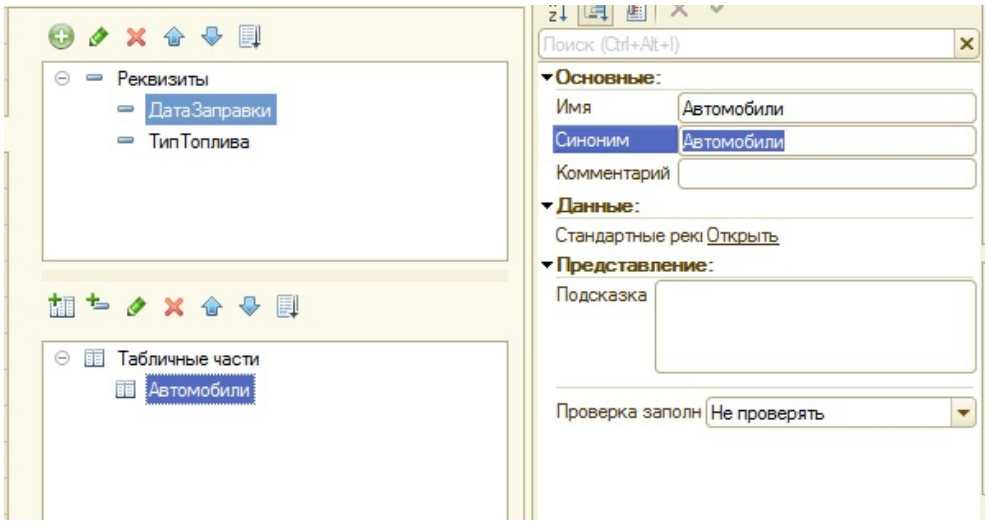


Рис. 4.1.58

И добавьте новый реквизит данной табличной части.

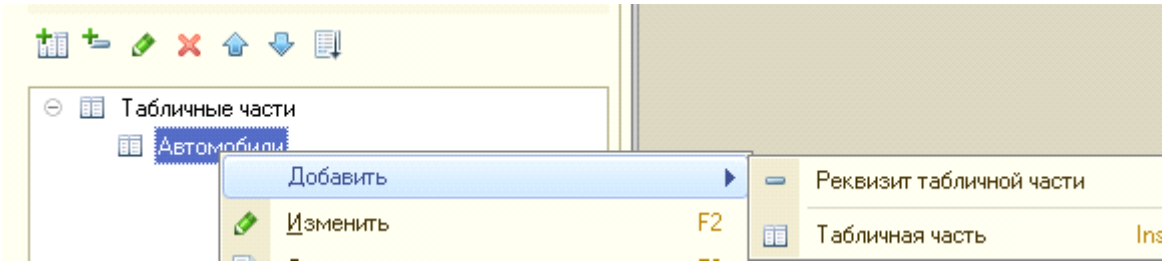


Рис. 4.1.59

Назовите его *Автомобиль* и присвойте тип *Ссылка на справочник Автомобили.*

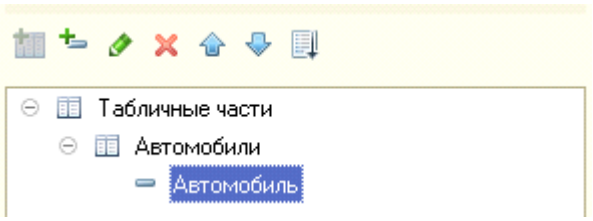


Рис. 4.1.60

Точно так же добавьте реквизит *Количество* с типом *Число* (длинна 10, точность 0).

В дальнейшем число с длинной X, а точностью Y будем писать так: Число (X,Y).

Осталось создать документ *«Отчет о расходе топлива»*, сделайте это самостоятельно, в этом документе будет точно такой же набор полей, как и в документе «*Заправка топлива*», только вместо *ДатаЗаправки* будет поле *ДатаОтчета*.

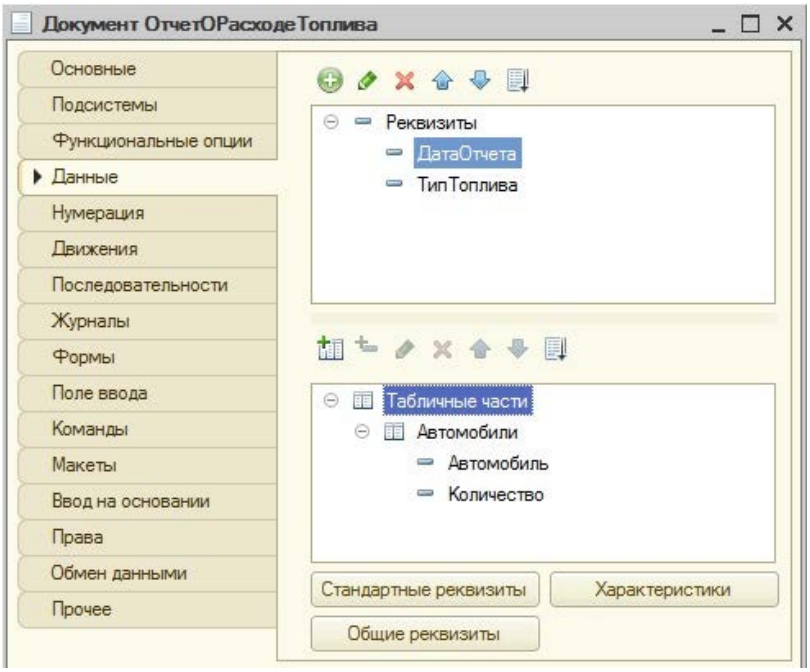


Рис. 4.1.61

Мы создали все необходимые документы для нашей конфигурации. Зайдите в «1С:Предприятие» и попробуйте создать несколько разных документов.

Ввод на основании

Рассмотрим еще одну интересную особенность объектов 1С - это *Ввод на основании*. Что такое *Ввод на основании*? Это значит, что на основе имеющегося объекта (документа или справочника) можно создать новый объект другого вида, причем часть полей нового объекта будут заполнены автоматически и взяты из основного объекта. Данная особенность очень удобна, поскольку позволяет в некоторых случаях экономить пользователю время и исключает возникновение дополнительных ошибок.

Доработаем нашу конфигурацию так, чтобы на основании документа *Прибытие в гараж* можно было создать документ *Выбытие из гаража*, причем реквизит *Автомобиль* и *Гараж* заполнятся автоматически.

Для этого зайдите в менеджер объекта *Выбытие из гаража*. И перейдите на закладку *«Ввод на основании»*.

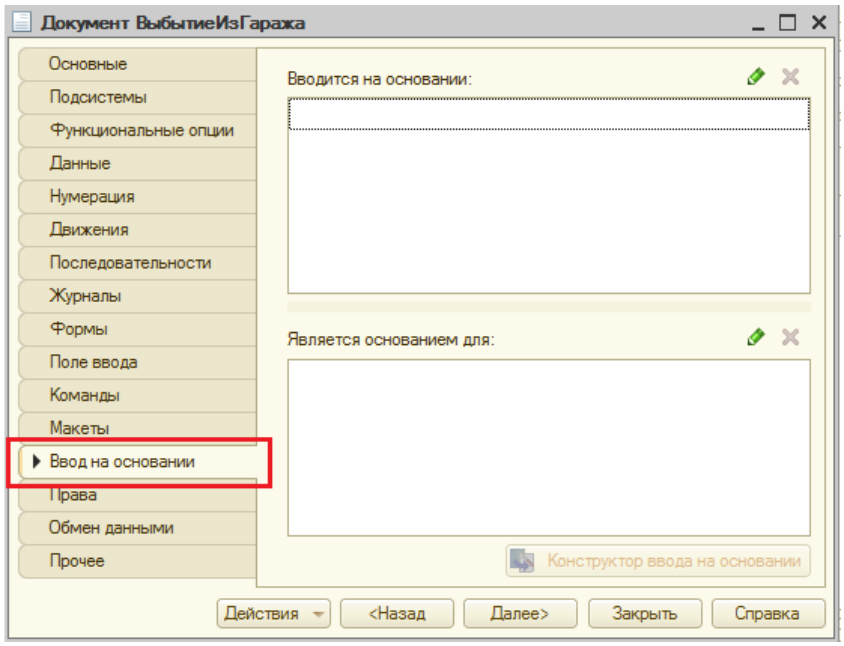


Рис. 4.1.62

В верхнем окне *«Вводится на основании»* нажмите на карандаш, откроется окно выбора объектов.

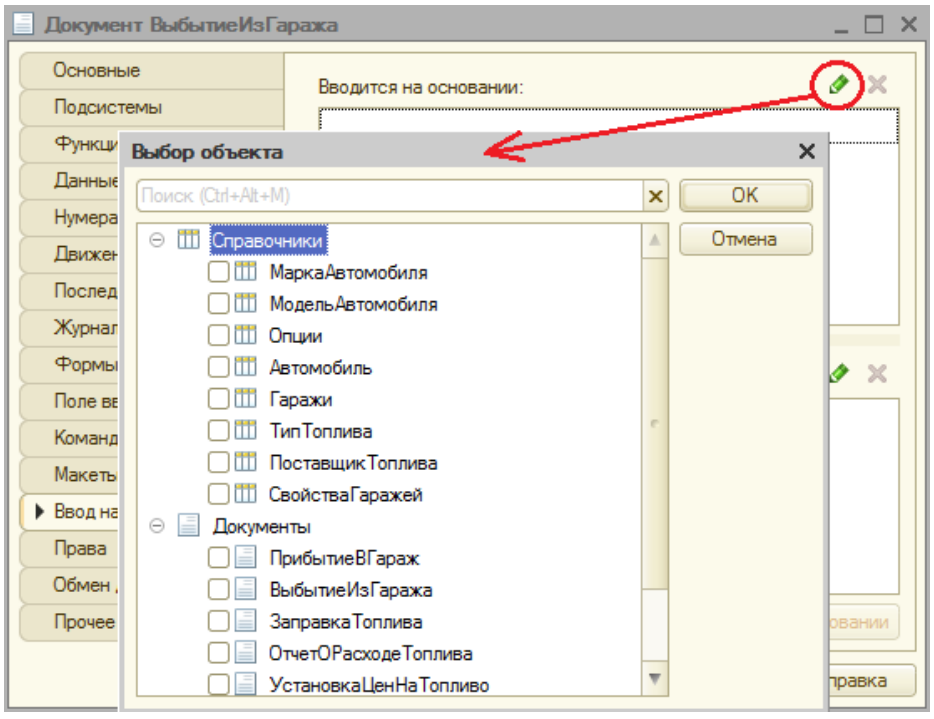


Рис. 4.1.63

В окне выбора объектов, установите флажок напротив документа *Прибытие в гараж* и нажмите кнопку *«ОК»*.

После этого в верхнем окне закладки «Ввод на основании» появится название выбранного документа:

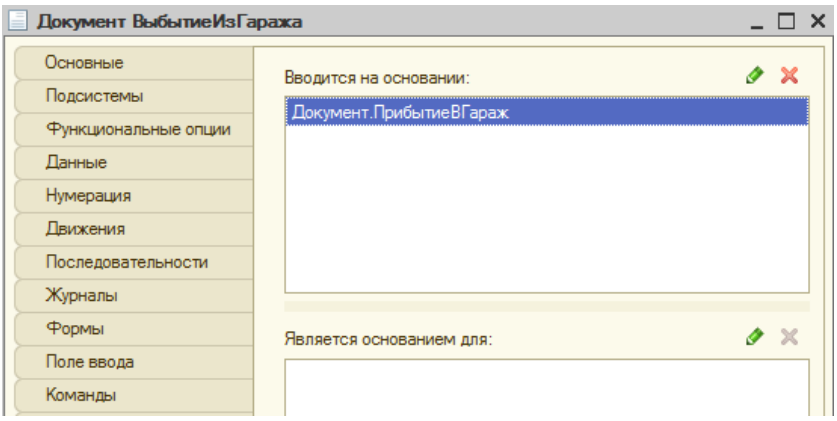


Рис. 4.1.64

А если Вы зайдете в аналогичную закладку менеджера документа «Прибытие в гараж», то увидите, что заполнено окно «Является основанием».

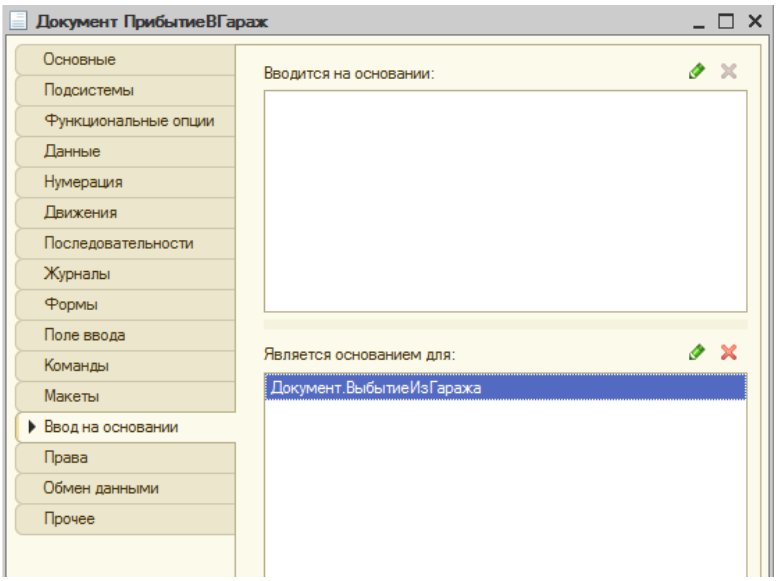


Рис. 4.1.65

Теперь Вам необходимо разработать механизм заполнения документа основания. Для этого необходимо создать в модуле объекта соответствующего документа процедуру *Обработка основания*.

Можно это сделать вручную, а можно воспользоваться конструктором. Мы воспользуемся конструктором. Для этого, не выходя из закладки *«Ввод на основании»* конструктора документа *Выбытие из гаража,* нажимаем на кнопку *«Конструктор ввода на основании»*.

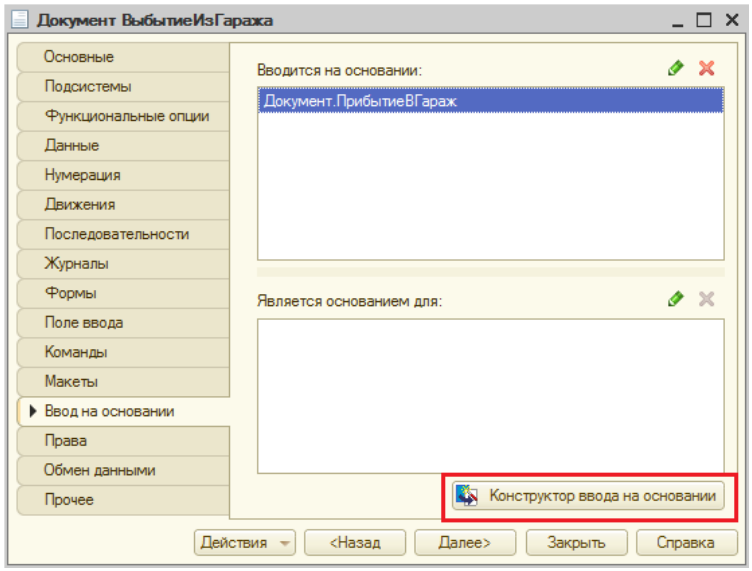


Рис. 4.1.66

Откроется конструктор, в котором необходимо заполнить реквизиты *Автомобиль* и *Гараж.*

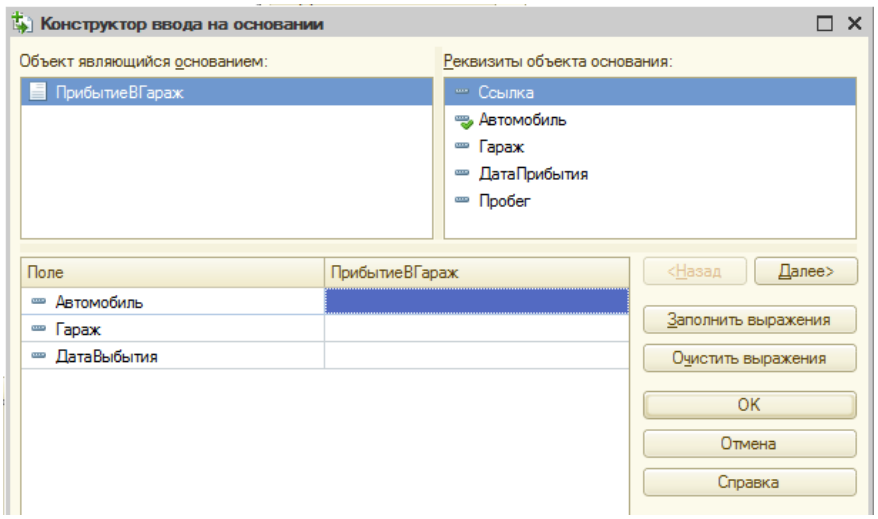


Рис. 4.1.67

В правом верхнем окне показаны реквизиты документа основания (*Прибытие в гараж*), а в нижнем окне реквизиты текущего документа *Выбытие из гаража.* При установке курсора на строку нужного поля нижнего в верхнем левом окне зеленым флажком будут отмечены реквизиты документа основания, тип которых совпадает с типом выделенного поля (см. рис. 4.1.67).

Чтобы совместить реквизит основного документа с реквизитом документа основания, необходимо выделить строку с нужным реквизитом в нижней таблице и дважды кликнуть на соответствующий реквизит левой верхней таблицы. Мы совместим реквизиты «Автомобиль – Автомобиль» и «Гараж – Гараж».

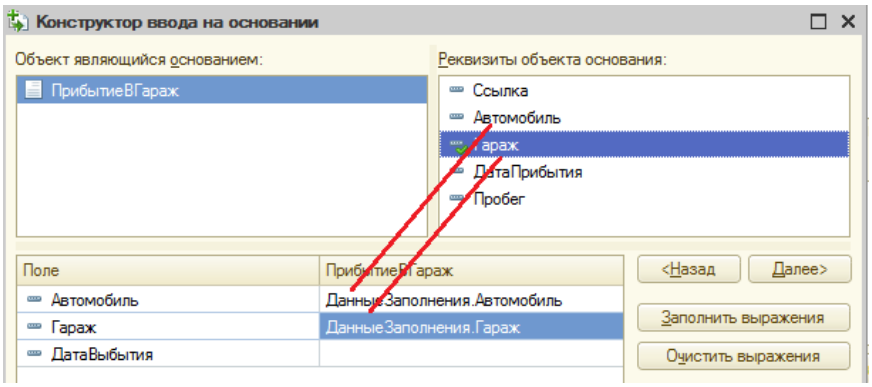


Рис. 4.1.68

Нажмите *«ОК»,* и процедура заполнения на основании будет создана автоматически.

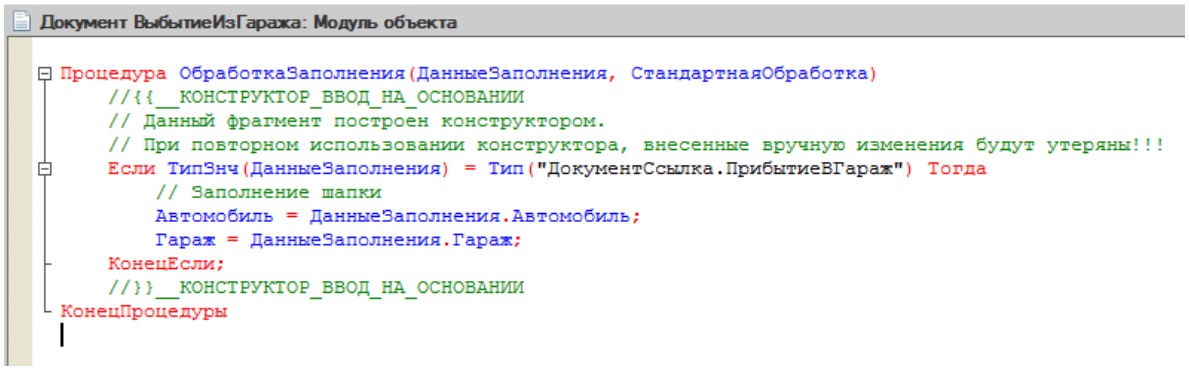


Рис. 4.1.69

Сохраните конфигурацию и создайте документ *Выбытие из гаража* на основании документа *Прибытие в гараж*.

Журнал документов

Следующий объект конфигурации, который мы рассмотрим, - это *Журнал документов*. ***Журнал документов***, как ясно из названия, предназначен для просмотра документов. Как Вы уже увидели, в списке каждого документа можно посмотреть только документ одного вида. Иногда очень полезно видеть документы разных видов в одном месте. Для этого и необходим журнал документов. Каждый документ может быть отнесен к тому или иному журналу, либо быть вообще без журнала. В отличие от справочников и документов, журнал не добавляет новой информации в систему, а служит только для отображения имеющейся информации.

Создайте журнал документов *Прибытие - Выбытие автомобилей*, в котором будут отображаться документы *Приезд в гараж* и *Выбытие из гаража*.

Для этого кликните правой кнопкой мышки по пункту *«Журналы документов»* в конфигурации и выберите пункт *«Добавить»*.

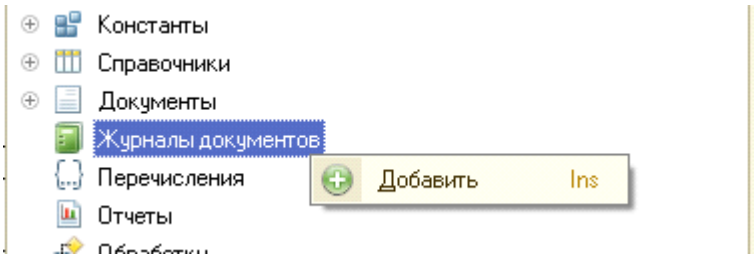


Рис. 4.1.70

Откроется форма журнала, где Вы назовете его *ПрибытиеВыбытиеАвтомобилей*.

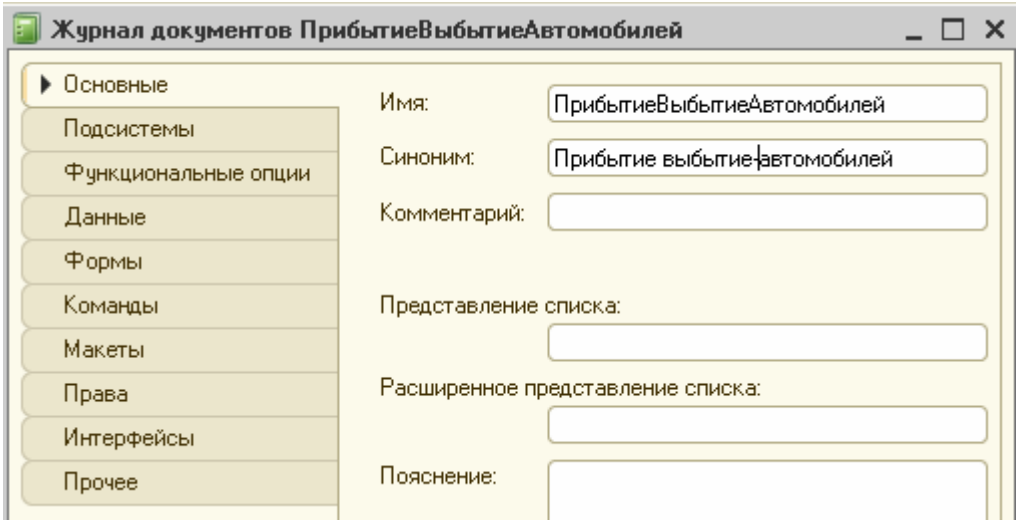


Рис. 4.1.71

Осталось задать регистрирующие документы. Для этого заходим в закладку *«Данные»*.

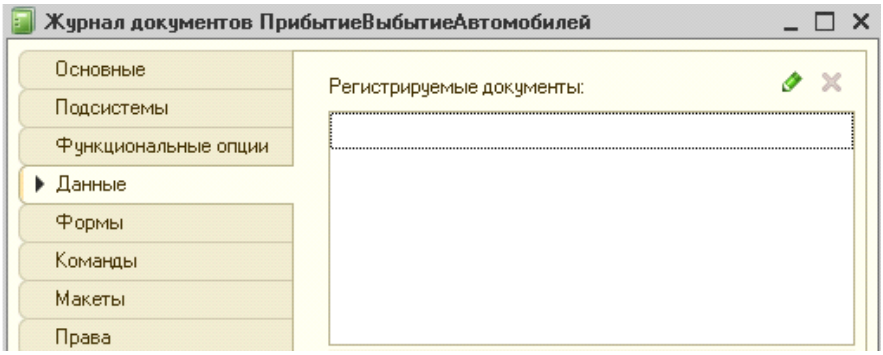


Рис. 4.1.72

Окошко *«Регистрирующие документы»* пустое, заполните его. Необходимо нажать на кнопку *«Редактировать элемент списка»* в виде карандашика, и в открывшемся окне выберите документы *ПрибытиеВГараж* и *ВыбытиеИзГаража*.

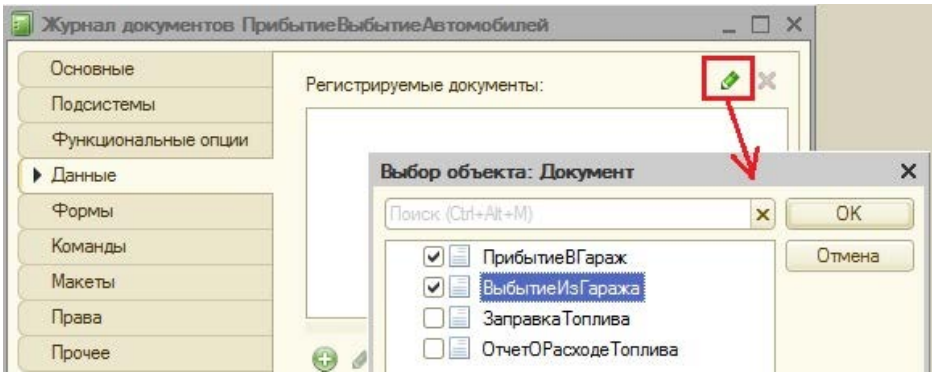


Рис. 4.1.73

Нажмите *«Закрыть»*. Журнал готов.

Сейчас запустим «1С:Предприятие» и посмотрим, как выглядят документы в нашем журнале (я надеюсь, Вы парочку уже создали).

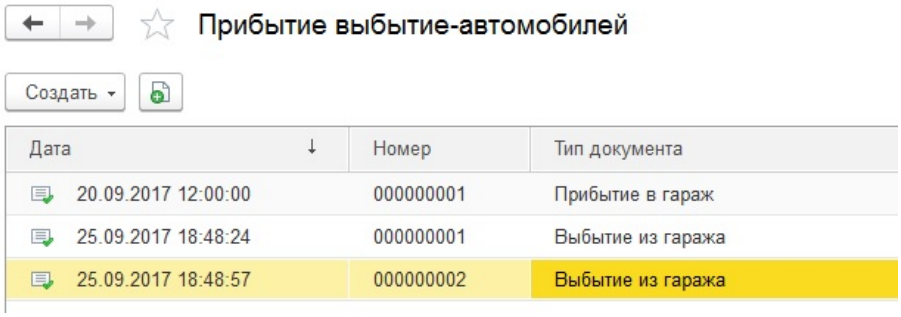


Рис. 4.1.74

Из формы журнала Вы видите, что у нас только номер, дата и тип документа, что, согласитесь, иногда бывает недостаточно. Добавим в журнал еще две колонки, это *Гараж* и *Автомобиль*. Для этого воспользуемся окном «Графы» на закладке конструктора журнала документов (см. рис. 4.1.75).

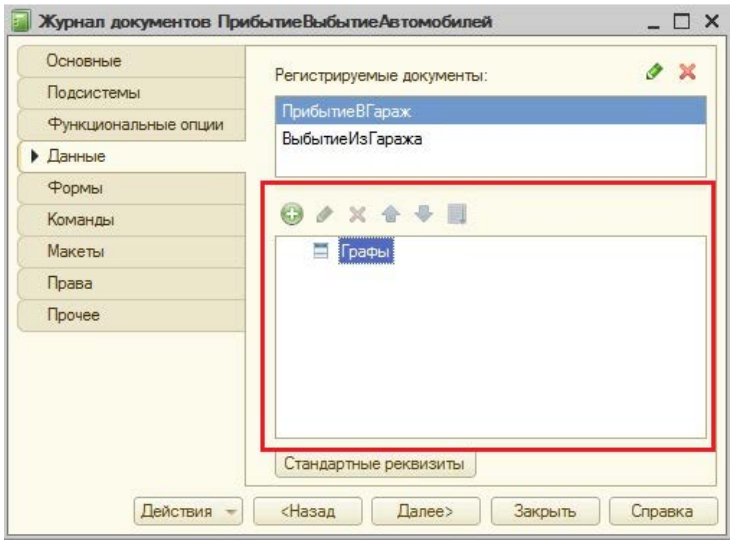


Рис. 4.1.75

Добавим новую графу в журнал документов. Для этого нажмем на кнопку «Добавить» окна «Графы».

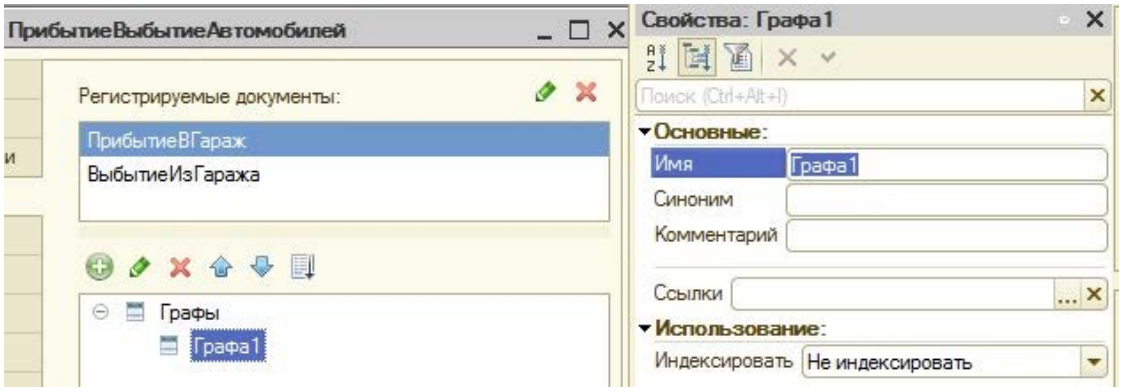


Рис. 4.1.76

Графа добавилась, теперь нам необходимо задать, какие поля документов будут отображаться в этой графе. Для этого в палитре свойств графы нужно нажать на кнопку «…» у свойства «Ссылки», после этого откроется окно, где нужно указать реквизиты документов журнала, которые должны отобразиться в данной графе (см. рис. 4.1.77).

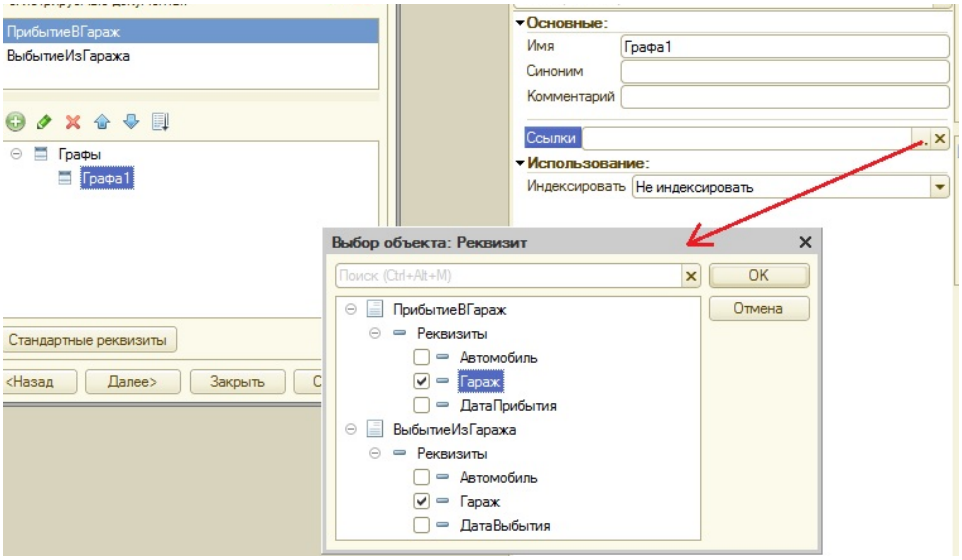


Рис. 4.1.77

Графу можно переименовать.

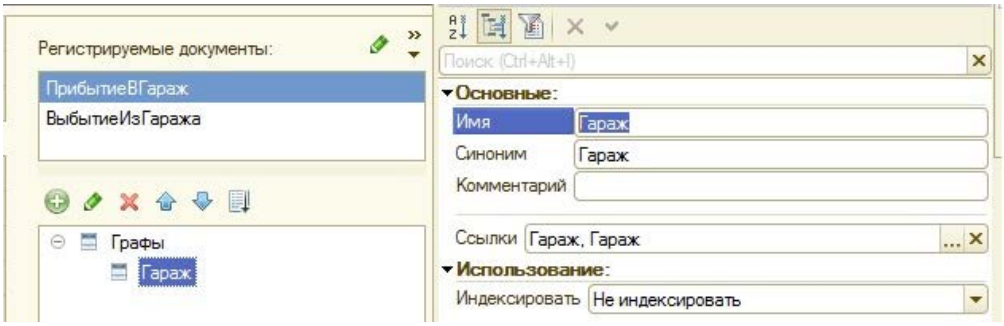


Рис. 4.1.78

После этих доработок журнал документов будет выглядеть гораздо интереснее:

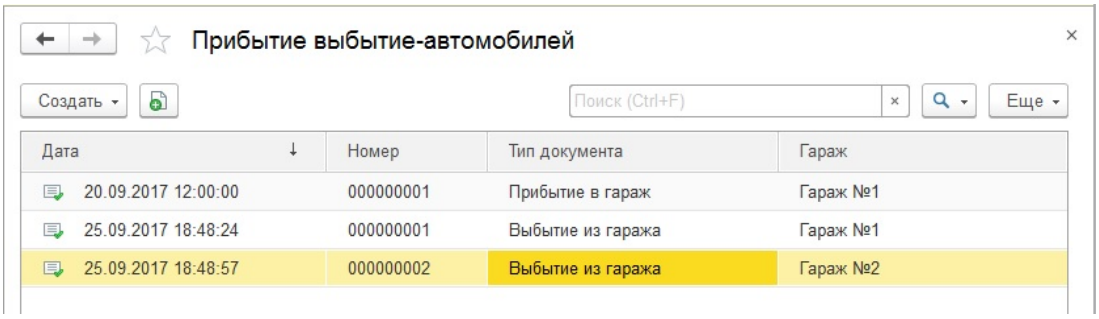


Рис. 4.1.79

Самостоятельно добавьте новую графу, чтобы журнал выглядел следующим образом:

Дата	Номер	Тип документа	Гараж	Автомобиль
20.09.2017 12:00:00	0000000...	Прибытие в гараж	Гараж №1	Автомобиль главного инженера
25.09.2017 18:48:24	0000000...	Выбытие из гаража	Гараж №1	Автомобиль главного директора
25.09.2017 18:48:57	0000000...	Выбытие из гаража	Гараж №2	Автомобиль главного инженера

Рис. 4.1.80

Перечисления

Следующий объект конфигурации - это *Перечисления*. ***Перечислениями*** называют объекты конфигурации, которые содержат в себе постоянные значения, не изменяемые в процессе работы с программой. Перечисления задаются на этапе конфигурирования, и их нельзя менять пользователю во время работы программы.

Создайте перечисление *Коробка передач* с двумя значениями: *Ручная* и *Автоматическая*. И добавим данный реквизит в справочник *Автомобили*.

Для этого кликаете правой клавишей мышки по пункту *«Перечисления»* и выбираете *«Добавить»*.

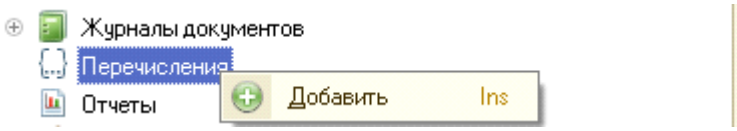


Рис. 4.1.81

У Вас откроется меню элемента перечисления. Назовите его *КоробкаПередач*.

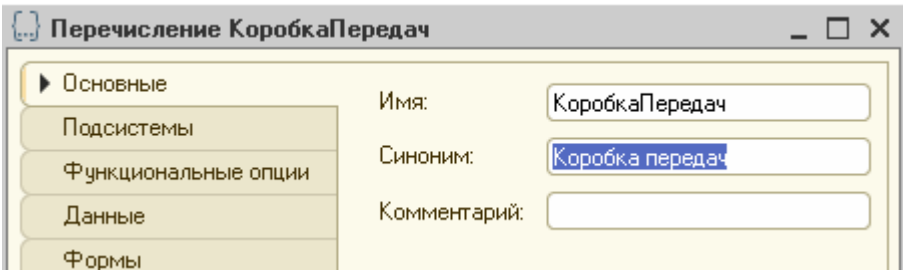


Рис. 4.1.82

Осталось создать значения Вашего перечисления, для этого переходите на закладку *«Данные»* и нажимаете *«Добавить»*.

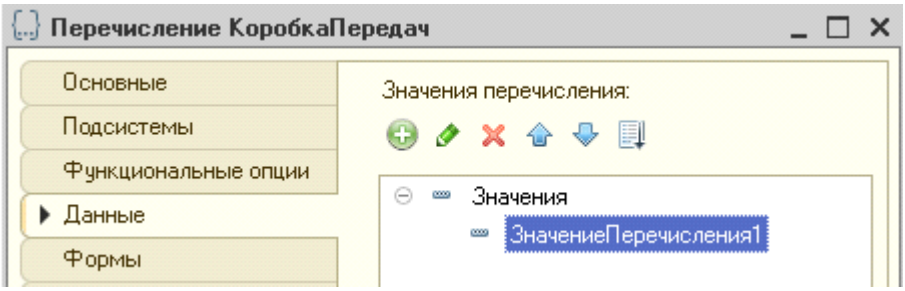


Рис. 4.1.83

Значение создалось, назовите его *Ручная*.

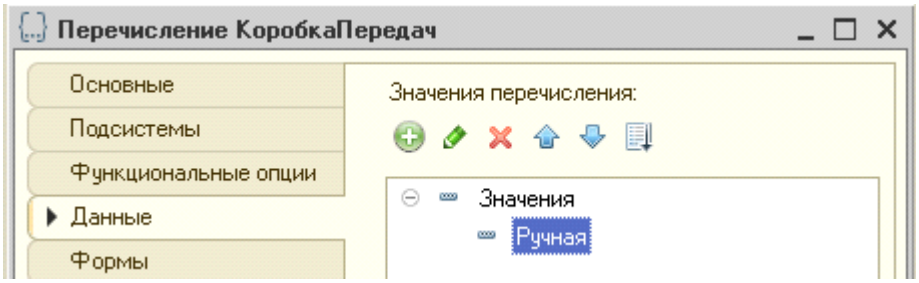


Рис. 4.1.84

Точно так же создайте второе значение - *Автоматическая*.

Теперь перейдите в Ваш справочник *Автомобили* и добавьте новый реквизит *Коробка передач,* тип которого будет *Ссылка на перечисление Коробка Передач*.

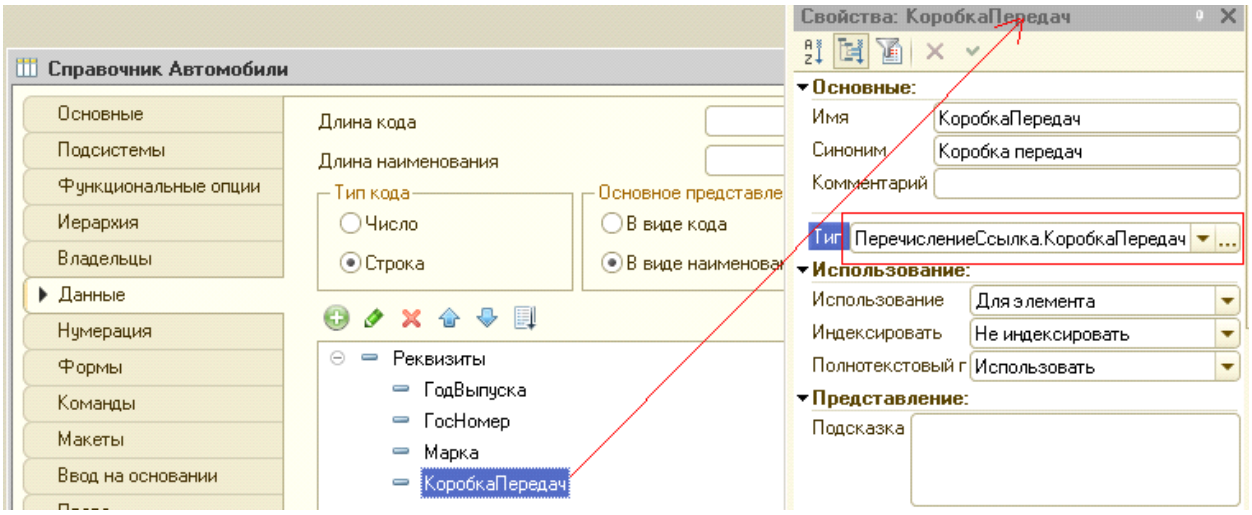


Рис. 4.1.85

В «1С:Предприятии» в уже созданных элементах справочника «Автомобили» заполните новый реквизит «Коробка передач».

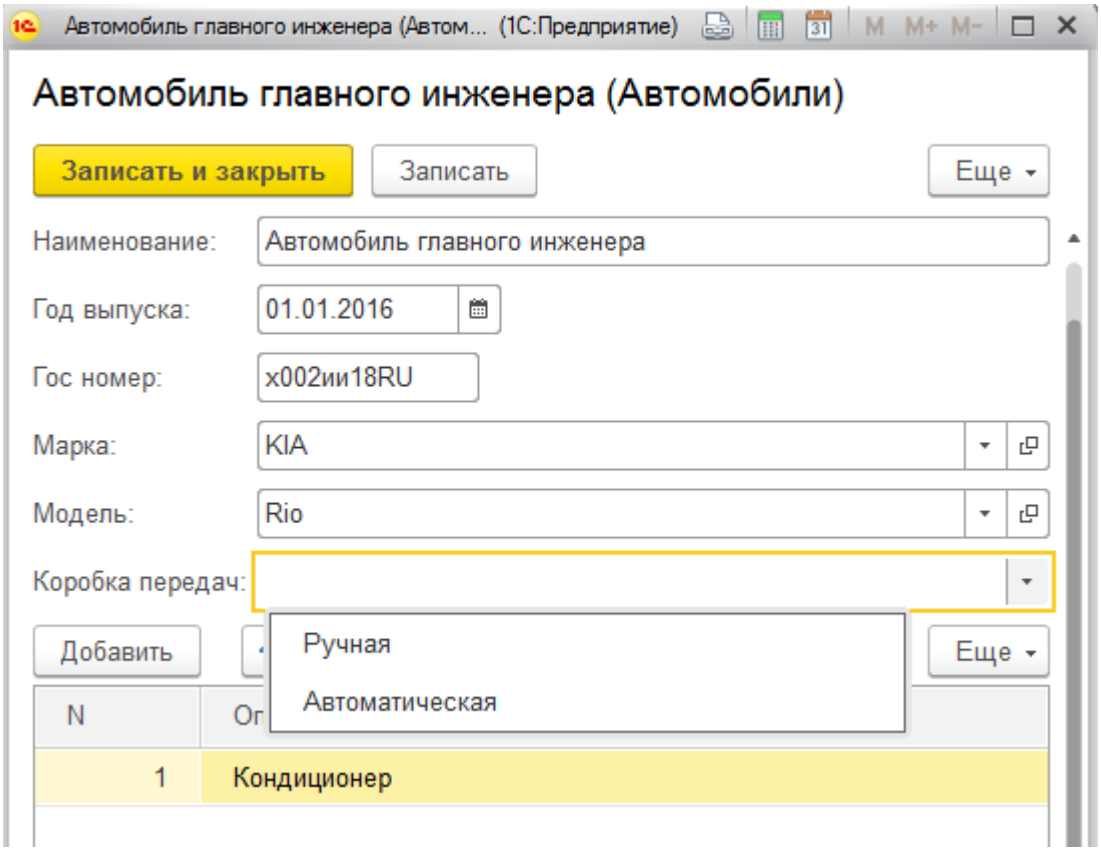


Рис. 4.1.86

Вы научились создавать в конфигураторе справочники, перечисления, документы и журналы документов. Теперь плавно перейдем к изучению регистров. Вся информация, которая вводится с помощью документов, должна быть отражена в регистрах, для того чтобы впоследствии можно было ее извлечь и проанализировать.

В конфигурации 1С можно создать четыре вида регистра, это: *Регистр накопления*, *Регистр сведений*, *Регистр бухгалтерии* и *Регистр расчета*. В этой книге мы рассмотрим два регистра: *Регистр накопления* и *Регистр сведений*.

Для новичков изучения принципов функционирования регистров накопления и сведений будет вполне достаточно, чтобы представлять, каким образом ведется учет в данной программе.

И первым делом мы рассмотрим регистр сведений. Более подробно о том, как работать с регистрами сведений, Вы узнаете в десятой главе. В этой главе мы изучим только моменты, связанные с конфигурированием регистра сведений.

Регистр сведений

Регистры сведений необходимы для хранения различной информации, которая может быть важна для прикладной области. Данная информация хранится в разрезе измерений, также она может изменяться во времени. *Регистры сведений,* информация которых изменяется во времени, называются *Периодическими*. Периодичность может быть разной, может быть периодичность в секунду, минуту, час и т.д. максимум - год.

Поскольку Вы уже изучили справочники, то у Вас должен возникнуть вопрос: чем отличается регистр сведений от справочника, и, самое главное, когда нужно использовать справочник, а когда регистр сведений.

Основное отличие в использовании справочников и регистров сведений в том, что в справочниках должны храниться объекты аналитики (контрагенты, номенклатура, автомобили, виды топлива и т.д.), а в регистре сведений должны храниться показатели для этих объектов. Например, это может быть цена на топливо у того или иного поставщика. Или гараж по умолчанию, где должен стоять автомобиль. Причем у регистров сведений осуществляется контроль уникальности в разрезе измерений (чего нет у справочников). К примеру, если у нас будет *непериодический* регистр сведений «Цены на топливо» с измерениями «Тип топлива» и «Поставщик» и одним ресурсом – цена, то мы не сможем создать две одинаковых записи с одними и теми же наборами измерений «Тип топлива» и «Поставщик».

Кроме измерений у регистра сведений существуют *Ресурсы* и *Реквизиты*. *Ресурс* должен хранить основную информацию регистра сведений, т.е. те данные, ради которых он создан, а *Реквизит* содержит дополнительную второстепенную информацию о записи (так же как и *Реквизит* у регистра накопления).

Регистры сведений бывают подчиненные регистратору, тогда записи в них будут создаваться при проведении документов. И независимые, тогда пользователь сам может создавать записи. В первом случае *Режим записи* устанавливается *Подчинение регистратору*, а во втором – *Независимый*.

Создадим в нашей конфигурации регистр сведений «*Основной гараж автомобиля»*, где измерение будет *Автомобиль* с типом *Ссылка на справочник Автомобили*, и ресурс: *Гараж*, этот регистр будет *непериодическим*.

Для этого кликаете правой клавишей мышки по пункту конфигурации *«Регистр сведений»,* в появившемся меню нажимаем *«Добавить»*.

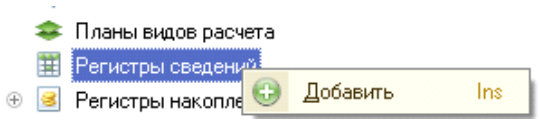


Рис. 4.1.87

Открывается форма регистра сведений, где Вы введите название *ОсновнойГаражАвтомобиля,* периодичность выберете *Непериодический*, а режим записи *Независимый*.

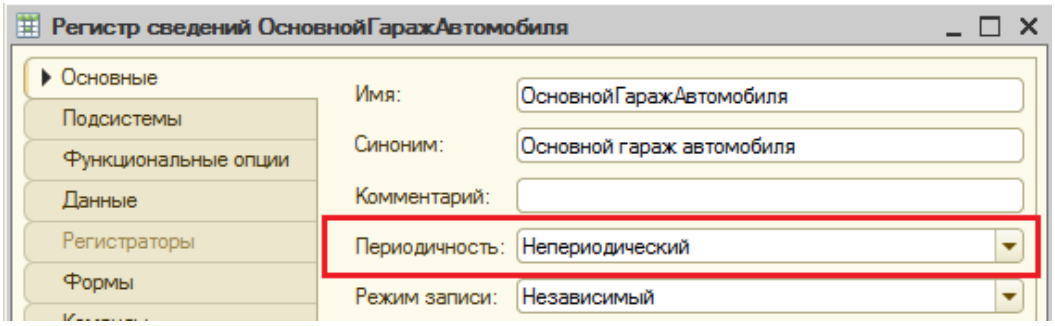


Рис. 4.1.88

Перейдите на закладку *«Данные»*, где необходимо создать измерение *Автомобиль*, тип которого будет *Ссылка на справочник Автомобили.* Это будет Ваше единственное измерение. Добавьте ресурсы *Гараж,* тип *Ссылка на справочник Гаражи*.

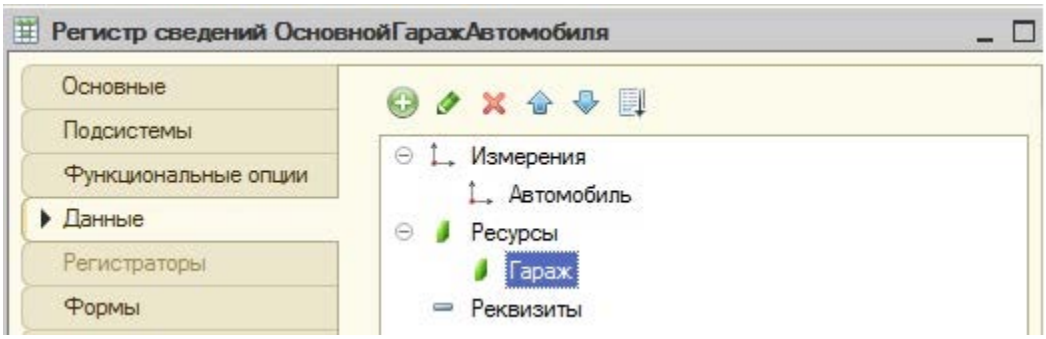


Рис. 4.1.89

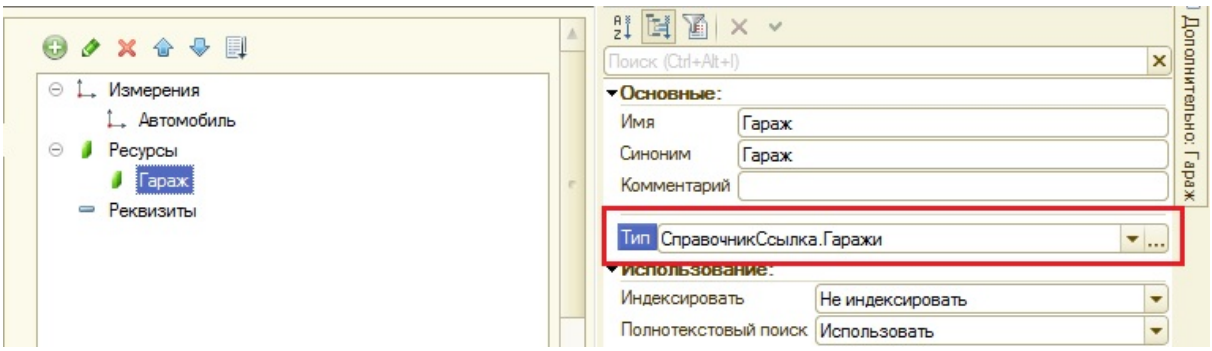


Рис. 4.1.90

Почему мы сделали только одно измерение, «Автомобиль»? Очевидно из названия, что у автомобиля может быть только *один* гараж. Как следствие не должно быть двух записей регистра сведений с одинаковыми автомобилями и разными гаражами. Мы задали только одно измерение, поэтому может быть только одна запись с конкретным автомобилем, поскольку у регистров сведений обеспечивается уникальность в разрезе измерений.

Поскольку у нас основной гараж будет устанавливаться для конкретного автомобиля, то логичнее привязать запись регистра сведений к конкретному элементу так, чтобы, когда мы удалим это элемент, удалилась и эта запись. Делается это при помощи свойства измерения регистра сведений *Ведущее*.

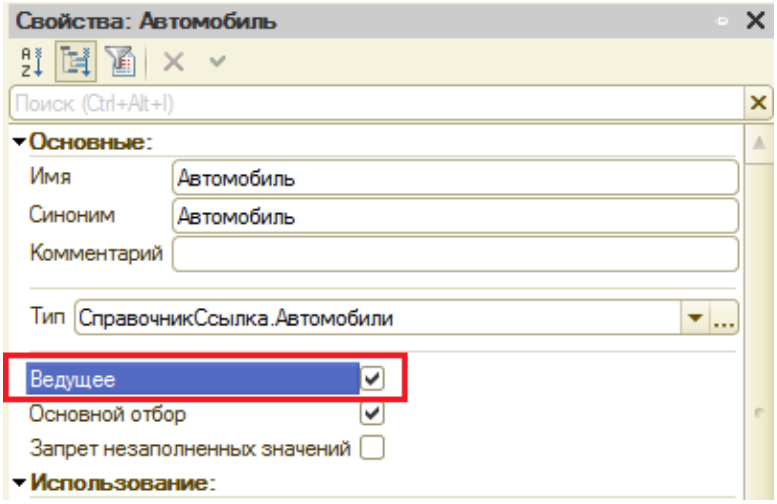


Рис. 4.1.91

Если это свойство измерения установлено, то записи регистра сведений, в которых в измерении содержится ссылка на определенный объект, будут существовать до тех пор, пока не будет удален этот объект.

И у этого свойства есть приятный интерфейсный бонус: в панели навигации формы элемента объекта будет команда на открытие формы записей регистра сведений, где будут перечислены только те записи регистра, у которых в измерении содержится ссылка на открытый объект.

Проверим это. Сохраним конфигурацию, обновим базу данных и зайдем в один из элементов справочника «Автомобили».

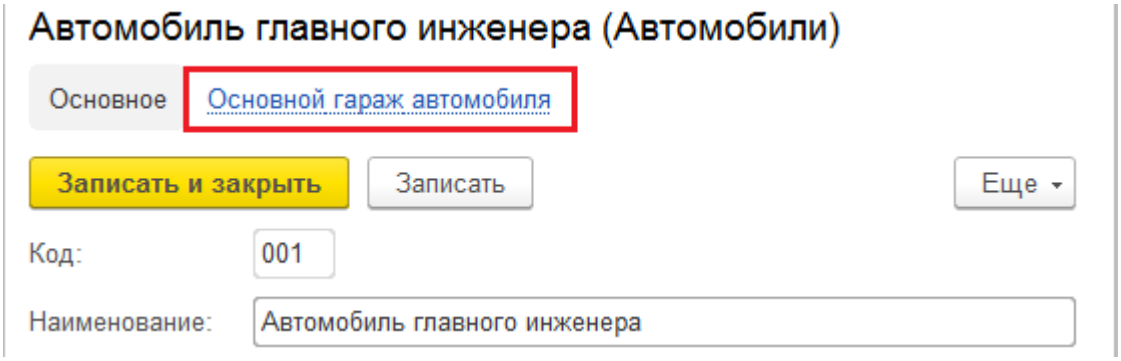


Рис. 4.1.92

Как видите, вверху элемента появилась гиперссылка «Основной гараж автомобиля», когда мы в неё перейдем, откроется форма записей регистра сведений «Основной гараж автомобиля», в которой мы можем добавить новую запись.

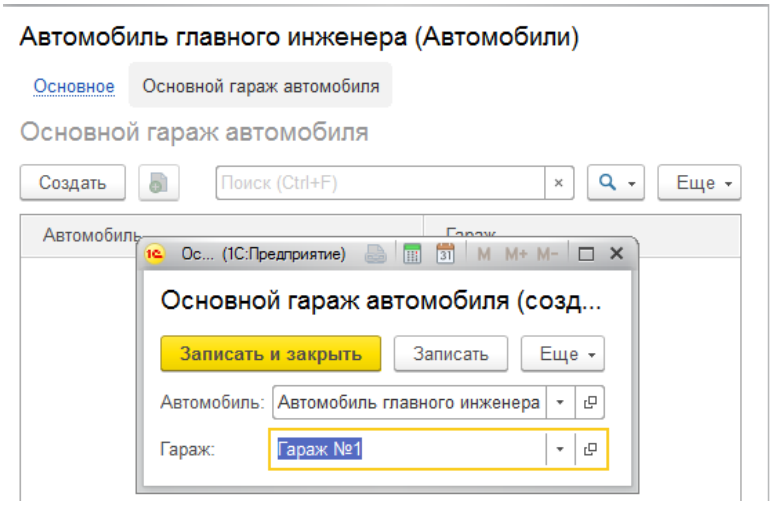


Рис. 4.1.93

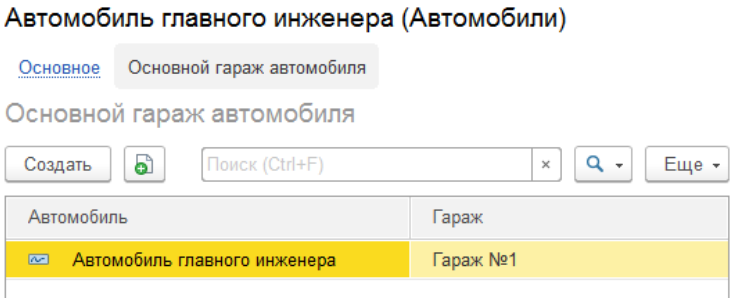


Рис. 4.1.94

И мы не сможем добавить еще один основной гараж автомобиля. Выйдет ошибка.

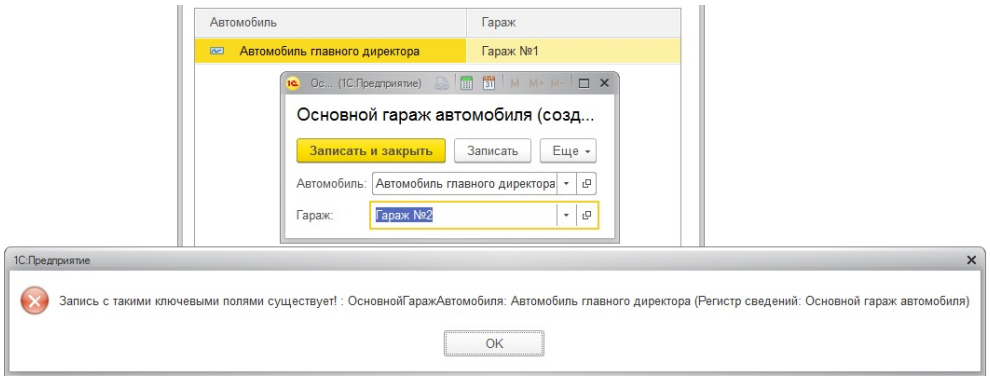


Рис. 4.1.95

Перейдем к периодическим регистрам сведений. Для этого создадим периодический регистр сведений «Цены на топливо» с периодичностью – день.

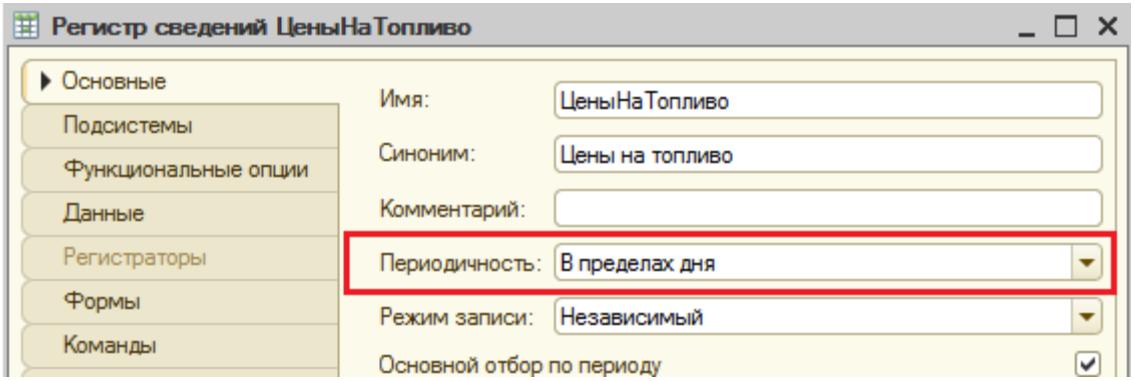


Рис. 4.1.96

С двумя измерениями *«ТипТоплива»* (ссылка на справочник «ТипыТоплива») и «ПоставщикТоплива» (ссылка на справочник «ПоставщикиТоплива», создайте самостоятельно (без реквизитов, длина кода 3, длина наименования 100)). И одним ресурсом – цена (число (10,2)).

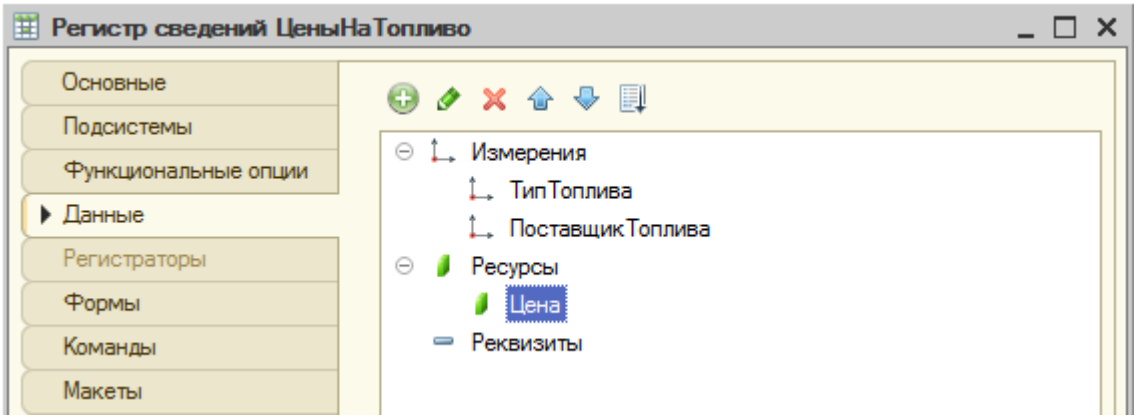


Рис. 4.1.97

Сохраним конфигурацию, обновим базу данных и попробуем позаполнять наш новый регистр сведений.

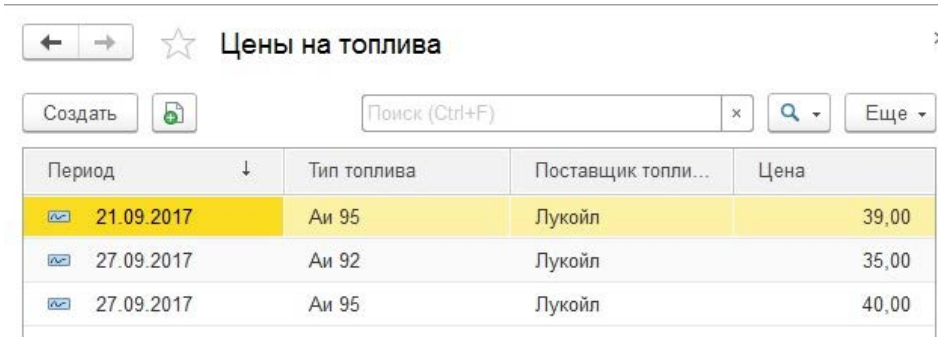


Рис. 4.1.98

Сейчас мы сможем заводить одной датой разные комбинации измерений, и одну и ту же комбинацию измерений разными датами, но не сможем завести на одно число две разные цены для определенного поставщика и определенного типа топлива.

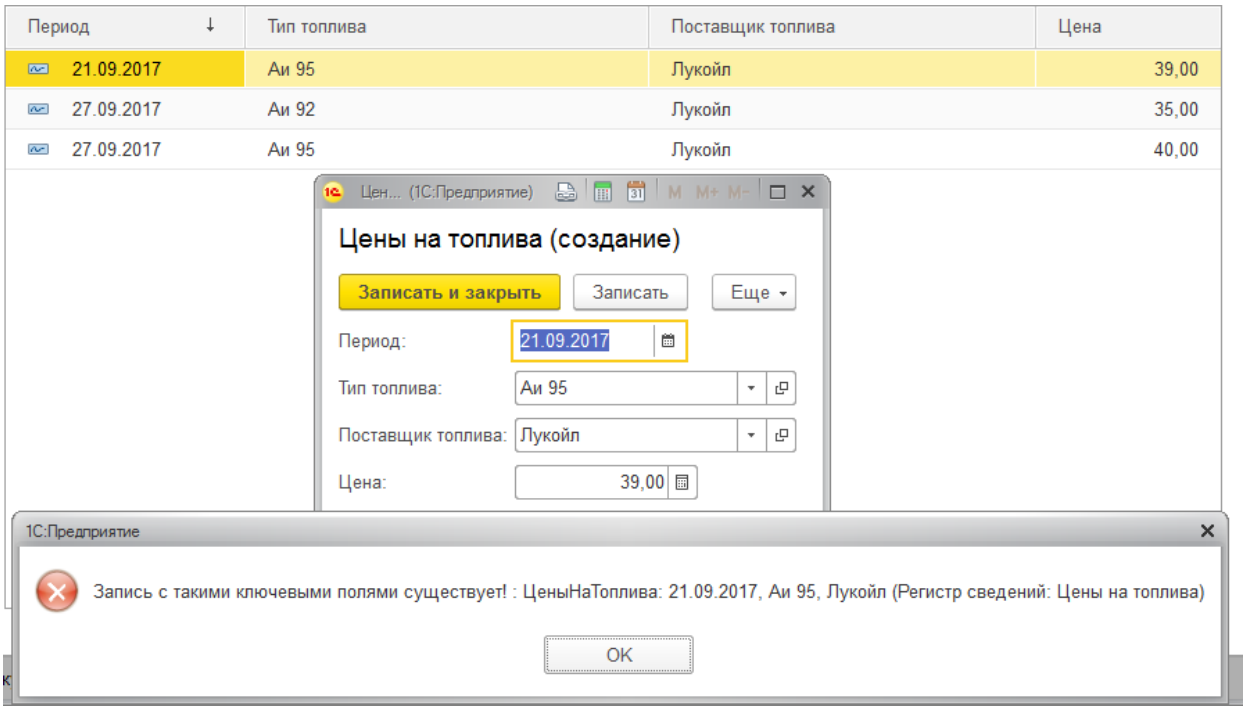


Рис. 4.1.99

А если цена будет по дню меняться несколько раз? Как быть? Можно создать более маленькую периодичность (секунда). А можно сделать режим записи – *подчиненный регистратору*, а периодичность – *по позиции регистратора*.

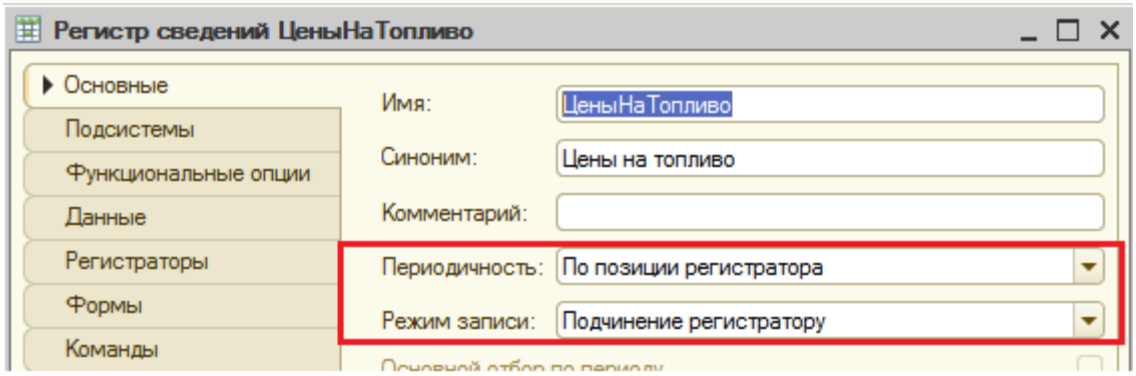


Рис. 4.1.100

В этом случае мы сможем создавать сколько угодно регистраторов, пусть даже с одним и тем же временем, и наш регистр будет нормально заполняться (удалите все созданные ранее записи регистра сведений «Цены на топливо»).

Для этого создадим новый документ «*Установка цен на топливо*» со следующими данными и настройками.

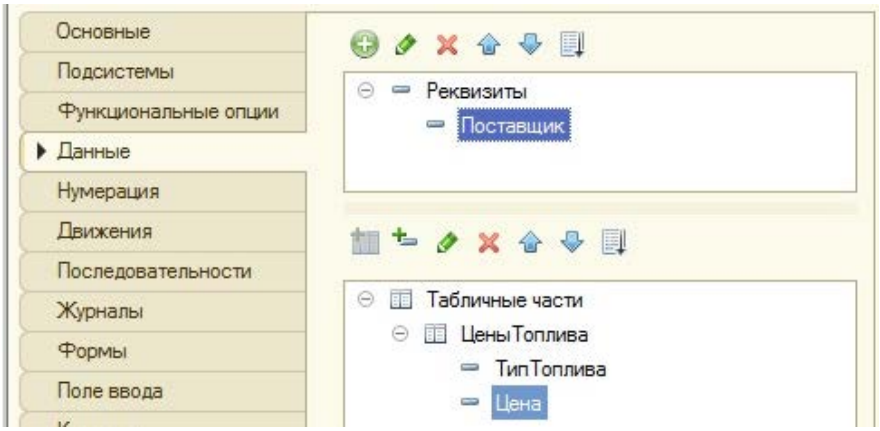


Рис. 4.1.101

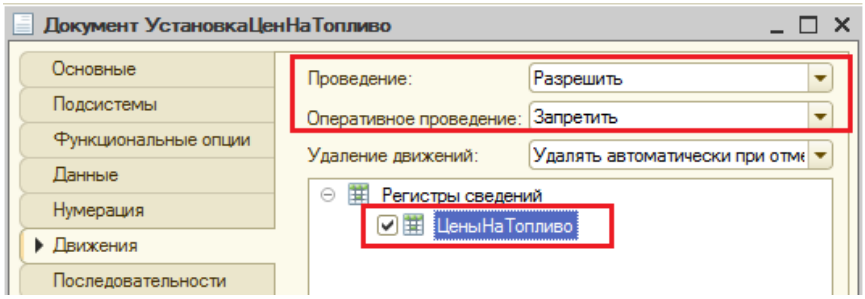


Рис. 4.1.102

Не забудьте поставить везде, где нужно ссылочные типы (определите это самостоятельно).

Более подробно работу с документами будем проходить в шестой главе.

Сейчас при помощи конструктора движений создадим обработку проведения по нашему регистру. Чтобы он открылся, нажмите на кнопку «Конструктор движений» в закладке «Движения».

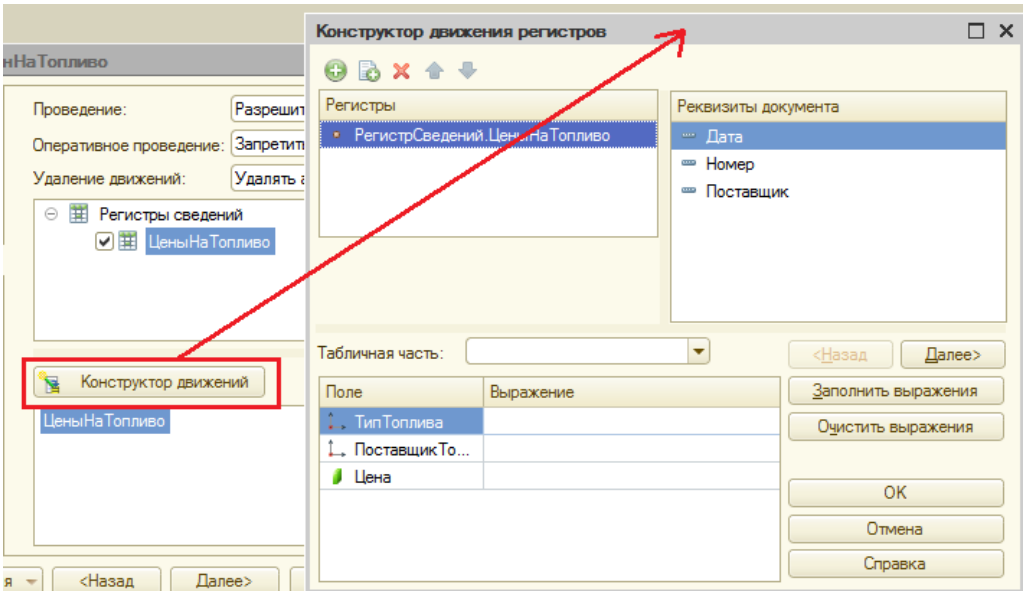


Рис. 4.1.103

Наши данные по ценам находятся в табличной части документа, поэтому используем поле конструктора «Табличная часть», чтобы заполнение происходило из неё. Выберем в этом поле нашу единственную табличную часть.

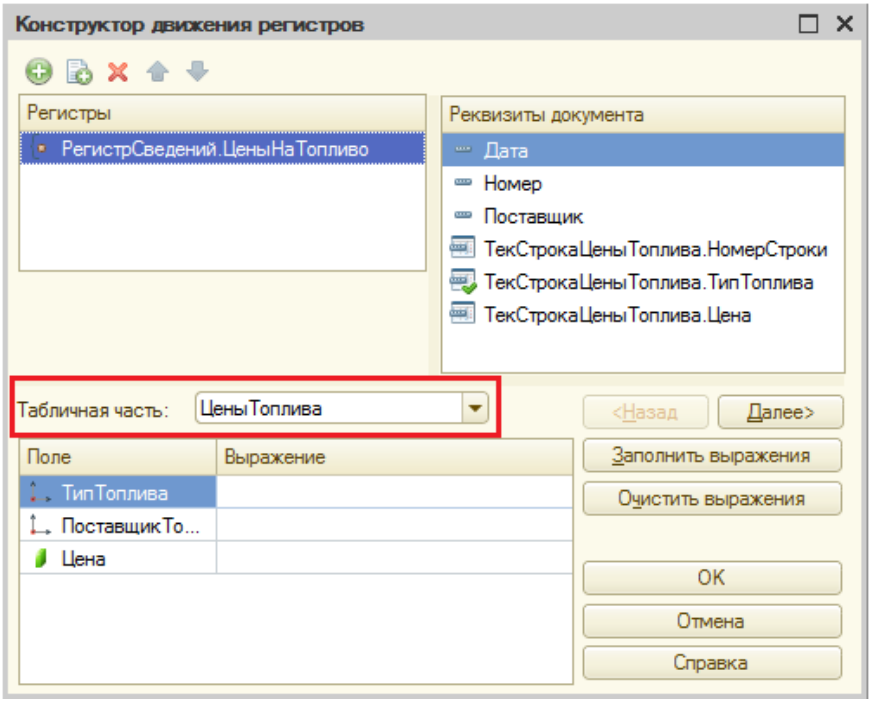


Рис. 4.1.104

Подберем в измерение «ТипТоплива» и ресурс «Цена» соответствующие реквизиты из табличной части, а в измерение «ПоставщикТоплива» реквизит «Поставщик» из шапки.

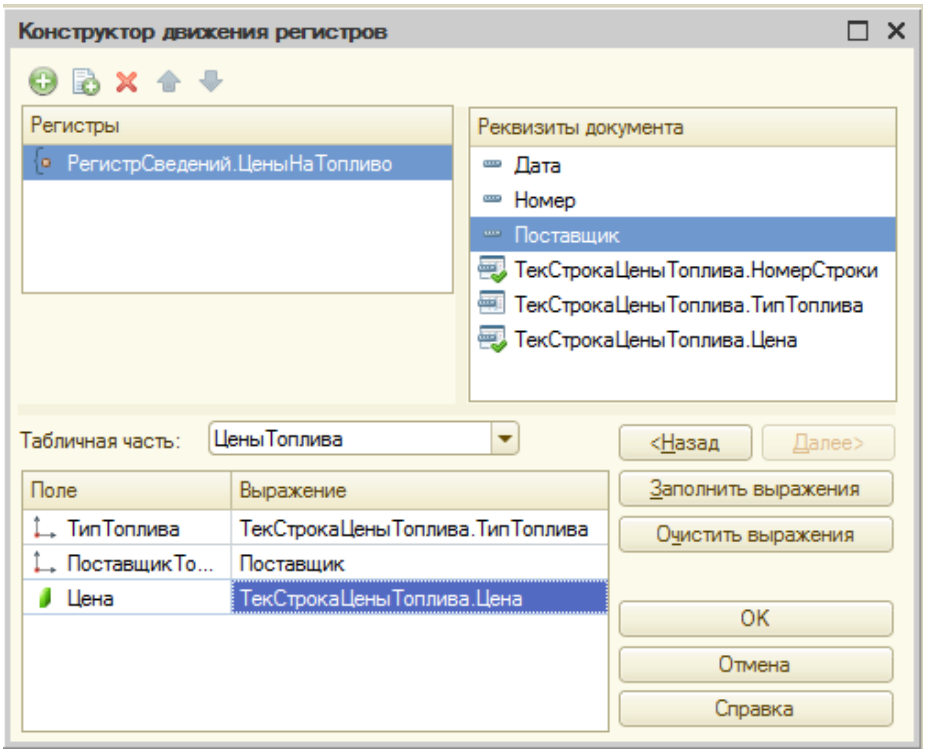


Рис. 4.1.105

Все сделали, теперь нажмем кнопку «ОК» и в модуле документа «Установка цен» автоматически будет создана процедура.

Документ УстановкаЦенНаТопливо: Модуль объекта

```
Процедура ОбработкаПроведения(Отказ, Режим)
    //{{__КОНСТРУКТОР_ДВИЖЕНИЙ_РЕГИСТРОВ
    // Данный фрагмент построен конструктором.
    // При повторном использовании конструктора, внесенные вручную изменения будут утеряны!!!

    // регистр ЦеныНаТопливо
    Движения.ЦеныНаТопливо.Записывать = Истина;
    Для Каждого ТекСтрокаЦеныТоплива Из ЦеныТоплива Цикл
        Движение = Движения.ЦеныНаТопливо.Добавить();
        Движение.Период = Дата;
        Движение.ТипТоплива = ТекСтрокаЦеныТоплива.ТипТоплива;
        Движение.ПоставщикТоплива = Поставщик;
        Движение.Цена = ТекСтрокаЦеныТоплива.Цена;
    КонецЦикла;

    //}}__КОНСТРУКТОР_ДВИЖЕНИЙ_РЕГИСТРОВ
КонецПроцедуры
```

Рис. 4.1.106

Ничего в ней не трогаем, сохраняем конфигурацию и обновляем базу данных. Попробуем создать несколько одинаковых документов с одинаковыми поставщиками, с одинаковыми типами топлива, но с разными ценами.

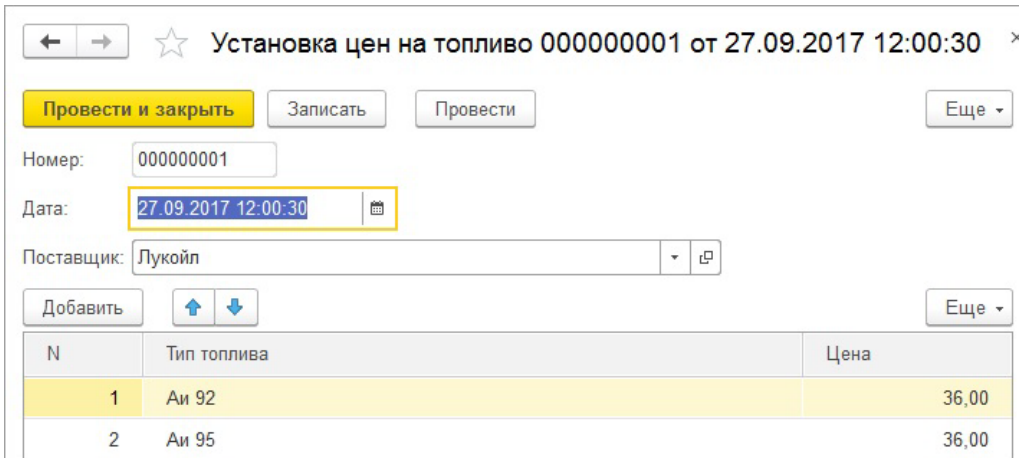


Рис. 4.1.107

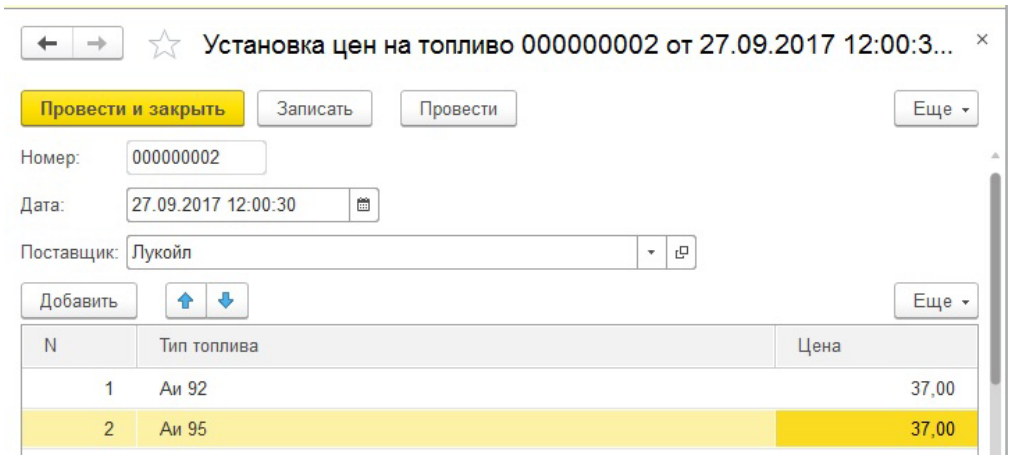


Рис. 4.1.108

Они прекрасно проведутся. Посмотрим, как будут выгладить записи в регистре сведений. В этот регистр мы зайдем через команду «Все функции», которая включается в параметрах.

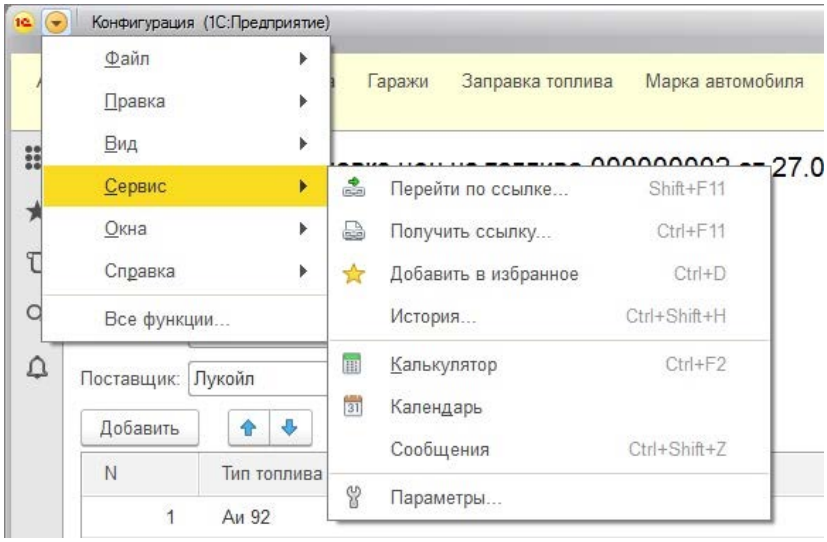


Рис. 4.1.109

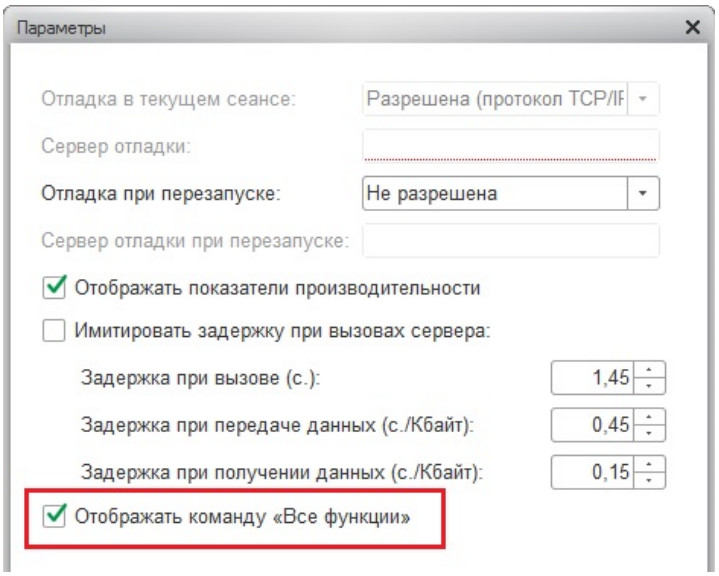


Рис. 4.1.110

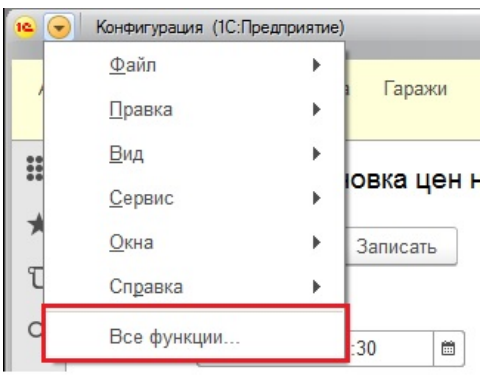


Рис. 4.1.111

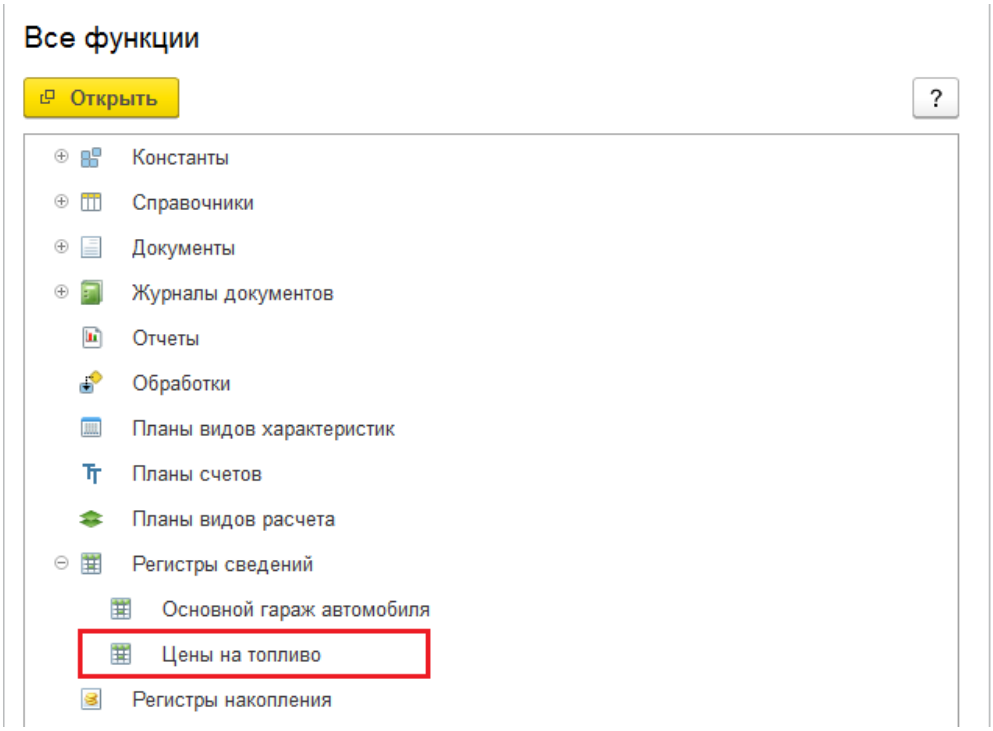


Рис. 4.1.112

Период	Регистратор	Н.	Тип т...	Поставщик топлива	Цена
27.09.2017 12:00:30	Установка цен на топливо 000000001 от ...	1	Аи 92	Лукойл	36,00
27.09.2017 12:00:30	Установка цен на топливо 000000001 от ...	2	Аи 95	Лукойл	36,00
27.09.2017 12:00:30	Установка цен на топливо 000000002 от ...	1	Аи 92	Лукойл	37,00
27.09.2017 12:00:30	Установка цен на топливо 000000002 от ...	2	Аи 95	Лукойл	37,00

Рис. 4.1.113

Как видите, у нас получилось несколько одинаковых записей с разными ценами. Почему такое стало возможным? Ведь «Период», по сути, тоже измерение, и по нему тоже должен вестись контроль уникальности записей. Потому что мы периодичность установили «По позиции регистратора», а позиция регистратора это не только дата документа, это *момент времени*, в который входит дата и время документа, а также ссылка на сам документ. У двух разных документов будет два разных момента времени, поэтому и возможна такая запись регистра сведений, как на рис. 4.1.113.

Регистр накопления

Регистр накопления хранит в себе данные о движении различных материальных величин, это может быть: прибытие и выбытие материалов, оборот денежных средств и т.п.

Вся информация о движении хранится в разрезе измерений, а сама информация об учете (количество, сумма и т.п.) хранится в ресурсах. Помимо этого есть реквизиты, в которых хранится прочая справочная информация.

Основная функция регистра накопления заключается в суммировании ресурсов, т.е. если какая-либо информация хранится в регистре накопления, то Вы можете в любой момент извлечь сумму всех ресурсов по любым измерениям, которые Вам интересны.

Регистр накопления не может существовать без *Регистратора*, т.е. документа, который при проведении создает записи данного регистра.

Регистры накопления бывают двух видов. Это *Оборотные регистры* и *Регистры остатков*. *Оборотные регистры* просто фиксируют наличие движения по данному регистру, без разницы, был ли приход или расход. *Регистры остатков* фиксируют движения двух видов: приход и расход.

Ярким примером сущности, которая может храниться в оборотном регистре, является прибыль: она может только появиться, нам интересен только сам факт наличия прибыли, к примеру, сколько прибыли было за предыдущий месяц. Поэтому если разрабатывать регистр накопления, который вел бы учет прибыли, то он должен быть оборотным. В нашей конфигурации в оборотном регистре будет храниться пробег автомобиля. Пробег не может «прийти» или «уйти», он всегда накапливается, и нам нужно знать, какой пробег у автомобиля был за неделю, месяц, год.

Другое дело товар: он приходит и уходит, и нам необходимо знать остатки товара на складе или в организации, поэтому регистр накопления, который ведет учет товара, должен иметь вид регистра остатков. В нашем случае мы будем вести учет топлива: сколько заправили топлива в автомобиль, и сколько его автомобиль израсходовал.

Это общая информация о регистрах. Теперь перейдем к их строению. Разберем сначала, что такое *Измерения*. Часто информацию нужно хранить в нескольких разрезах.

Например, нам необходима информация о наличии топлива в автомобилях. Само по себе топливо уже будет измерение, но просто сведения о топливе будут неполными. Нам необходимо знать, куда именно заправлено наше топливо, поэтому мы вводим еще одно измерение – автомобиль. Пусть топливо будет *Аи95* и *Аи92* (измерение *Топливо*), а автомобиль – *Директора* и *Гл. инженера* (измерение *Автомобиль*). Смотрим на таблицу 4.1.1*.

Топлива	**Автомобиль**	**Количество, л.**
Аи95	Директора	1
Аи92	Гл. инженер	2
Аи95	Гл. инженер	5

Табл. 4.1.1

* упростим, что разные виды топлива можно заправлять в один автомобиль, и вести по ним учет

Согласно таблице 3.1.1, топливо типа *Аи95* заправлено в автомобили *Директора* и *Гл. инженера*, всего топлива этого вида заправлено 6 литров. А в автомобиль *Гл. инженера*

заправлено топливо *Au92* и *Au95*, всего *7* литров. Таким образом, мы подошли к важной функции измерений - возможности суммировать нужный ресурс по любому значению измерения.

Сейчас мы создадим регистр накопления. Это будет регистр оборотов *ПробегАвтомобиля*, у него будет одно измерение - *Автомобиль*, соответствующего типа, и один ресурс – *Пробег* типа число (10,0).

Для этого кликаете правой кнопкой мышки по пункту *«Регистры накопления»* и нажимаете *«Добавить».*

Рис. 4.1.114

Откроется форма регистра накопления. Введите наименование, синоним. Вид регистра установите *Обороты.*

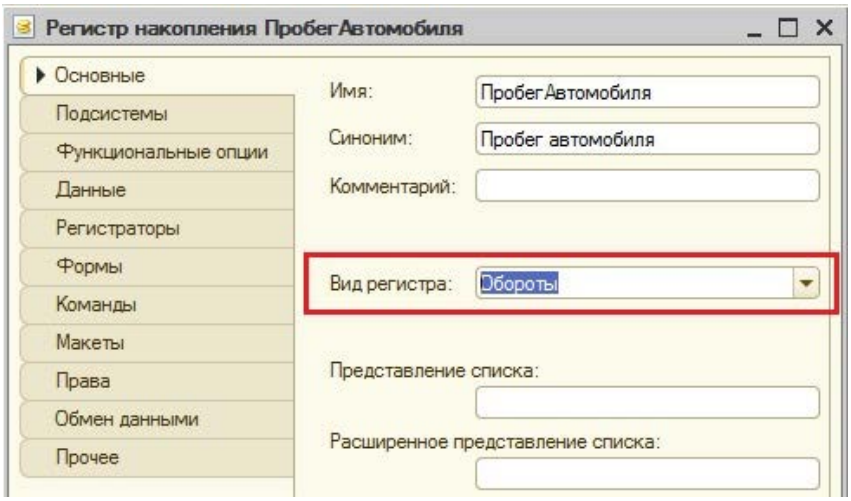


Рис. 4.1.115

После переходите на закладку *«Данные»*, где введите информацию об измерениях и ресурсах.

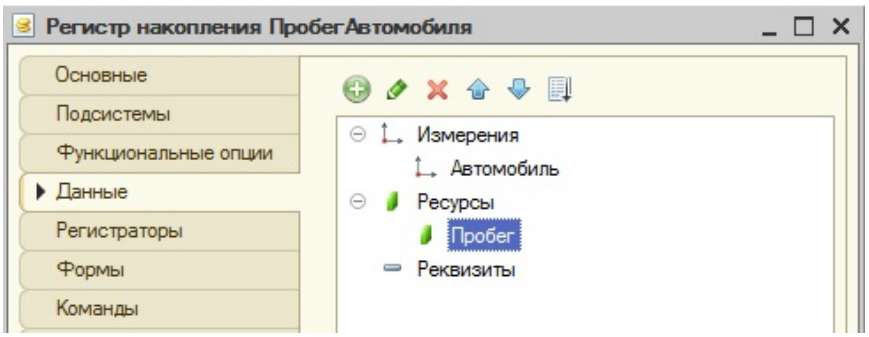


Рис. 4.1.116

Следующим шагом необходимо отметить те документы, которые будут делать движения по данному регистру. Переходите на закладку *«Регистраторы»* и ставьте флажки напротив документа «Прибытие в гараж», именно этим документом будет фиксироваться пробег.

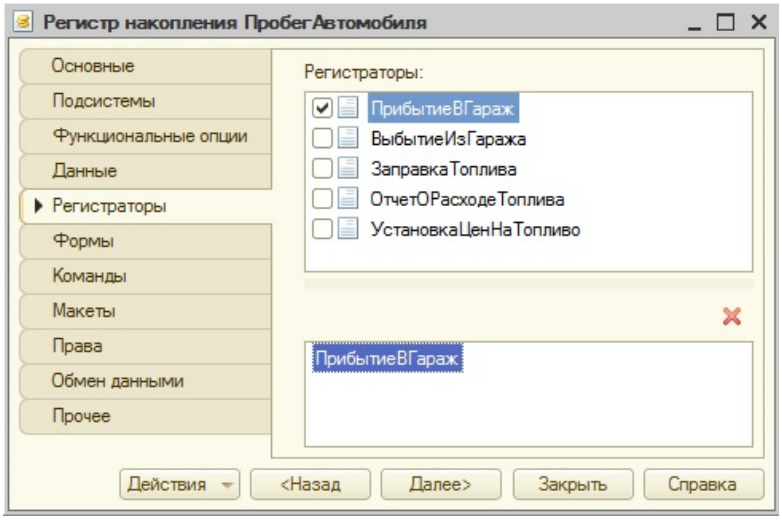


Рис. 4.1.117

В реквизиты документа «Прибытие в гараж» добавьте реквизит *Пробег*, с точно таким же типом, как и у ресурса *Пробег* вновь созданного регистра накопления (число (10,0)).

Для удобства и скорости можете этот ресурс просто «скопировать», а потом «вставить» в реквизиты документа «Прибытие в гараж», или вообще «перетащить»

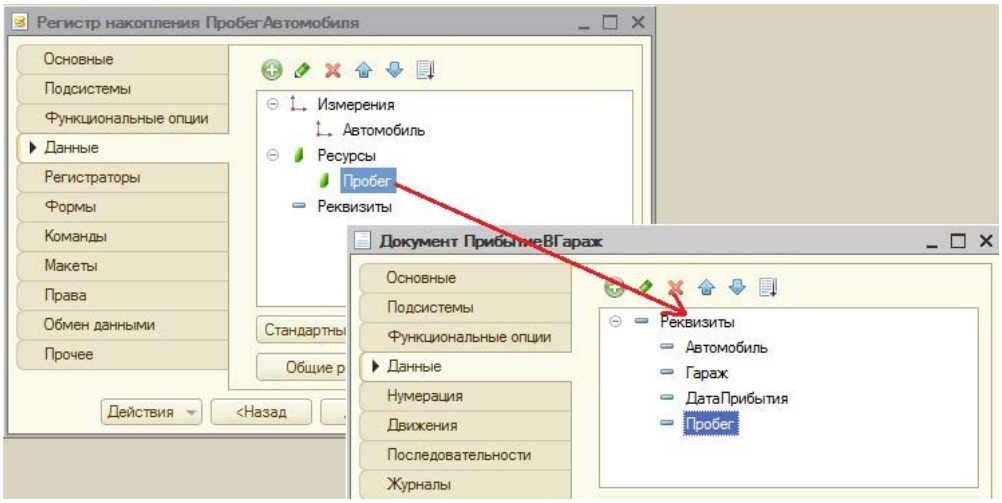


Рис. 4.1.118

Закройте конструкторы регистра и документа.

Теперь необходимо прописать сами движения, которые должны возникнуть при проведении документа.

Для этого в дереве документов выделяете нужный Вам документ (это будет *Прибытие в гараж*), кликаете правой кнопкой мыши и находите в меню подменю *«Конструкторы»*, по нему перейдите в *«Конструкторы движений»* и запустите конструктор.

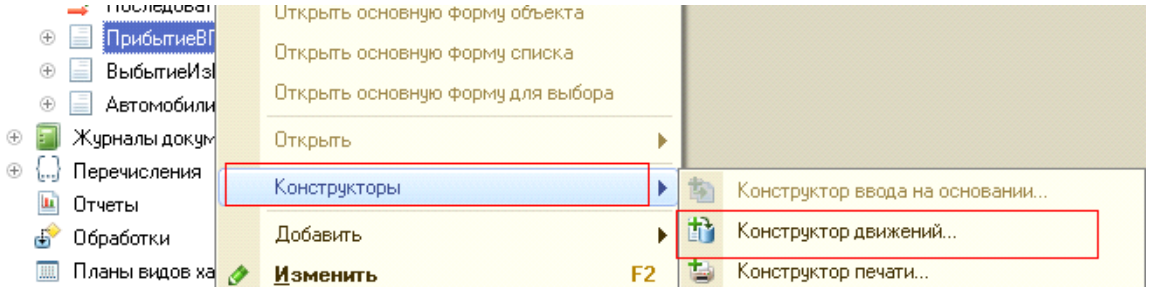


Рис. 4.1.119

Открылась соответствующая форма, в которой Вы проставите соответствие измерений и ресурсов регистра с реквизитами документа.

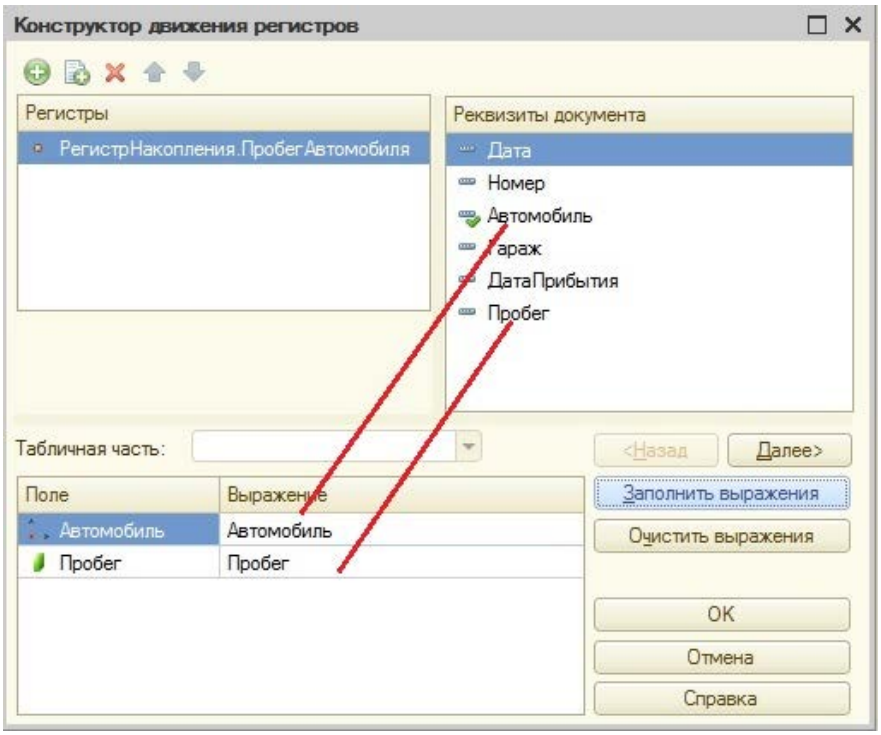


Рис. 4.1.120

Нажимаем кнопку *«ОК»*, и будет создана обработка проведения с соответствующим кодом. Пока не будем разбирать этот код, работу с набором записей регистров накоплений Вы будете проходить в десятой главе.

Сохраним конфигурацию, обновим базу данных и зайдем в какой-нибудь документ «Прибытие в гараж», заполним у него пробег и проведем. А после проведения, посмотрим, какие движения сделал этот документ по регистру накопления «Пробег автомобиля» (зайдите через «Все функции»).

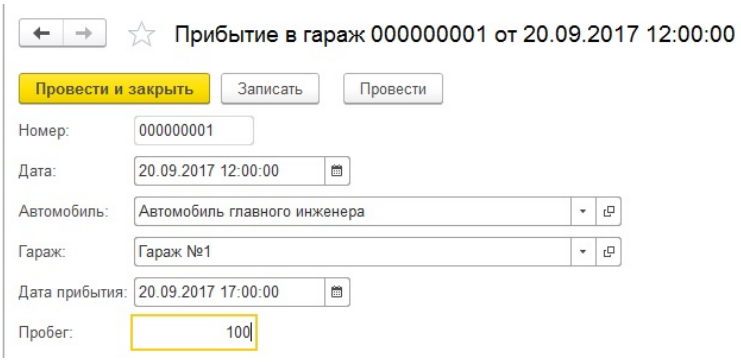


Рис. 4.1.121

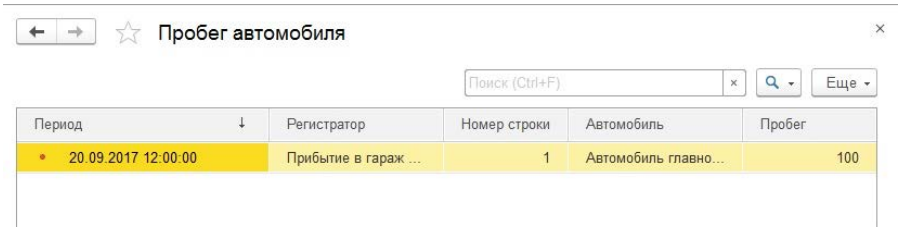


Рис. 4.1.122

Сейчас создайте регистр накопления *ТопливоВАвтомобилях*. Это будет регистр *остатков*, у него будет два измерения – *ТипТоплива* и *Автомобиль*, соответствующих типов, и один ресурс – *Количество* типа число (10,2).

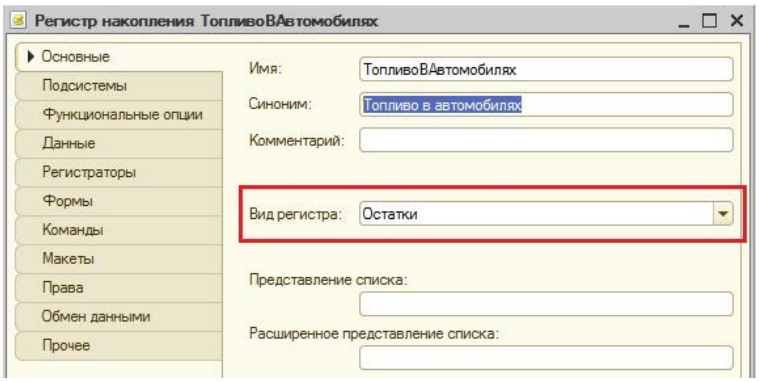


Рис. 4.1.123

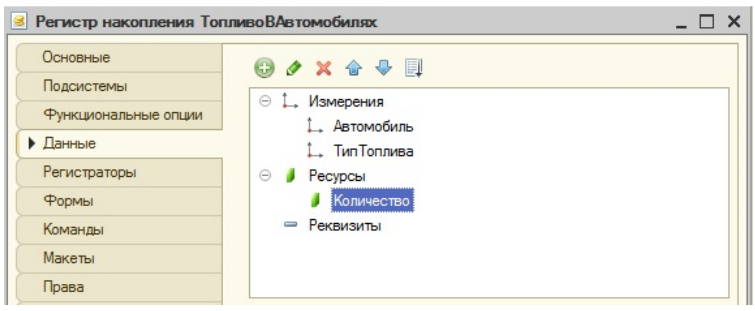


Рис. 4.1.124

Регистраторами данного регистра будут документы «Заправка топлива» и «Отчет о расходе топлива».

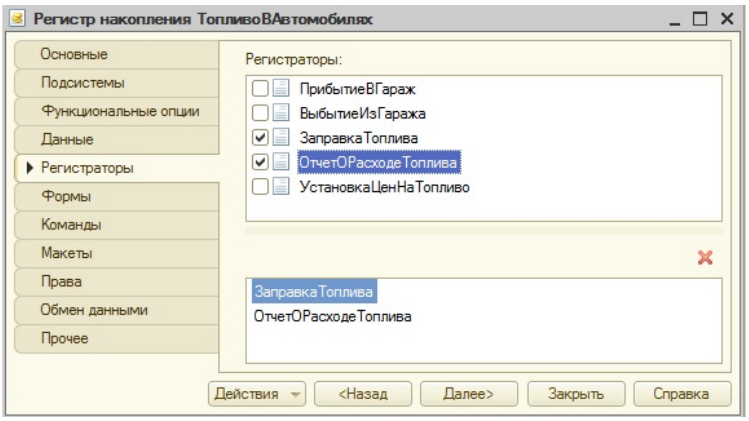


Рис. 4.1.125

Закройте конструктор регистра, сейчас пропишем движения, которые должны возникнуть при проведении документа «Заправка топлива». Уже знакомым Вам способом (рис. 4.1.119) откройте конструктор движений документа «Заправка топлива»

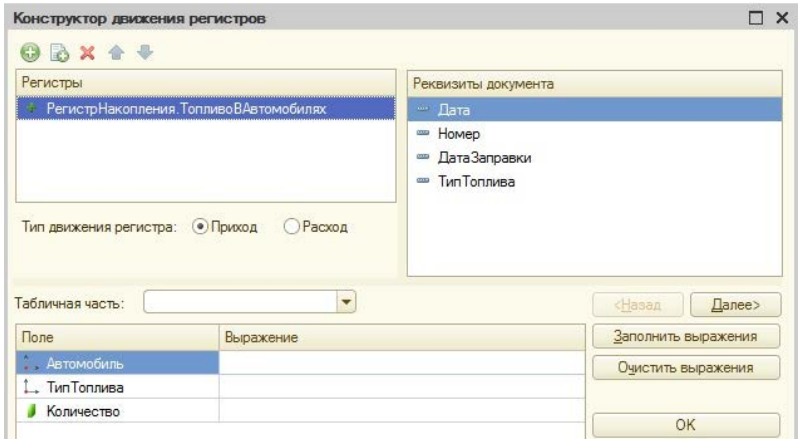


Рис. 4.1.126

У этого документа данные для движений по регистру накопления будем брать из табличной части *Автомобили*. Поэтому в поле «Табличная часть» конструктора выберем «Автомобили».

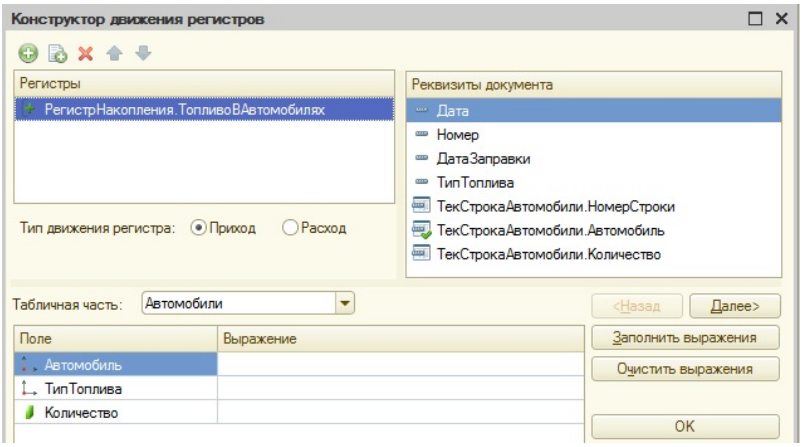


Рис. 4.1.127

В правом верхнем поле сразу же появились реквизиты из табличной части документа. Сейчас необходимо проставить соответствия между измерениями и ресурсами регистра накопления и реквизитами шапки и табличной части документа.

Подсказка: сопоставление можно выполнить автоматически, нажав на кнопку «Заполнить выражения», будет выполнено сопоставление по типу реквизитов и измерений (ресурсов, реквизитов).

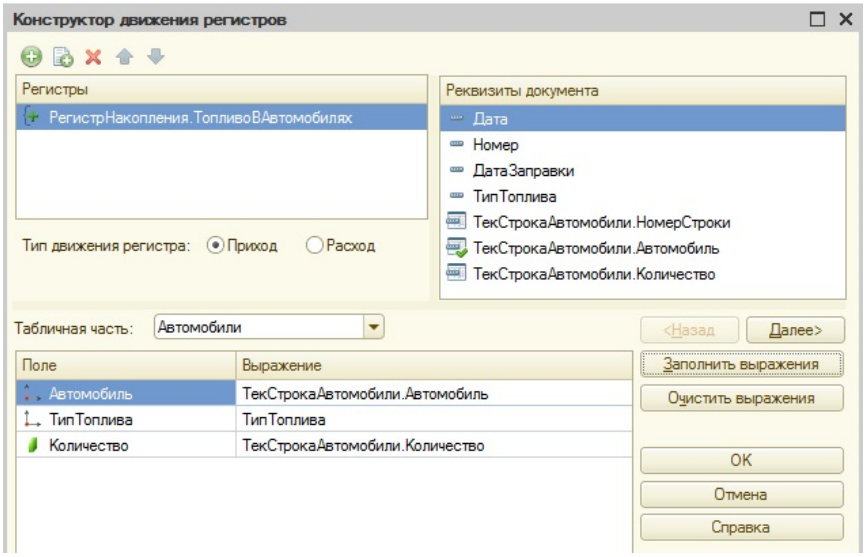


Рис. 4.1.128

Тип движения регистра оставляем *приход*, поскольку данный документ будет отображать факт прибавления топлива в автомобиле.

Если Вам не понятно, какое движение должен делать тот или иной документ, то руководствуйтесь следующим мнемо-правилом:

Если природа документа отражает увеличение какой-нибудь сущности где-либо (приход товара на склад, прием «налички» в кассу, начисление налога со сделки и т.п.), то этот документ должен делать движение прихода по регистру накопления. Если же природа документа отражает уменьшение какой-нибудь сущности где-либо (отгрузка товара, выдача «наличики» из кассы, оплата налога и т.п.), то этот документ должен делать движение расхода по регистру накопления.

Нажимаем кнопку *«ОК»*, и будет создана обработка проведения с соответствующим кодом.

То же самое проделайте с документом *«Отчет о расходе топлива»*, только ставим *Тип движения регистра - Расход*.

Рис. 4.1.129

Сохраним конфигурацию, обновим базу данных, создадим и проведем документ «Заправка топлива».

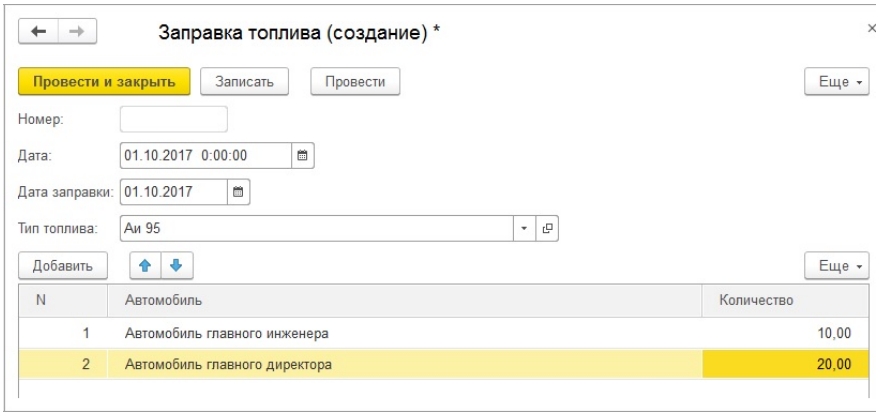


Рис. 4.1.130

А также документ *«Отчет о расходе топлива»*.

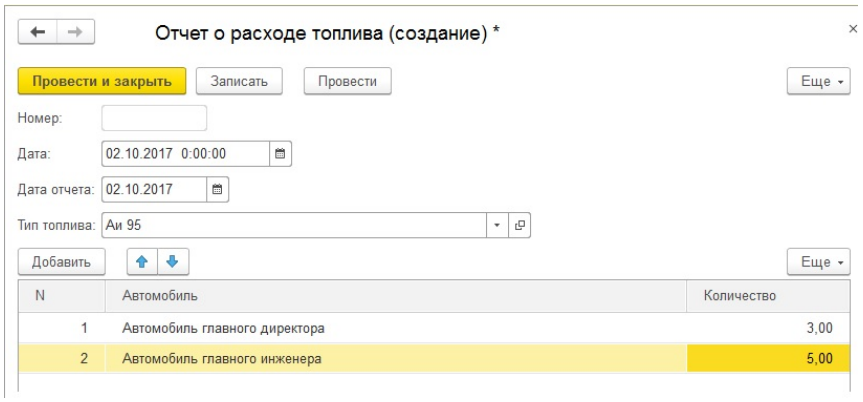


Рис. 4.1.131

И посмотрим, какие движения сделали эти документы.

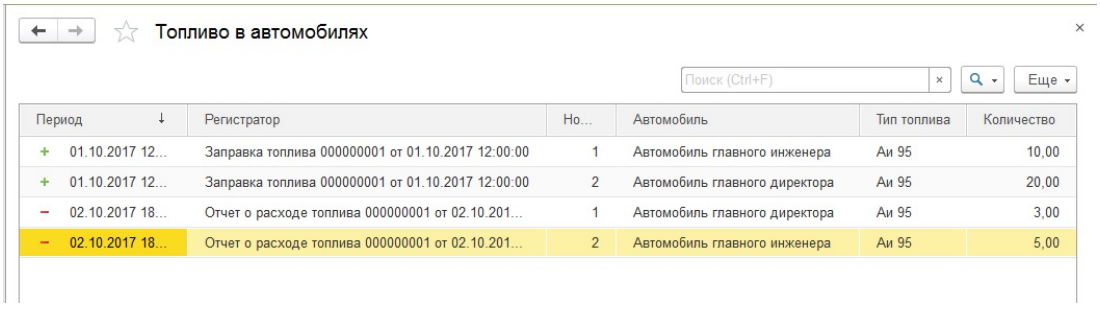


Рис. 4.1.132

План видов характеристик

Рассмотрим такой интересный объект конфигурации, как *план видов характеристик*.

Часто при разработке прикладных решений могут появляться задачи добавления дополнительных характеристик для различных объектов учета. Например, у нас есть справочник *Гаражи*, вполне возможно, что в процессе работы нам понадобится хранить такую информацию, как место гаража, длина, ширина, тип гаража и т.д.

Всю дополнительную информацию можно хранить в реквизитах этого справочника, и это решение вполне имеет право на жизнь, но только в том случае, когда состав характеристик всегда будет один и тот же, и он заранее известен. Если же состав реквизитов будет постоянно меняться, то нам каждый раз придется переписывать конфигурацию: добавлять новый реквизит со всеми вытекающими последствиями (переписывание форм и т.п.).

Может быть и иная ситуация: у разных объектов может быть разный состав характеристик. Например, для какого-то гаража важно указать длину и ширину, а для какого-то эта информация не столь критична. Тогда часть реквизитов будут лишними.

Во всех этих случаях гораздо рациональнее использовать *планы видов характеристик*, в которых хранится дополнительная информация объектов аналитического учёта. Что такое *план видов характеристик*? По сути это такой справочник, в котором элементы (характеристики) могут быть различных типов. Например, есть характеристика *Высота гаража*, тип которой *Число*. В характеристику *Высота гаража*, можно записать только число и не более. Причем у одного *плана видов характеристик* могут быть характеристики разных типов. Таким образом, можно резюмировать: если у обычного справочника все элементы одного типа (ссылка на этот справочник), то у плана видов характеристик элементы могут быть разных типов, в зависимости от того, как настроено в конфигурации.

Реализуем следующую задачу: создадим возможность хранения различных характеристик справочника *Гаражи*. Характеристики могут быть в виде примитивных типов (например, количество мест в гараже), а также и в виде каких-то значений (например, расположение гаража: «Север», «Юг» и т.д.), которые могут создавать сами пользователи.

Для этого создадим план видов характеристик – *ДополнительныеСвойстваГаражей*.

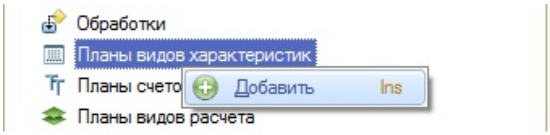


Рис. 4.1.133

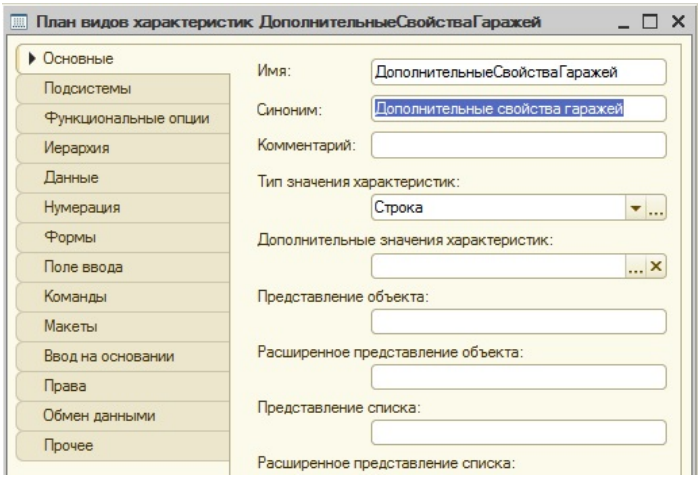


Рис. 4.1.134

В конструкторе плана видов характеристик на закладке «Основные» обратите внимание на поле «Тип значения характеристик», в этом поле необходимо задать тип (или типы). Характеристики нашего нового плана вида характеристик будут только тех типов, которые указаны в этом поле. Если нажать на кнопку «…» данного поля, то выйдет окно редактирования типа данных.

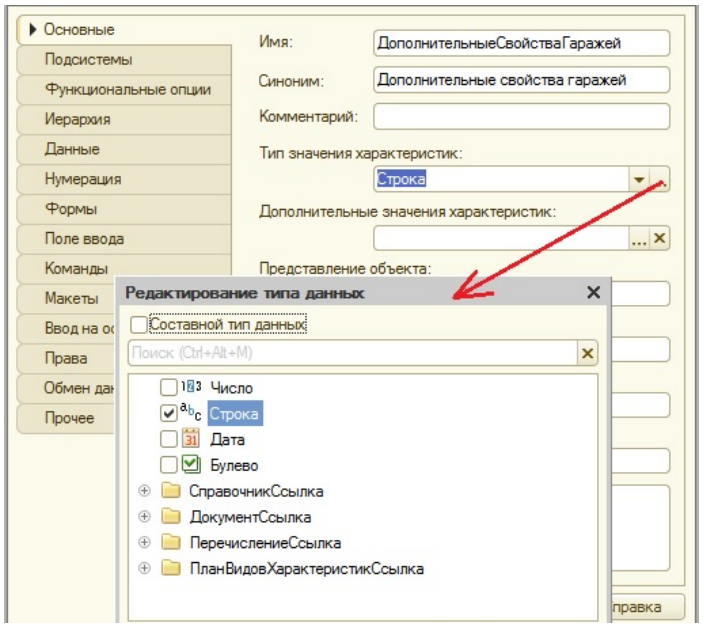


Рис. 4.1.135

Где можно задать как конкретный тип, который будет у всех характеристик объекта, так и перечень типов (установить флаг «Составной тип данных»). Мы зададим все примитивные типы.

Рис. 4.1.136

Но, в условиях задачи было сказано, что характеристики могут храниться в виде каких-то определенных значений, и у пользователя должна быть возможность выбрать одно из этих значений. Причем эти значения пользователь может задать самостоятельно. Где наиболее оптимально хранить значения той или иной характеристики? Причем так, чтобы эти значения не пересекались.

К примеру, если есть характеристика *«Расположение гаража»*, которая содержит произвольные значения расположения гаража («Север», «Юг», «Запад» и т.д.), а также есть характеристика *«Тип гаража»,* которая содержит произвольные значения типа гаража («Кирпичный», «Железный» и т.д.), значения этих характеристик не должны пересекаться. Если мы выбрали характеристику «Расположение гаража», то должны видеть только значения расположения, но никак не значения типа гаража.

Можно, конечно, создать отдельные справочники «Расположение гаража» и «Тип гаража», но мы не можем быть уверены, что потом не добавятся еще какие-либо новые характеристики, которые будут содержать произвольные значения, и нам опять не придется менять конфигурацию. Самым оптимальным решением будет создать единый справочник для хранения всех значений подобных характеристик. Этот справочник должен быть подчинен плану видов характеристик «ДополнительныеСвойстваГаражей». В этом случае у любой характеристики (элемента плана видов характеристик), тип которой «ссылка на подчиненный справочник», будет свой набор значений, ни с чем не пересекающийся.

Немного запутанно? Сейчас все станет понятно.

Создадим такой справочник и назовем его *СвойстваГаражей* (создайте самостоятельно, длина кода – 3, длина наименования -50). На закладке «Владельцы» установим единственного владельца, план видов характеристик «Дополнительные свойства гаражей» (см. рис. 4.1.137).

После создания справочника перейдем в план видов характеристик «Дополнительные свойства гаражей» и в поле «Тип значения характеристик» закладки «Основные» добавим новый тип – ссылку на справочник «Свойства гаражей» (см. рис. 4.1.138). Теперь мы сможем создать элементы (характеристики) нашего плана видов характеристик с этим типом, а также создать элементы справочника «Свойства Гаражей», подчиненные нужной характеристике.

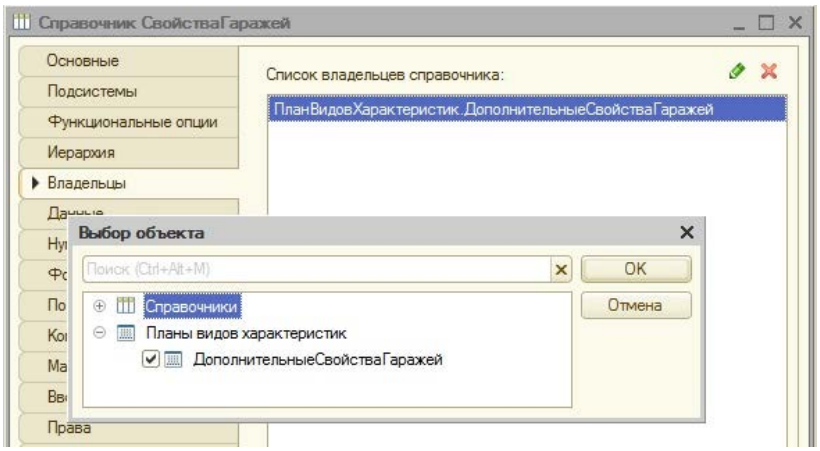


Рис. 4.1. 137

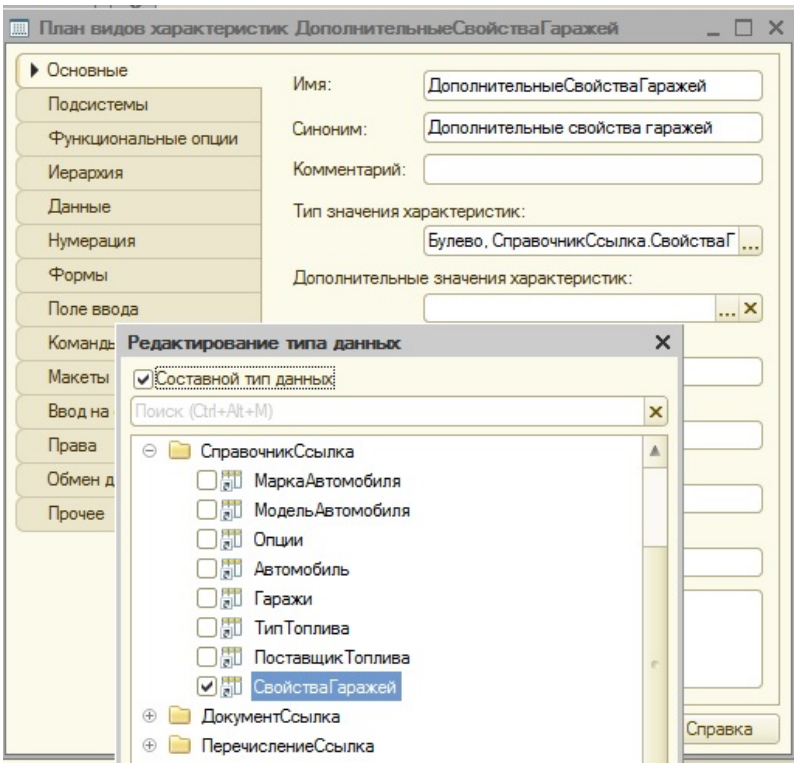


Рис. 4.1. 138

Но это еще не всё, в плане видов характеристик нужно указать, что значения характеристик хранятся именно в справочнике «Свойства гаражей». Указывается это в свойстве «Дополнительные значения характеристик» закладки «Основные».

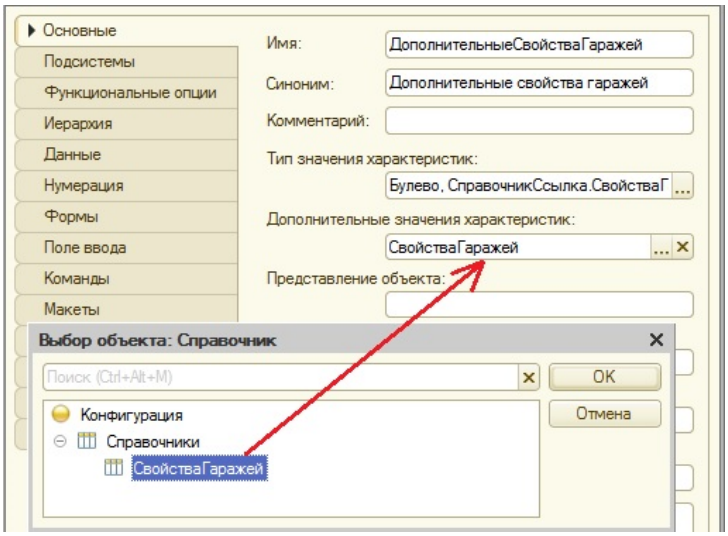


Рис. 4.1.139

В это свойство нужно выбрать название справочника, который подчинен нашему плану видов характеристик и в котором будут храниться значения характеристик.

И остался последний шаг: необходимо связать воедино гараж, дополнительные свойства гаража (характеристики) и значение этих свойств (характеристик). Поскольку когда мы создадим какие-то свойства гаражей, их значения, то эти данные и конкретные гаражи будут отдельно.

Мы сделаем это, используя непериодический и независимый регистр сведений, который назовем «ЗначениеСвойствГаражей».

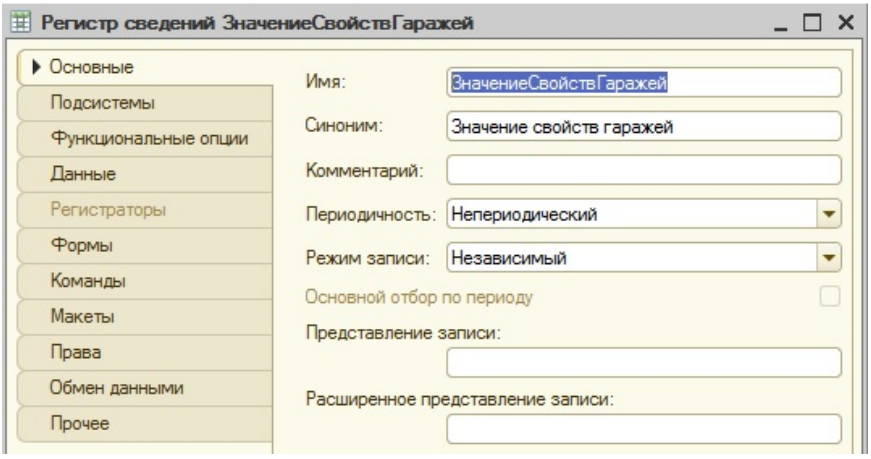


Рис. 4.1.140

Сейчас нам нужно определить состав измерений и ресурсов. Самое очевидное создать три измерения: «Гараж» (тип ссылка на справочник *Гаражи*), «Свойство» (тип ссылка на план видов характеристик *ДополнительныеСвойстваГаражей*) и «Значение этого свойства». Но будет ли данный способ правильным? Очевидно, нет, т.к. возможна будет запись следующего вида:

Гараж №1	Тип гаража	Каменный
Гараж №1	Тип гаража	Железный

Т.е. у одного гаража будет одновременно два разных свойства, что, согласитесь, неправильно. Поэтому логично создать два измерения, в которых будут указаны гаражи и свойства, и один ресурс, в котором будет указано значение свойства гаража.

Создадим измерения.

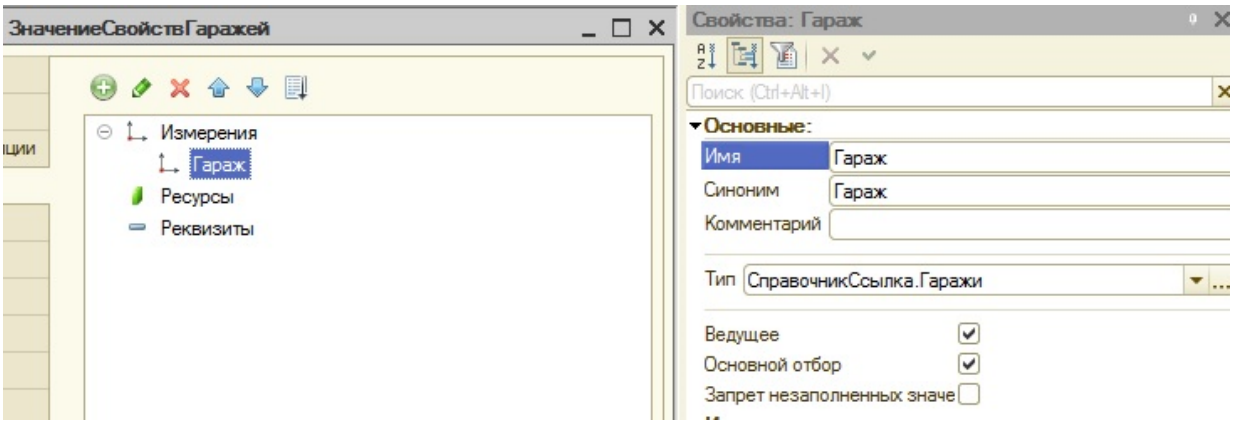


Рис. 4.1.141

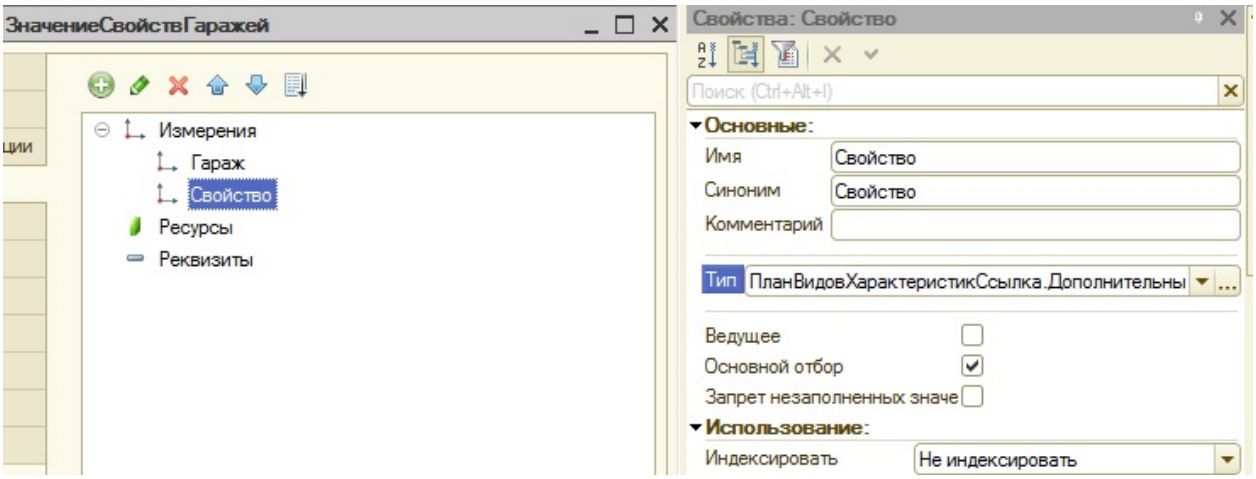


Рис. 4.1.142

У измерения «Гараж» установите свойство *Ведущее*.

И создадим ресурс «Значение», тип которого будет характеристика плана видов характеристик «Дополнительные свойства гаражей».

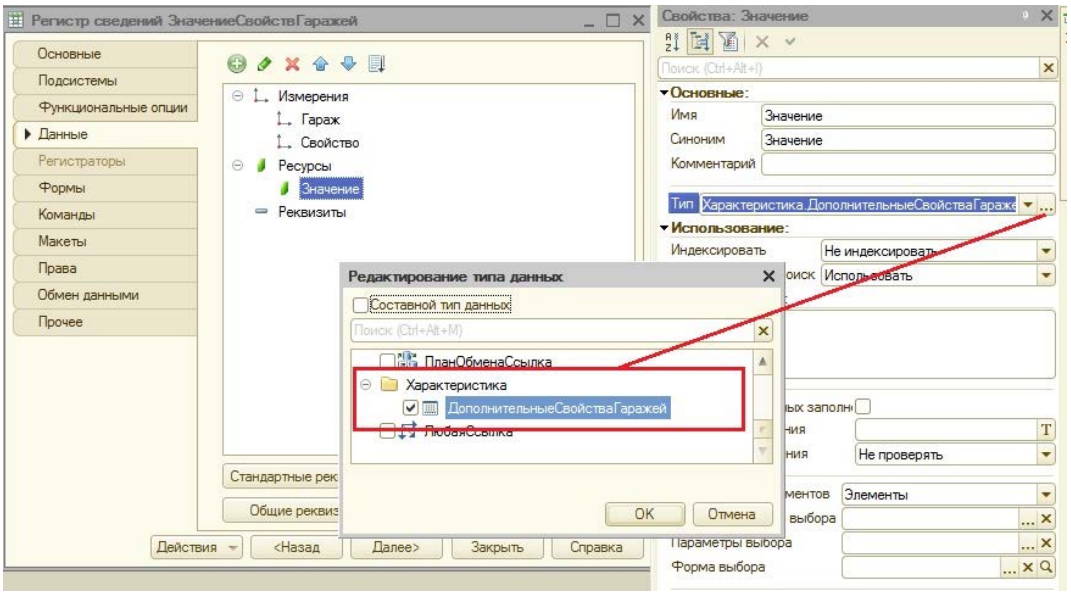


Рис. 4.1.143

В результате в ресурсе *Значение* будут те значения характеристик, которые указаны в плане видов характеристик «Дополнительны свойства гаражей».

Пока остановимся на этом. Сохраним конфигурацию, обновим базу данных и зайдем в план видов характеристик «Дополнительные свойства гаражей», используя меню «Все функции».

Создадим нашу первую характеристику *Длина*, тип которой будет *Число*.

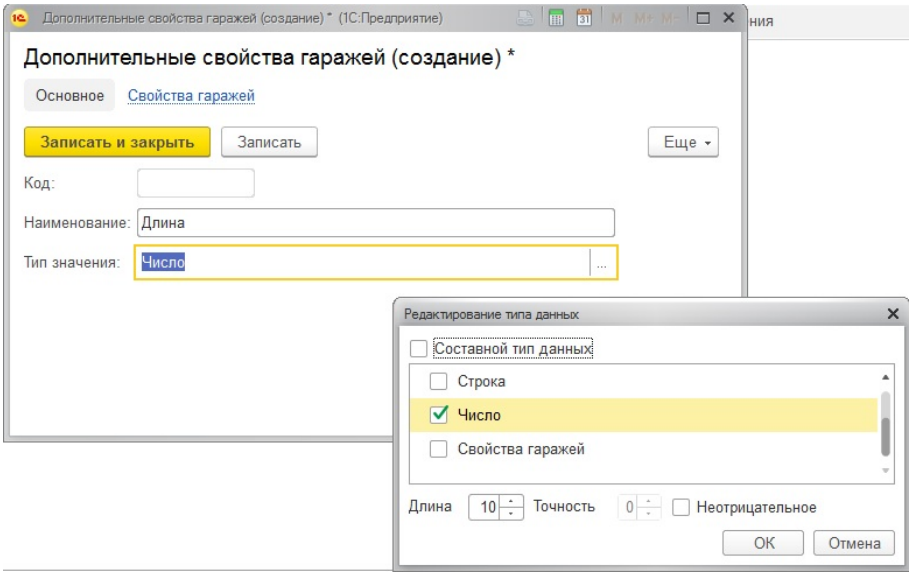


Рис. 4.1.144

Самостоятельно точно также создайте характеристику *Ширина*, тип которой также будет *Число*.

Создадим свойство *Расположение*, тип которого будет ссылка на справочник «Свойства гаражей».

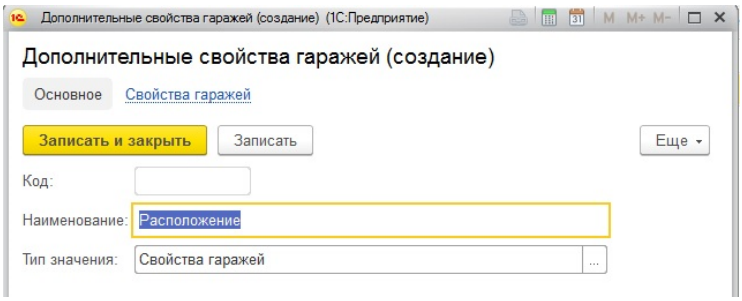


Рис. 4.1.145

И зададим значения этого свойства, для этого перейдем в подчиненный справочник «Свойства гаражей», нажав на команду «Свойства гаражей» формы элемента плана видов характеристик, где зададим несколько значений этого свойства.

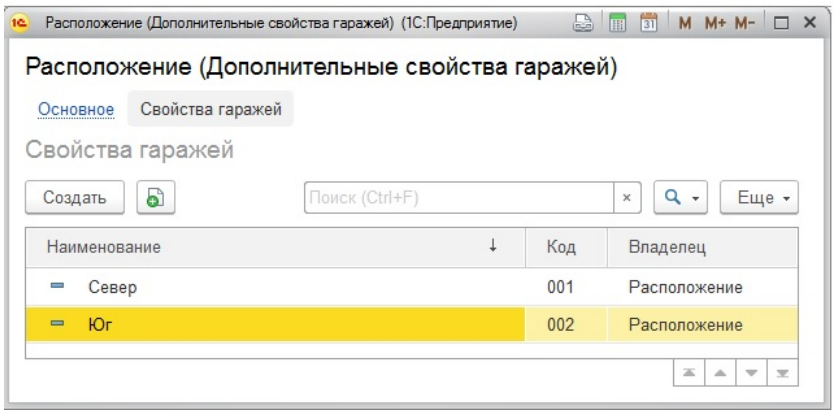


Рис. 4.1.146

Аналогично создадим характеристику «Вид гаража» с типом ссылка на справочник «Свойства гаражей», и с некоторыми значениями.

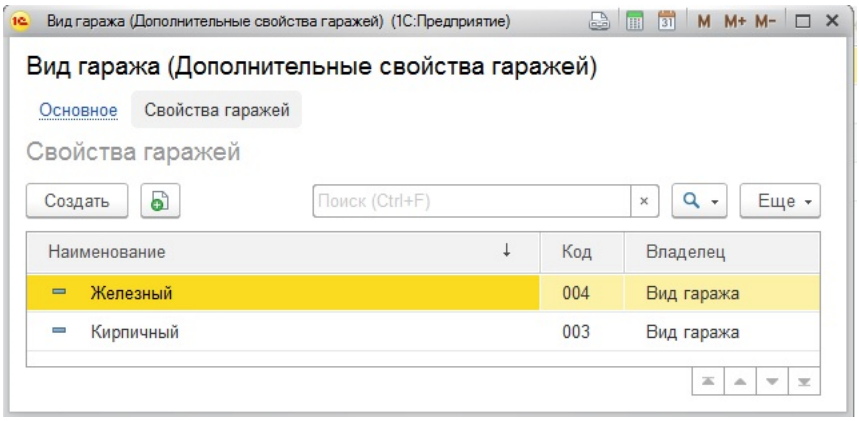


Рис. 4.1.147

Все нужные на данный момент свойства гаражей созданы, должен получиться следующий список плана видов характеристик:

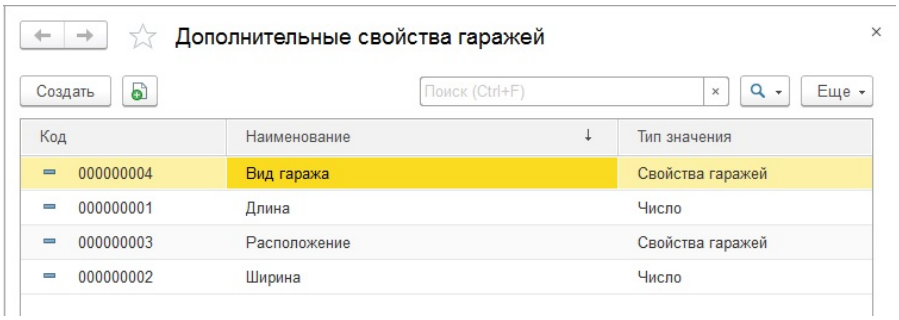


Рис. 4.1.148

Попробуем при помощи регистра «Значение свойств гаражей» привязать какую-нибудь характеристику и её значение к определенному гаражу. Заходить в сам регистр не нужно, достаточно зайти в элемент справочника *Гаражи* и выполнить команду «Значение свойств гаражей», которая расположена вверху формы элемента.

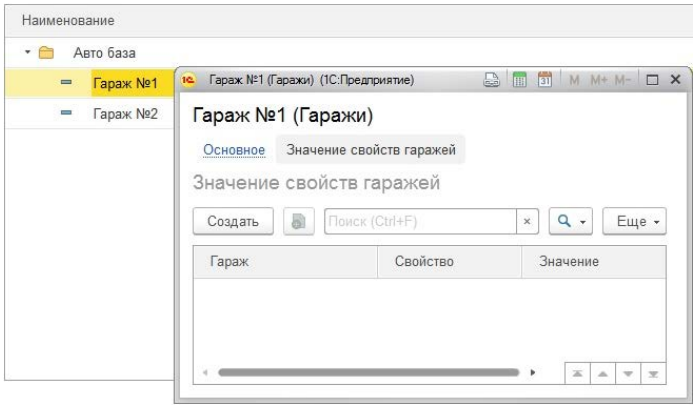


Рис. 4.1.149

Создадим новую запись регистра, в которой выберем свойство «Вид гаража».

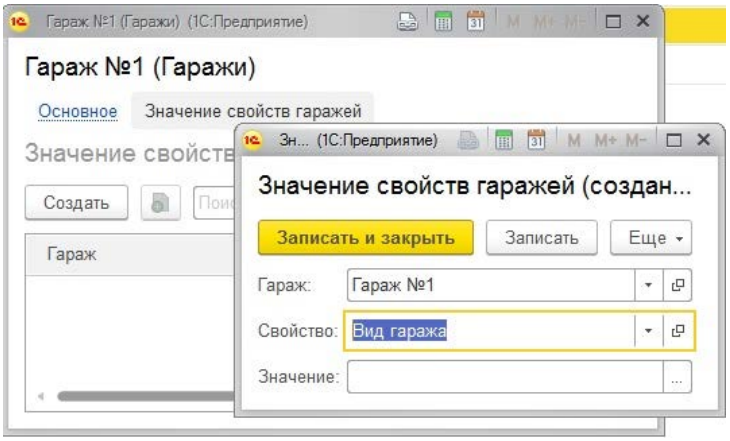


Рис. 4.1.150

Попробуем выбрать значение этого свойства. Нам сразу будет предложено выбрать тип данных. Что несколько неудобно, согласитесь.

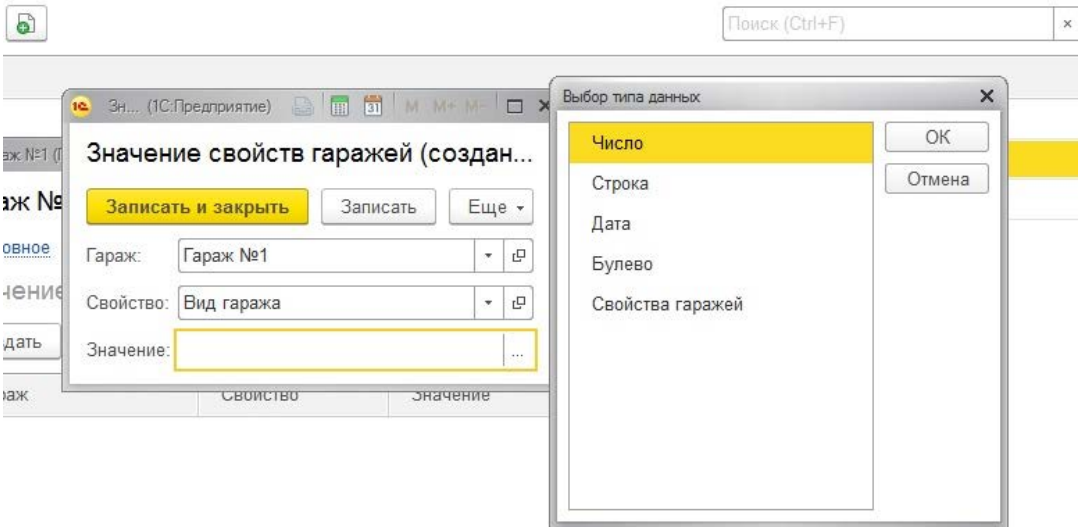


Рис. 4.1.151

Когда мы выберем тип «Свойства гаражей», то будут предложены значения всех свойств, что тоже очень неудобно.

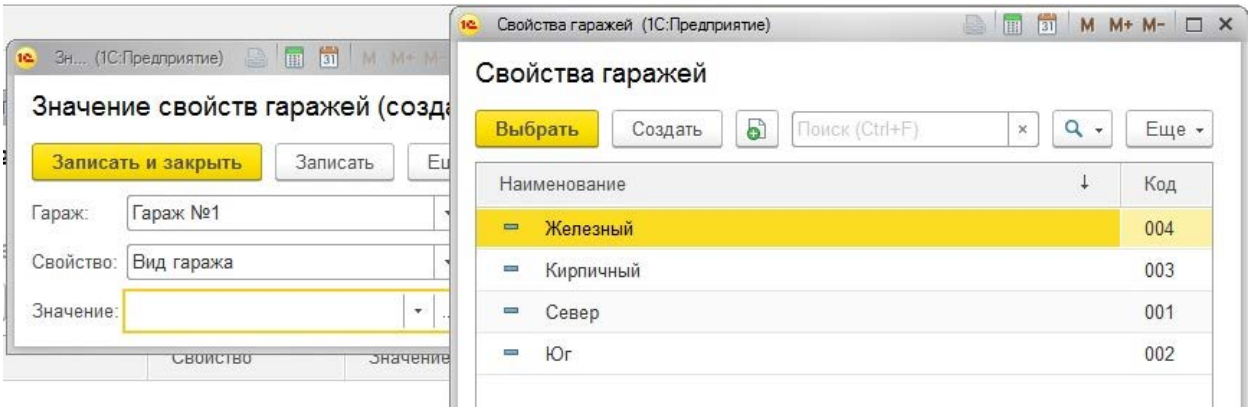


Рис. 4.1.152

Приведем выбор значения в нормальный вид. Для этого вернемся в конфигуратор, и откроем палитру свойств ресурса *Значение* регистра сведений «Значение свойств гаражей».

В этой палитре нас интересуют два поля – *Связь по типу* и *Связи параметров выбора* (рис. 4.1.153). В свойстве *Связь по типу* необходимо указать измерение, которое будет определять тип значения нашего ресурса. В нашем случае это будет измерение *Свойство* (см. рис. 154).

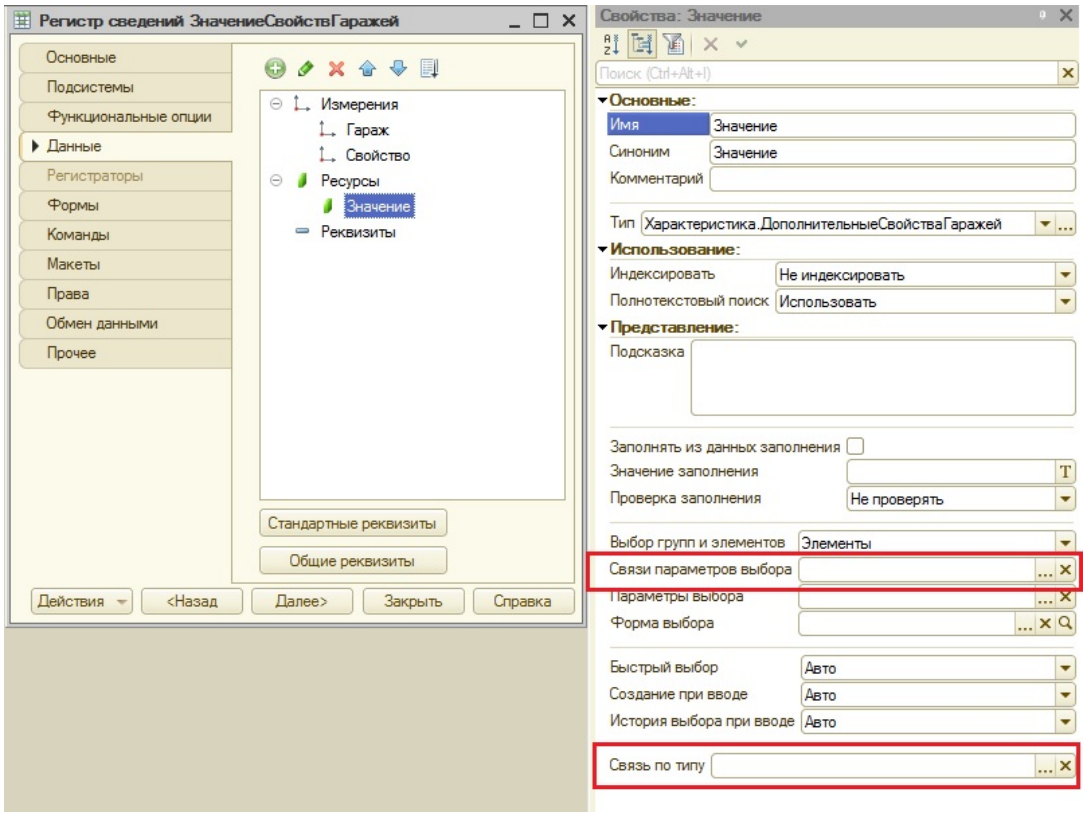


Рис. 4.1.153

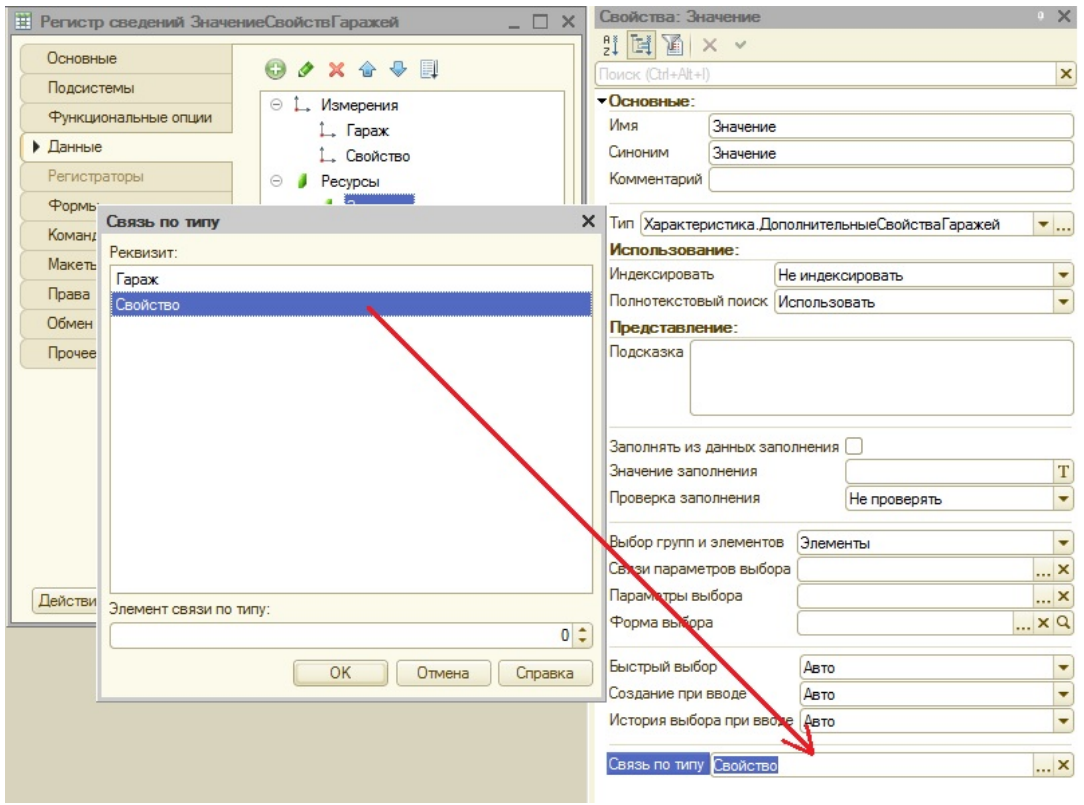


Рис. 4.1.154

При помощи свойства *Связи параметров выбора* настраивается зависимость одних реквизитов от других (в нашем случае зависимость ресурса от измерения). В открывшейся форме

связи нужно выбрать реквизит *Свойство*. И если Вы все правильно сделали, то в форме связи должна появиться строка, как на рис. 4.1.155.

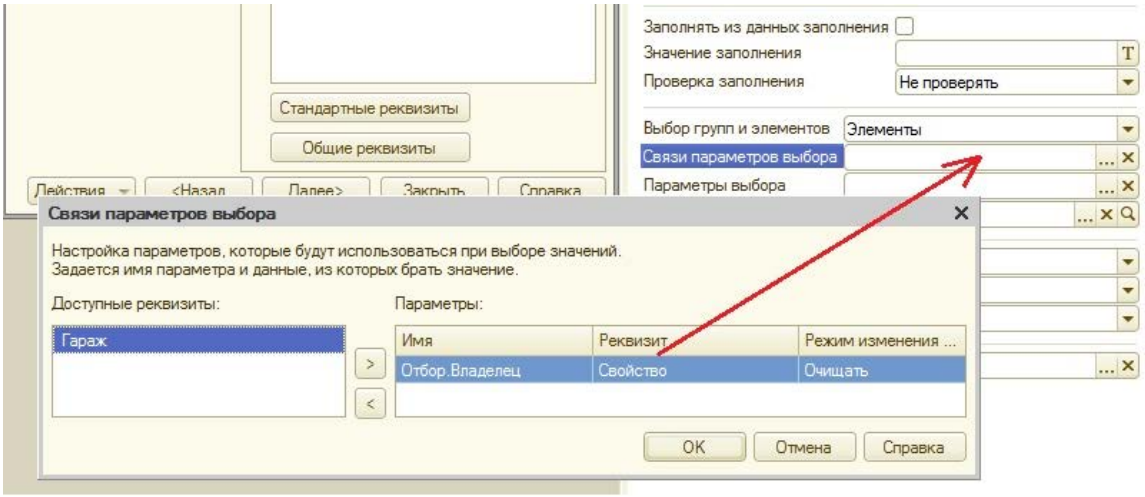


Рис. 4.1.155

Тем самым мы настроим отбор по владельцу. И при выборе элементов справочника «Свойства гаражей» будут выходить только те элементы, владелец у которых указан в измерении *Свойство* текущей строки регистра. Сохраним конфигурацию, обновим базу данных и попробуем задать пару характеристик для справочника «Гаражи».

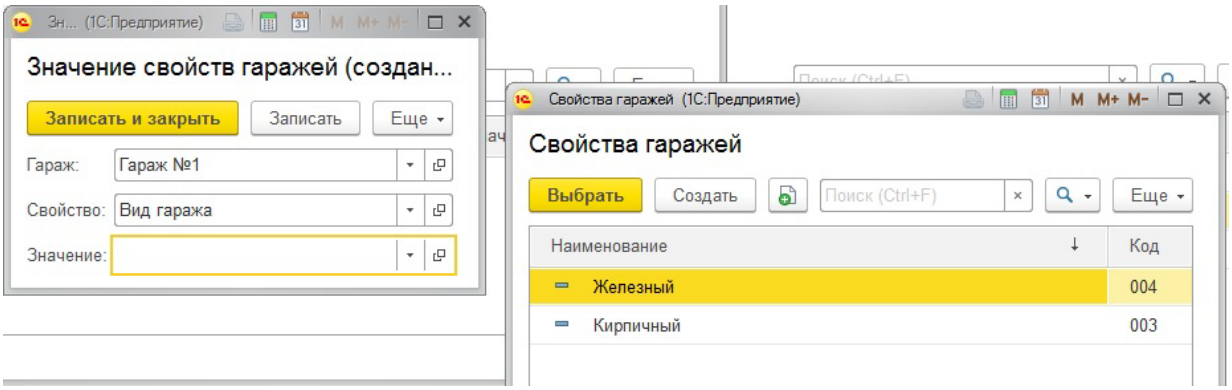


Рис. 4.1.156

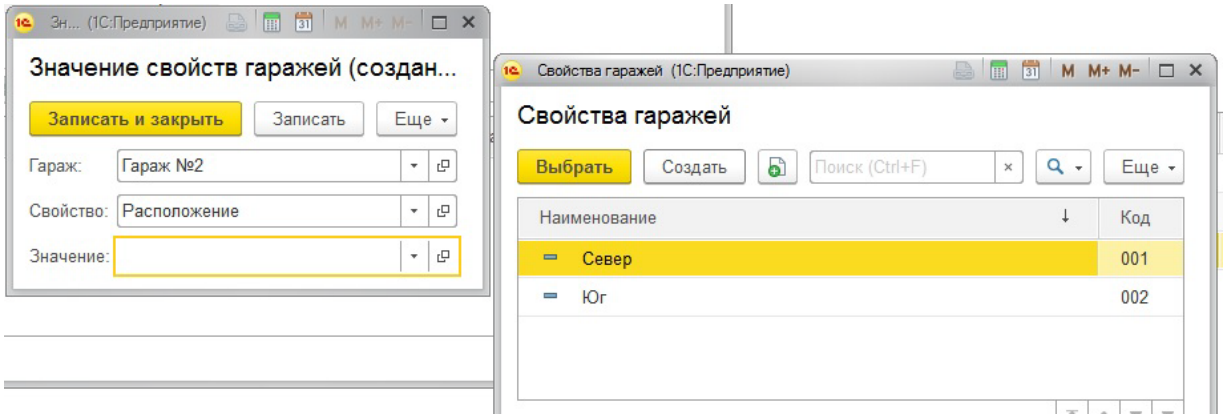


Рис. 4.1.157

Обработки

С *Обработками* Вы уже знакомы. Мы с вами работаем с внешними обработками, но могут быть и обработки в составе конфигурации.

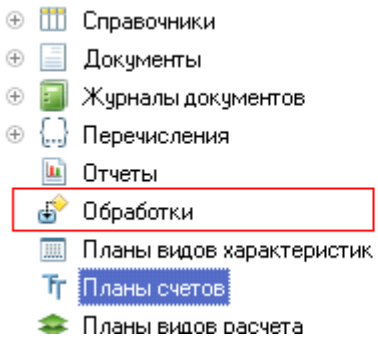


Рис. 4.1.158

Можно создать как внешнюю обработку, так и обработку, встроенную в конфигурацию. У обработок также могут быть реквизиты и табличные части. Создавайте любую обработку в конфигурации просто для примера и назовите ее *Обработка.*

Для этого кликаете правой клавишей мышки по пункту конфигурации *«Обработки»,* в появившемся меню нажимаем *«Добавить»*. Добавьте реквизит *Автомобиль* и табличную часть *Гаражи*.

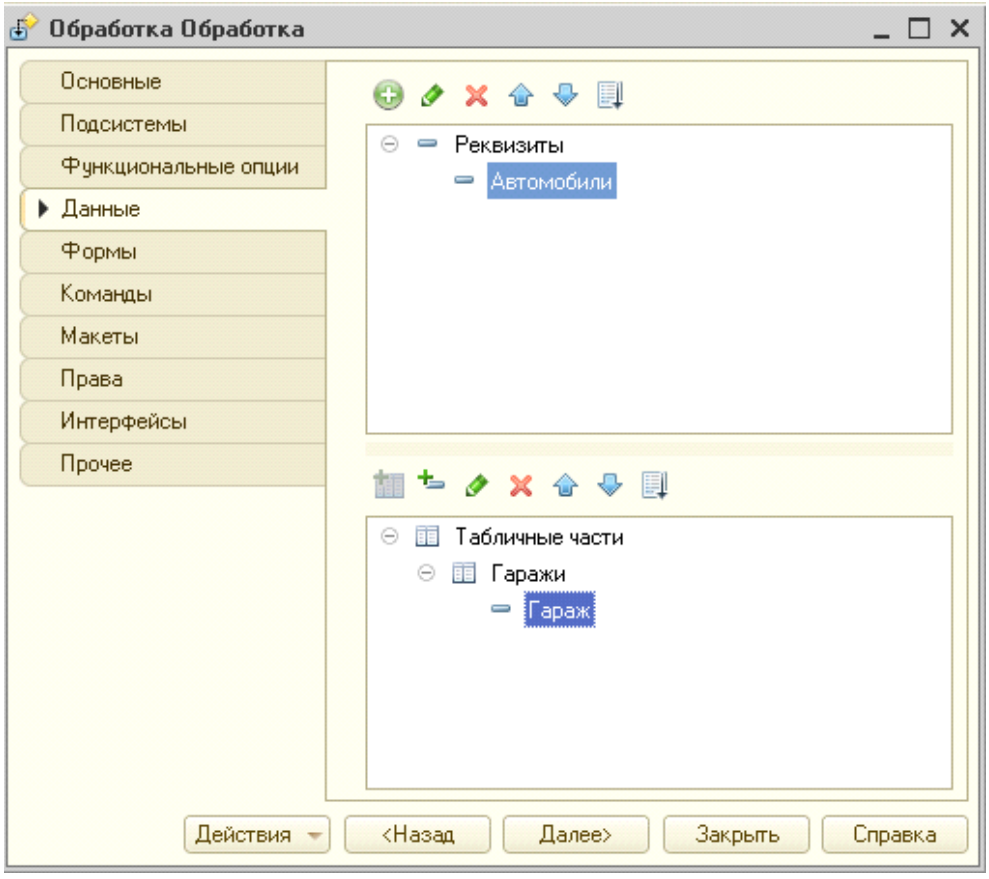


Рис. 4.1.159

Также можете создать на ней форму с данными реквизитами.

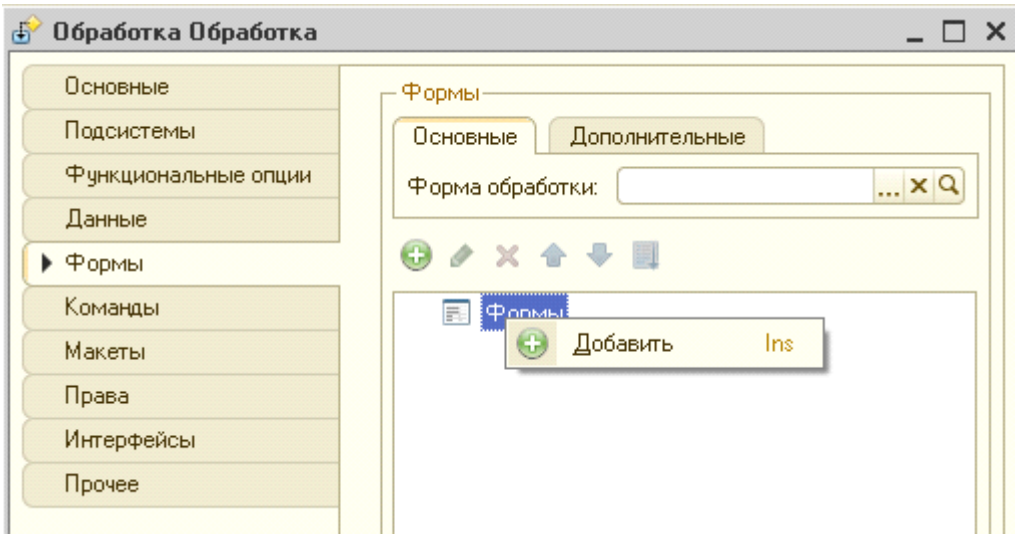


Рис. 4.1.160

Еще Вы можете вставить уже имеющуюся обработку в конфигурацию. Для этого кликнем правой кнопкой мышки по пункту конфигурации *«Обработки»* и выберем пункт *«Вставить внешнюю обработку, отчет…»*.

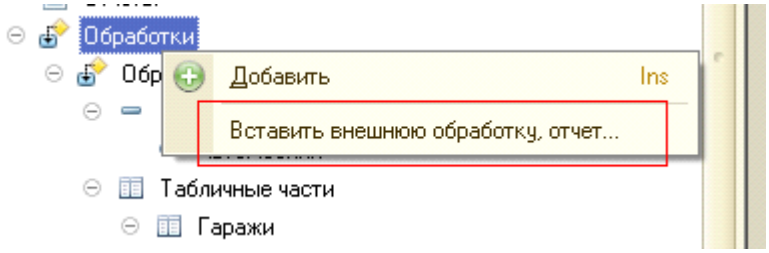


Рис. 4.1.161

И найдите любую недавно созданную Вами обработку (из предыдущих глав) и выберете ее. Она станет в составе конфигурации. Сохраните конфигурацию и откройте обработку из меню операции.

Отчеты

Отчеты необходимы для анализа информации о деятельности предприятия. Более подробно отчеты Вы будете изучать в 11-й главе. Пока замечу, что отчеты могут быть сами по себе (внешние отчеты), а могут и входить в состав конфигурации.

Основные метаданные ветки «Общие»

Разберем основные метаданные ветки «Общие» дерева конфигурации, которые пригодятся в первоначальной работе.

Параметры сеанса

Параметры сеанса - это переменные, которые доступны в течение всего сеанса пользователя в любом месте конфигурации. Например, в параметрах сеанса можно хранить текущего пользователя, или имя компьютера.

Создадим параметр сеанса, который назовем *ИмяТекущегоКомпьютера*. Для этого выделим ветку «Параметры сеанса» и в контекстном меню нажмем на пункт «Добавить».

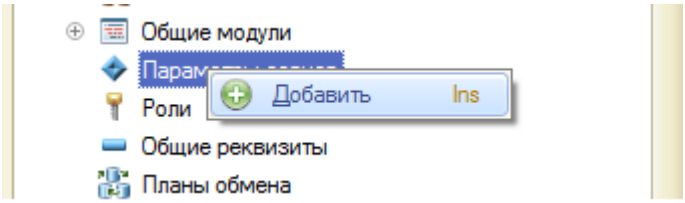


Рис. 4.1.162

В правой части экрана откроется палитра свойств параметра сеанса, где укажем название, синоним и тип (Строка (50)).

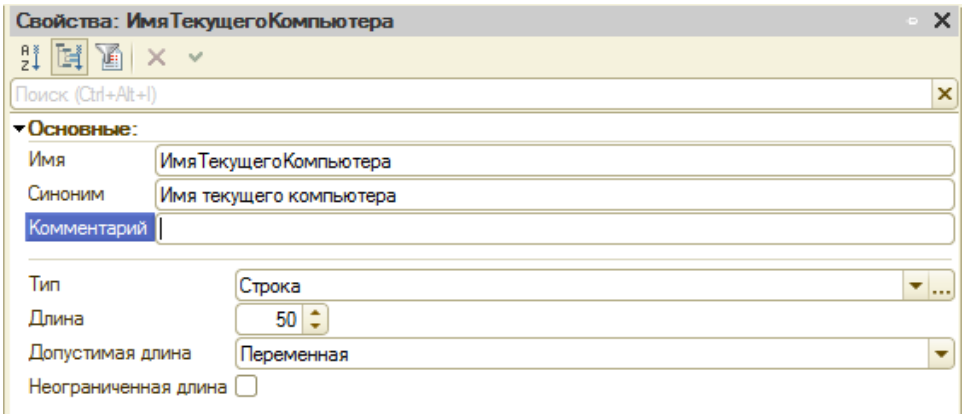


Рис. 4.1.163

Определяются параметры сеанса в модуле сеанса, который мы будем проходить ниже (часть вторая текущей главы, стр. 247), где и научимся определять имя текущего компьютера.

Определяемые типы

Кроме примитивных и ссылочных типов (ссылочные типы будем проходить в шестой главе), в конфигураторе 1С можно задать предопределяемый тип. Например, у нас много где встречается такой реквизит как цена, которые имеет всегда тип «Число» (длина 10, точность 2). Чтобы никогда не задумываться о параметрах числа для реквизитов, где будут храниться цены, можно создать предопределенный тип, который назовем, как вариант, *ТипДляЦен*. И этот тип можно использовать во всех подобных реквизитах.

Создадим такой тип: выделим ветку «Определяемые типы» и нажмем кнопку «Добавить».

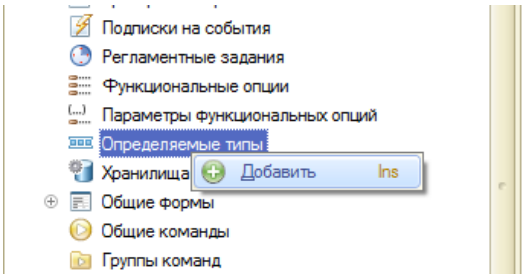


Рис. 4.1.164

В правой части откроется палитра свойств определяемого типа, где мы зададим такие параметры как «Имя», «Синоним» и «Тип определяемого типа».

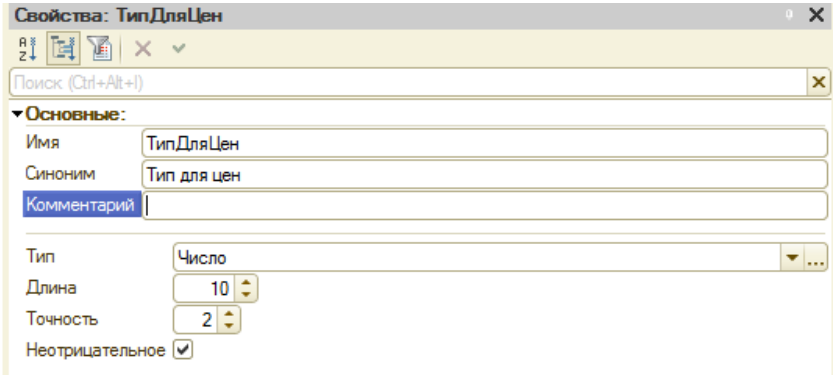


Рис. 4.1.165

Сейчас мы сможем у всех реквизитов конфигурации, в которых хранятся какие-либо цены, поменять тип. Для этого нужно зайти в палитру свойств нужного реквизита, нажать на кнопку «…» у свойства *Тип*, и выбрать нужный определяемый тип из папки «Определяемый Тип» окна редактирования типа данных.

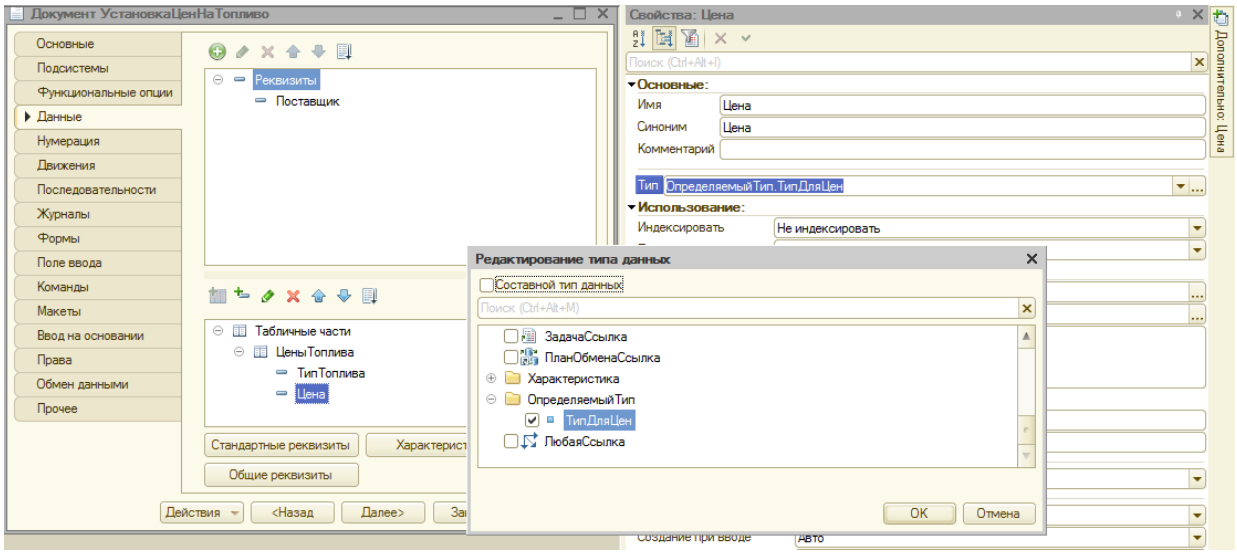


Рис. 4.1.166

Поменяйте тип у всех оставшихся реквизитов «Цена» самостоятельно.

Общие реквизиты

В процессе разработки конфигурации могут возникать ситуации, когда у ряда объектов может быть одинаковый реквизит. Например, реквизит «Комментарий» или «Ответственный» у документов. В платформе 8.3 есть возможность создать несколько общих реквизитов, которые разработчик может использовать в том или ином объекте на свое усмотрение.

Создадим общий реквизит *Комментарий*, который будем использовать во всех документах (только документах). Для этого выделим ветку «Общие реквизиты» и нажмем кнопку «Добавить».

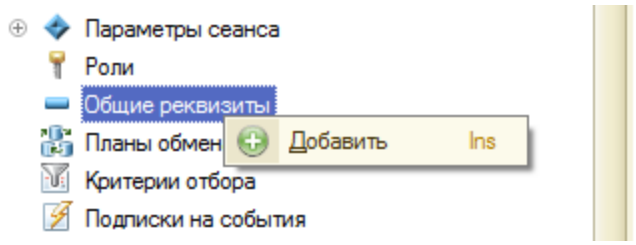


Рис. 4.1.167

После добавления общего реквизита справа откроется палитра свойств этого реквизита, где зададим имя, синоним и тип (неограниченная строка).

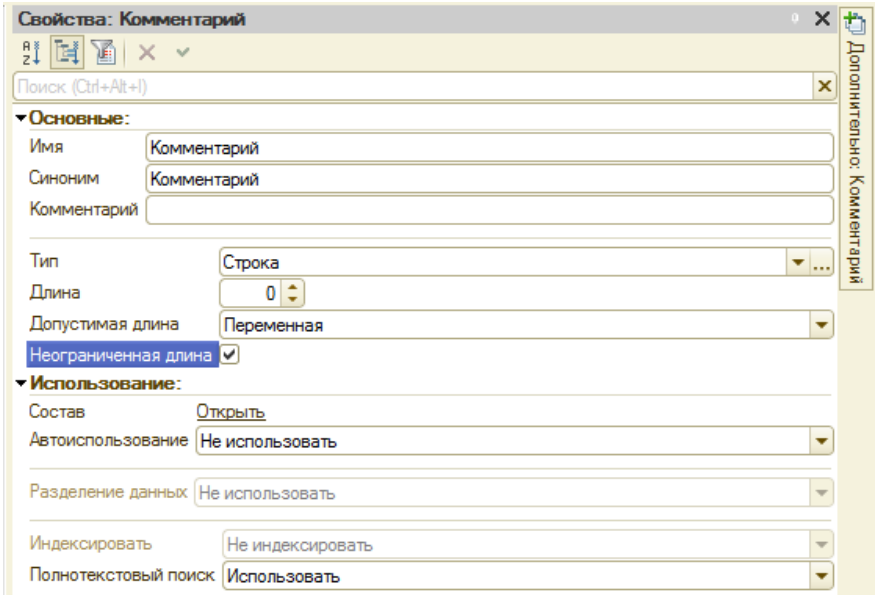


Рис. 4.1.168

Применение реквизита в объектах данных задается в категории *Использование* палитры свойств реквизита. Где нас интересуют два свойства – *Состав* и *Автоиспользование.* Разберемся со свойством *Автоиспользование*. Именно при помощи этого свойства мы определяем, как будет использоваться общий реквизит по умолчанию. Если установлено значение *Использовать*, то значит, что реквизит используется везде, и нужно задать объекты, в которых не нужно применять этот реквизит (при помощи свойства *Состав*). Если же установлено значение *Не использовать*, то

наоборот – реквизит по умолчанию нигде не используется, и нужно задать объекты, где мы будем работать с этим реквизитом (при помощи свойства *Состав*).

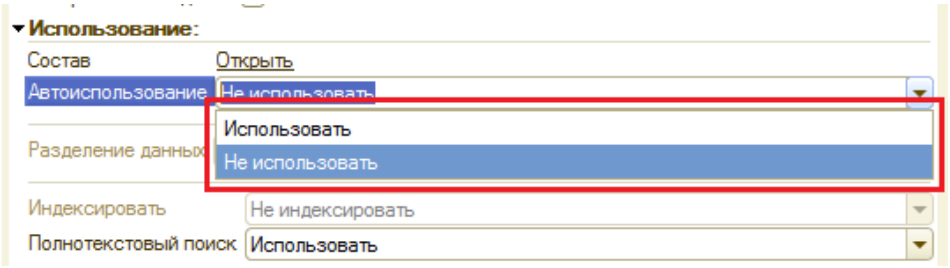


Рис. 4.1.169

По умолчанию устанавливается значение *Не использовать*. Мы оставим значение этого реквизита по умолчанию и зададим состав объектов, где будет применять общий реквизит. Для этого нужно нажать на гиперссылку «Открыть» свойства «Состав», и откроется окно состава общих реквизитов, где в верхней части расположено дерево объектов метаданных, к которым можно применить общий реквизит, а в нижней – метаданные, к которым общий реквизит применен.

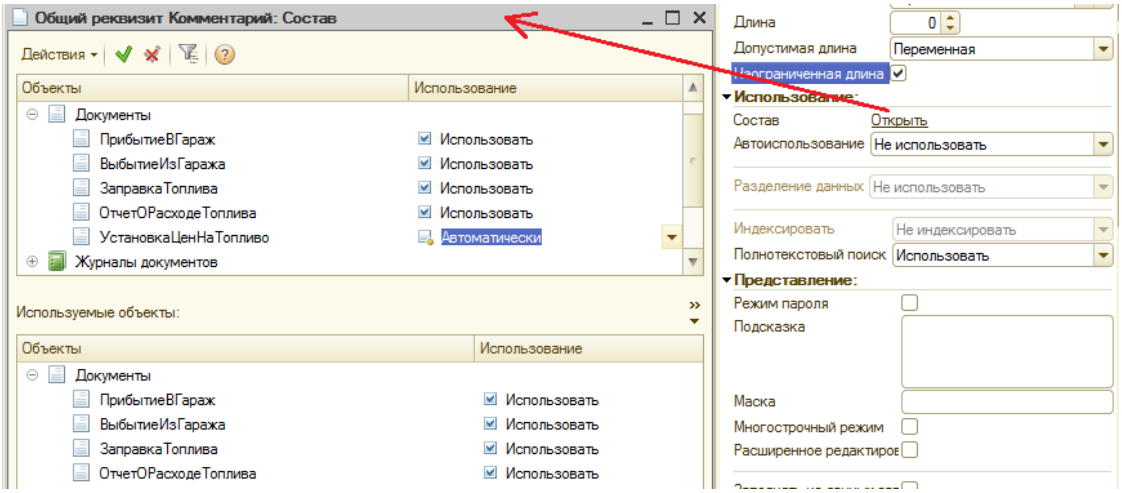


Рис. 4.1.170

Чтобы общий реквизит использовался в нужном объекте, необходимо в верхней части окна в колонке «Использование», установить *Использовать*. После этого объект отобразится в нижней части окна.

Сейчас, если мы откроем любой документ, к которому применили общий реквизит, то увидим этот реквизит на форме документа.

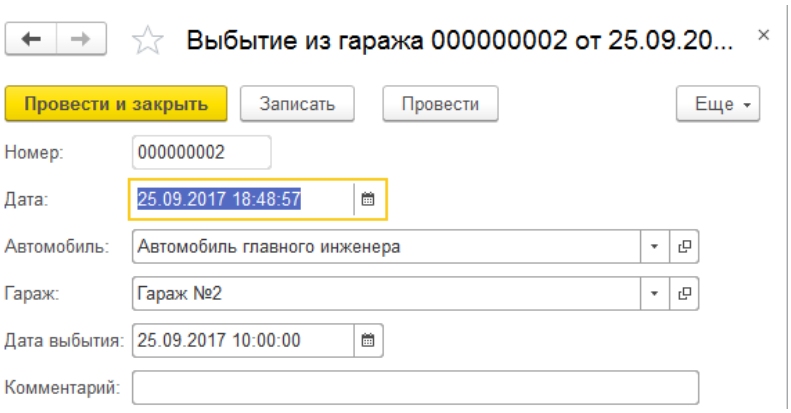


Рис. 4.1.171

Роли

При конфигурировании какого-либо программного решения могут возникнуть задачи разграничения прав ряда пользователей. Например, у каких-то пользователей может быть право на создание, проведение и удаление документов «Прибытие в гараж» и «Выбытие из гаража», а у каких-то нет. Данное разграничение прав в конфигурации 1С выполняется при помощи ролей.

В нашей конфигурации создадим три роли: *ПолныеПрава*, *Диспетчер*, *УчетчикТоплива*.

У роли *ПолныеПрава*, как ясно из названия, будет полный набор прав на все метаданные. Запомните! Роль с полными правами всегда обязательно в конфигурации. У ролей «Диспетчер» и «Учетчик топлива» будет следующий набор прав:

Роль	Объект метаданных	Права
Диспетчер	**Справочники:**	
	Марки автомобилей Модели автомобилей Опции Автомобили Гаражи Свойства гаражей	Полные
	Тип топлива Поставщик топлива	Чтение, просмотр
	Документы	
	Прибытие в гараж Выбытие из гаража	Полные
	Заправка топлива Отчет о расходе топлива Установка цен на топливо	Чтение, просмотр
	Журналы документов	
	Прибытие-выбытие автомобилей	Чтение, просмотр
	ПВХ	
	Дополнительные свойства гаражей	Полные
	Регистры сведений	
	Основной гараж автомобиля	Полные

	Значение свойств гаражей	
	Цены на топливо	Чтение, просмотр
	Регистры накопления	
	Пробег автомобиля	Полные
	Топливо в автомобилях	Чтение, просмотр
	Константы	
	Все константы	Чтение, просмотр
	Ветка «Общие»	
	Параметры сеанса	Получение, установка
	Общие реквизиты	Просмотр, редактирование
	Форма констант	Просмотр
Учетчик топлива	**Справочники:**	
	Марки автомобилей Модели автомобилей Опции Автомобили Гаражи Свойства гаражей	Чтение, просмотр
	Тип топлива Поставщик топлива	Полные
	Документы	
	Прибытие в гараж Выбытие из гаража	Чтение, просмотр
	Заправка топлива Отчет о расходе топлива Установка цен на топливо	Полные
	Журналы документов	
	Прибытие-выбытие автомобилей	Чтение, просмотр
	ПВХ	
	Дополнительные свойства гаражей	Чтение, просмотр
	Регистры сведений	
	Основной гараж автомобиля Значение свойств гаражей	Чтение, просмотр
	Цены на топлива	Полные
	Регистры накопления	
	Пробег автомобиля	Чтение, просмотр
	Топливо в автомобилях	Полные
	Константы	
	Все константы	Чтение, просмотр
	Ветка «Общие»	
	Параметры сеанса	Получение, установка
	Общие реквизиты	Просмотр, редактирование
	Форма констант	Просмотр

Табл. 4.1.2

Первым делом создадим роль с полными правами. Для этого выделим ветку «Роли» и в контекстном меню нажмем пункт «Добавить».

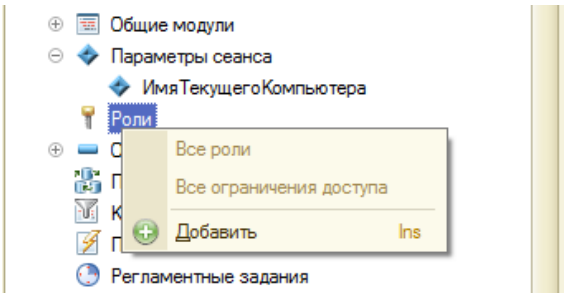


Рис. 4.1.172

У нас сразу же откроется палитра свойств роли и конструктор прав.

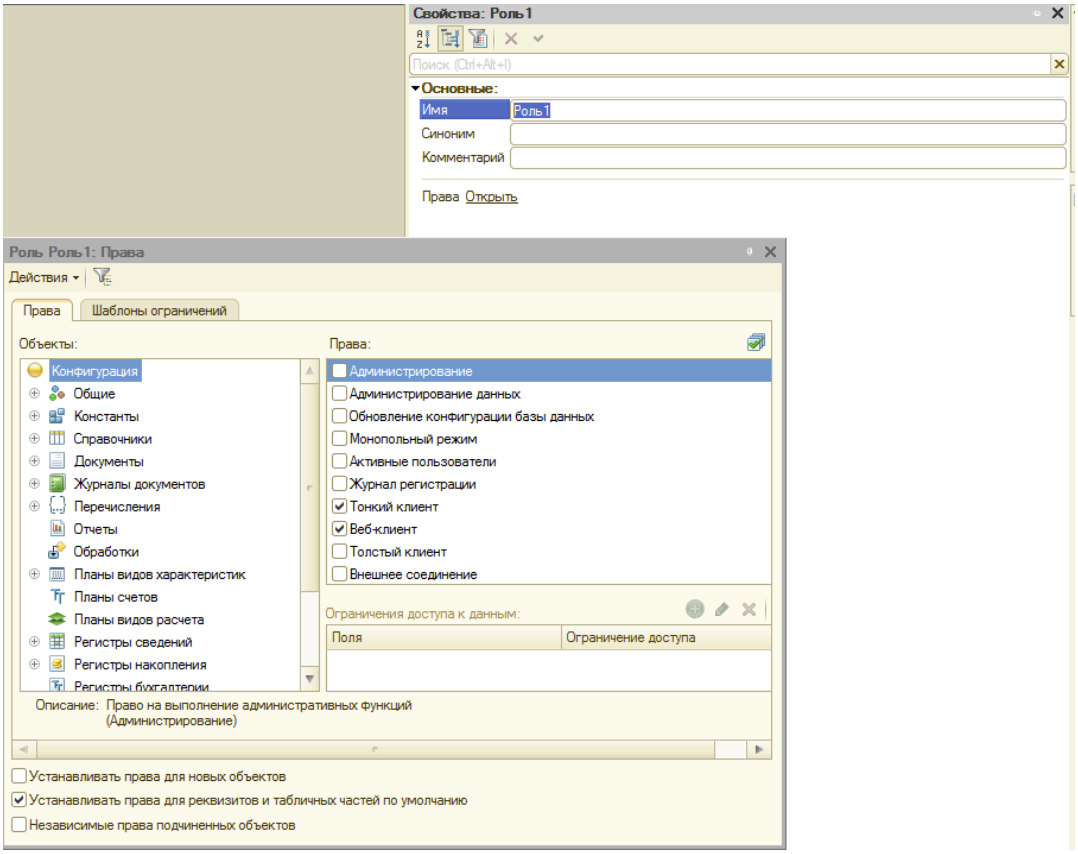


Рис. 4.1.173

Новую роль назовем *ПолныеПрава*. Для того, чтобы установить у этой роли права на все объекты, воспользуемся меню «Действия» конструктора прав, где нажмем на пункт «Установить все права».

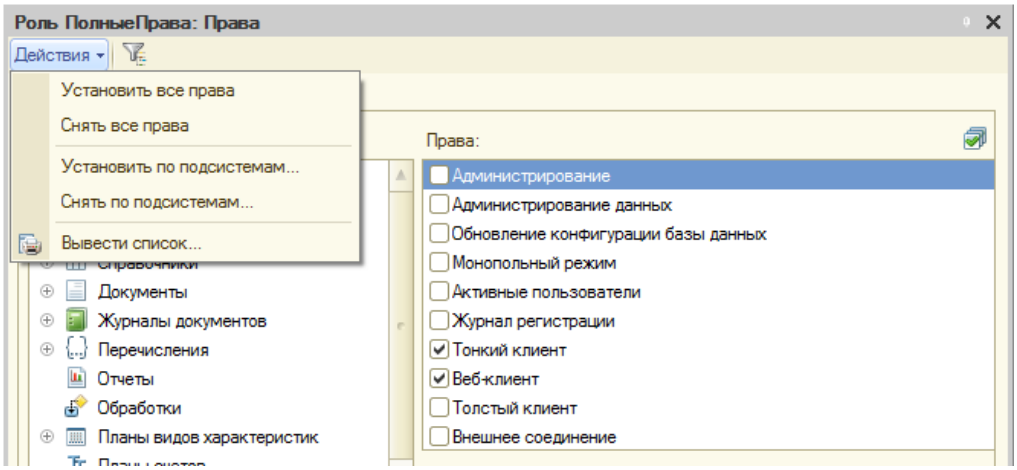


Рис. 4.1.174

Также рекомендую установить флаг «Устанавливать права для новых объектов», чтобы при создании нового объекта метаданных, роли «Полные права» добавлялись все права на этот объект.

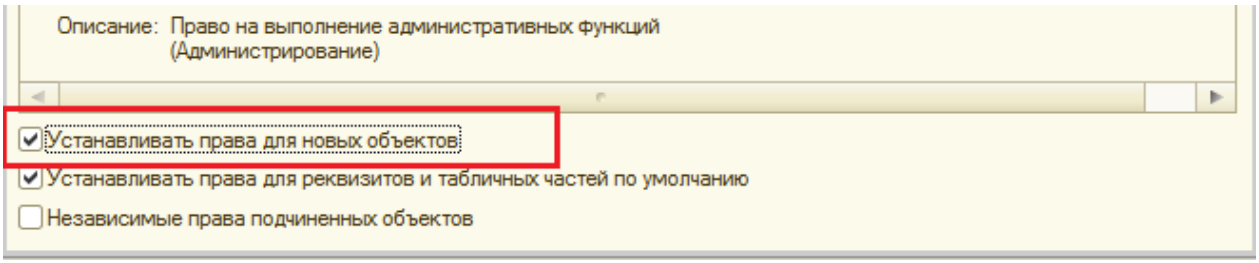


Рис. 4.1.175

В списке прав конфигурации установим все права (флаг «Отметить все элементы»).

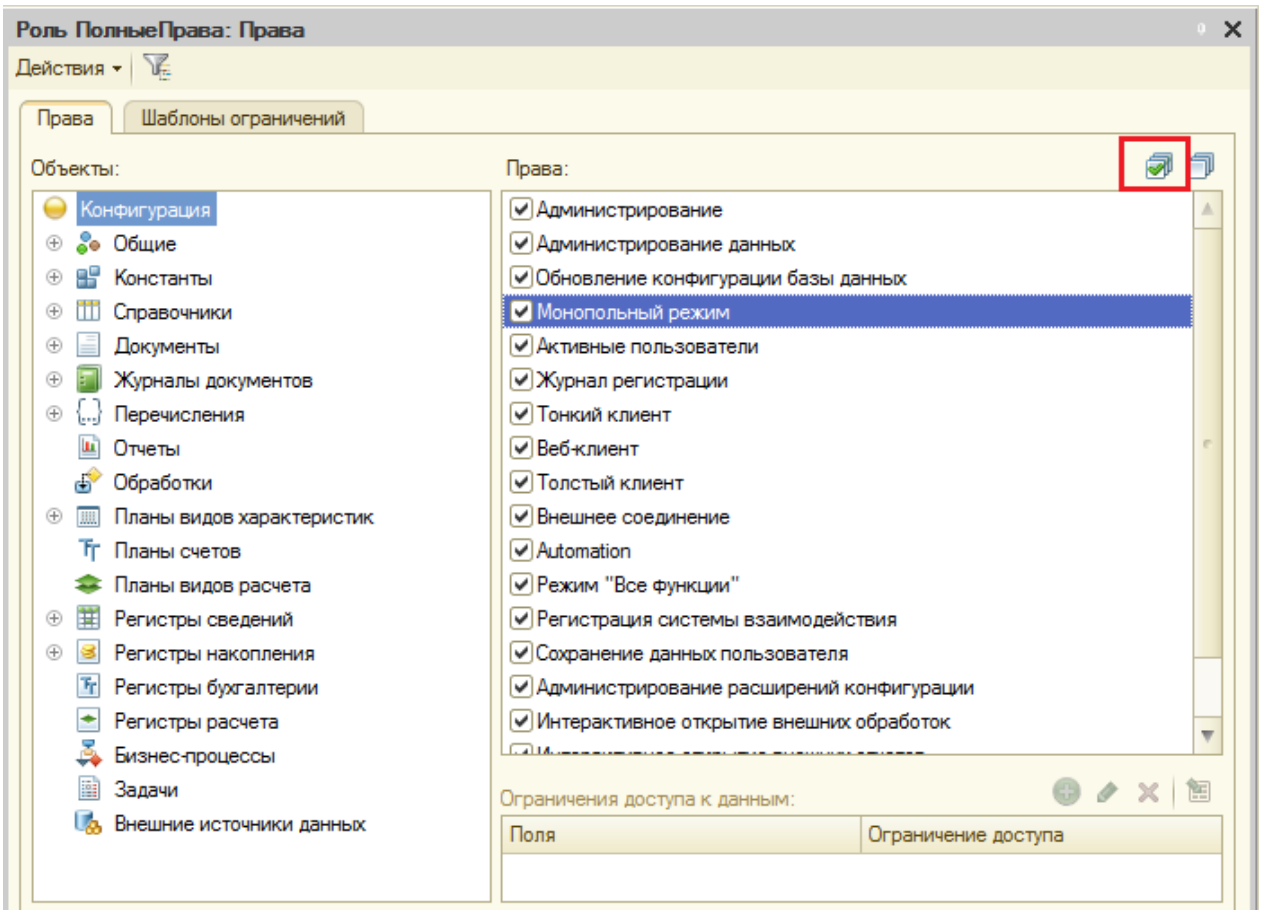


Рис. 4.1.176

Всё! Роль с полными правами создана. Теперь создадим пользователя с полными правами. Перед этим необходимо обновить конфигурацию базы данных.

Для создания пользователя базы данных в конфигураторе нужно войти в список пользователей. Список пользователей доступен по следующему пути: «Главное меню» – «Администрирование» – «Пользователи».

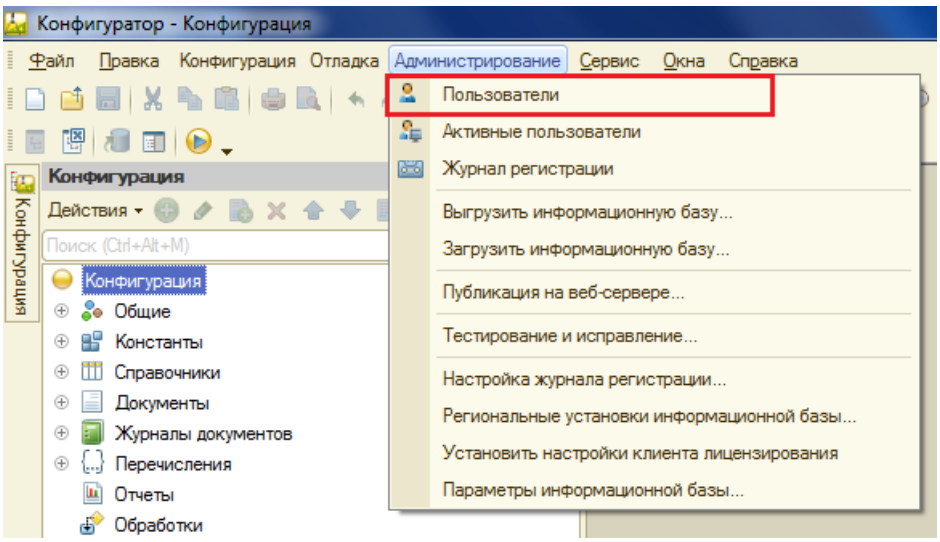


Рис. 4.1.177

В открывшемся списке необходимо нажать на кнопку «Добавить», после этого выйдет окно добавления нового пользователя.

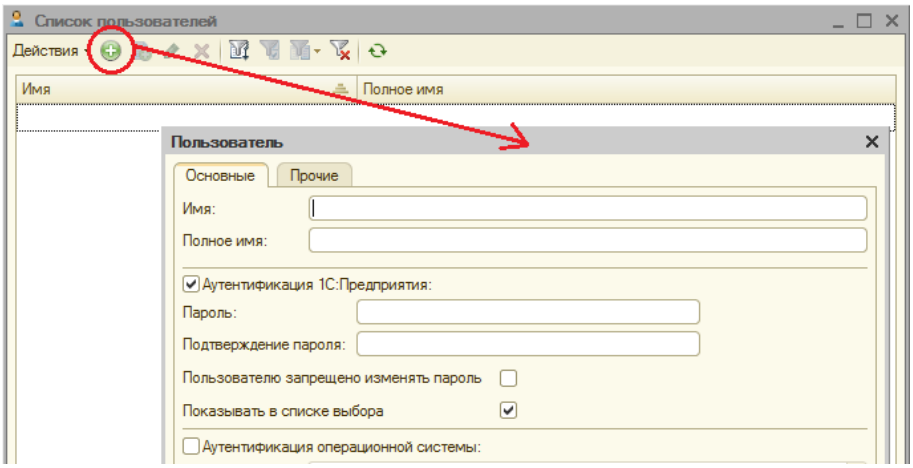


Рис. 4.1.178

Введем имя «Администратор», все остальное оставим как есть и перейдем на закладку «Прочее».

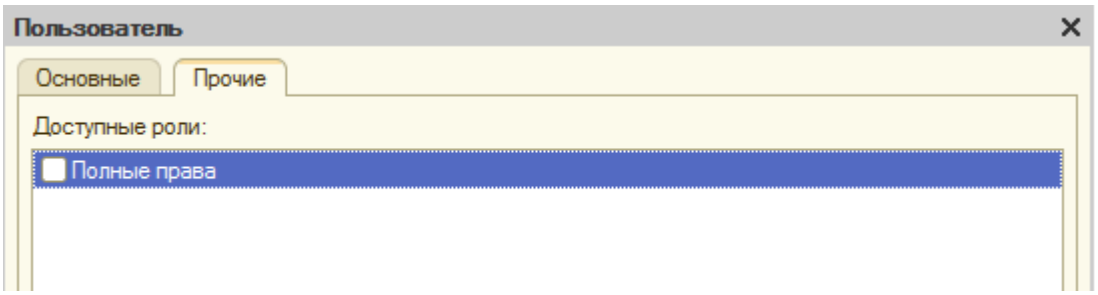


Рис. 4.1.179

В этой закладке нас интересует список «Доступные роли». Пока в нем только одна роль - «Полные права». Поставим у этой роли флаг и нажмем кнопку «ОК» – пользователь создан. С этого момента, когда Вы будете входить в «1С:Предприятие» (без разницы, в пользовательский режим, или в конфигуратор), платформа потребует выбрать пользователя. Пока у нас только один пользователь – Администратор.

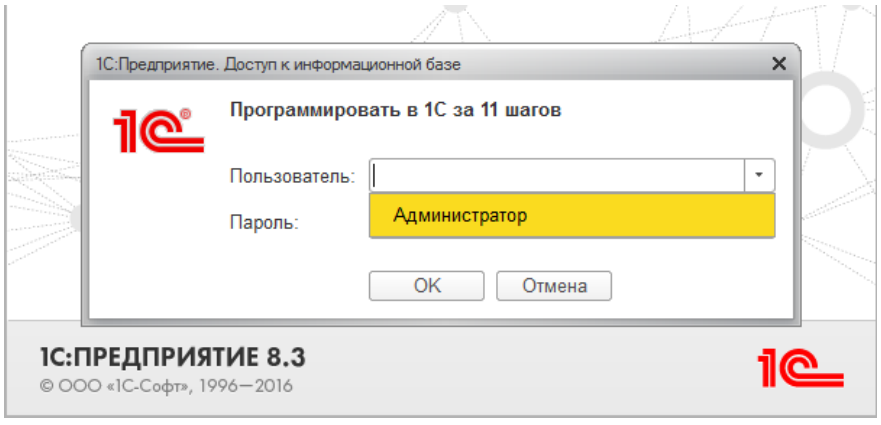


Рис. 4.1.180

Теперь создадим ещё одну роль, точно так же, как и роль *ПолныеПрава* (см. рис. 4.1.172). Для новой роли дадим название *Диспетчер* и в открывшемся конструкторе прав установим право «Толстый клиент» у конфигурации, остальной набор прав *для конфигурации* оставим как есть.

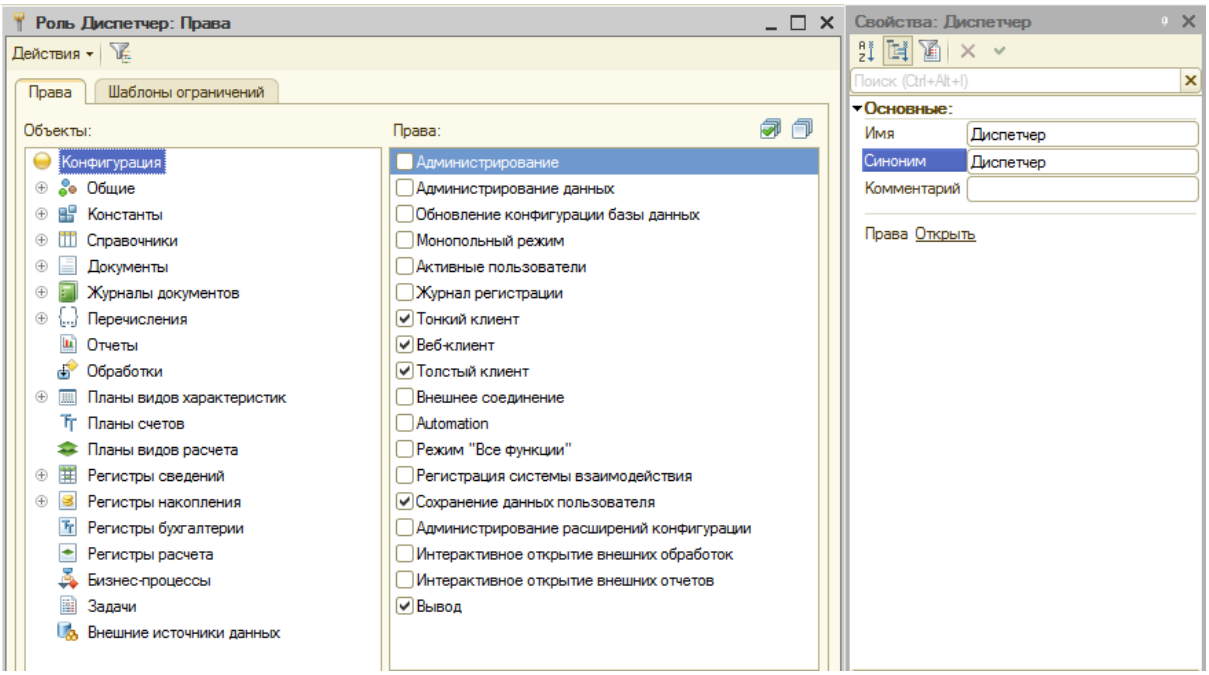


Рис. 4.1.181

Теперь необходимо для ряда объектов установить полные права (см. табл. 4.1.2), а для ряда – урезанные: только для чтения и просмотра.

В рамках этой книги мы не будем подробно разбирать смысл того или иного права, изучим только общие моменты.

Установим у роли «Диспетчер» для справочника «Марки автомобилей» полные права (кроме интерактивного удаления). Найдем нужный справочник в дереве конфигурации конструктора прав роли.

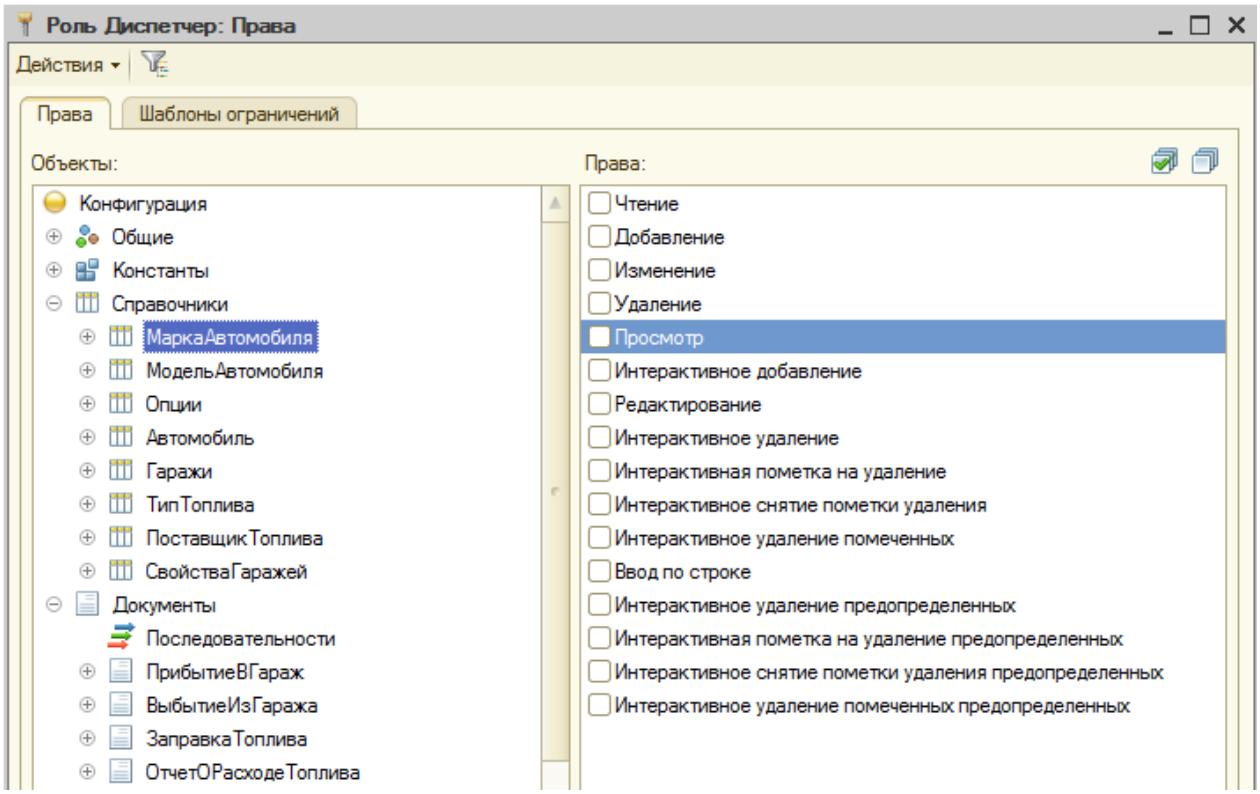


Рис. 4.1.182

Установим для этого элемента метаданных флажки всех прав, кроме права «Интерактивное удаление» и «Интерактивное удаление предопределенных».

*Право **Интерактивное удаление** дает пользователю возможность удалить объект непосредственно без контроля ссылочной целостности, что не всегда корректно. Подробно удаление объектов будем изучать в шестой главе.*

Для быстрой отметки воспользуемся кнопкой «Пометить все элементы», а отметки прав «Интерактивное удаление» и «Интерактивное удаление предопределенных» снимем вручную. Должен получиться следующий набор прав:

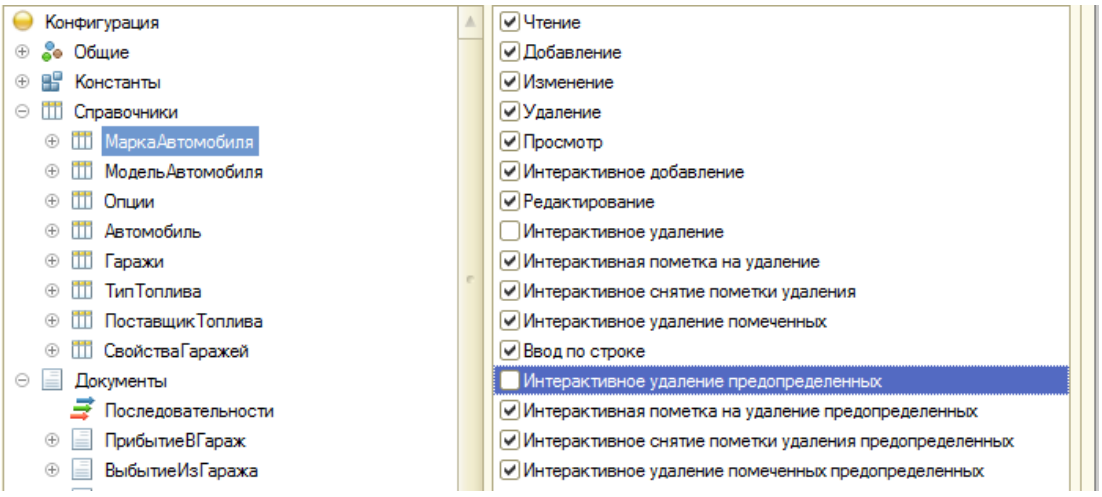


Рис. 4.1.183

Для документа «Прибытие в гараж» полный набор прав будет выглядеть следующим образом:

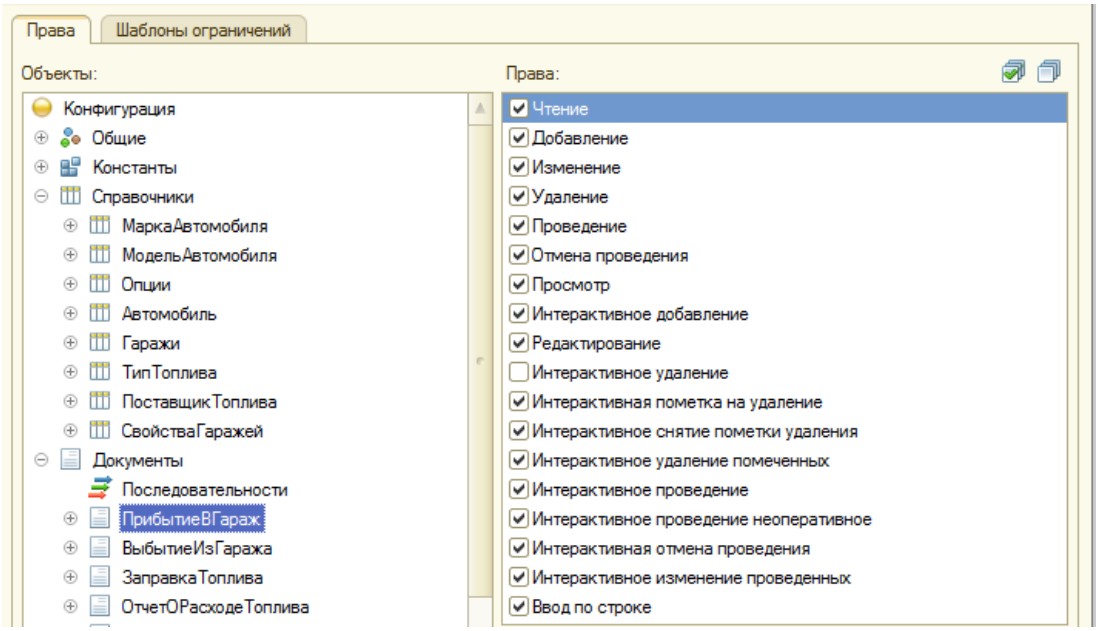


Рис. 4.1.184

А для справочника «Тип топлива» дадим права только на чтение и просмотр, в этом случае будет следующий набор прав:

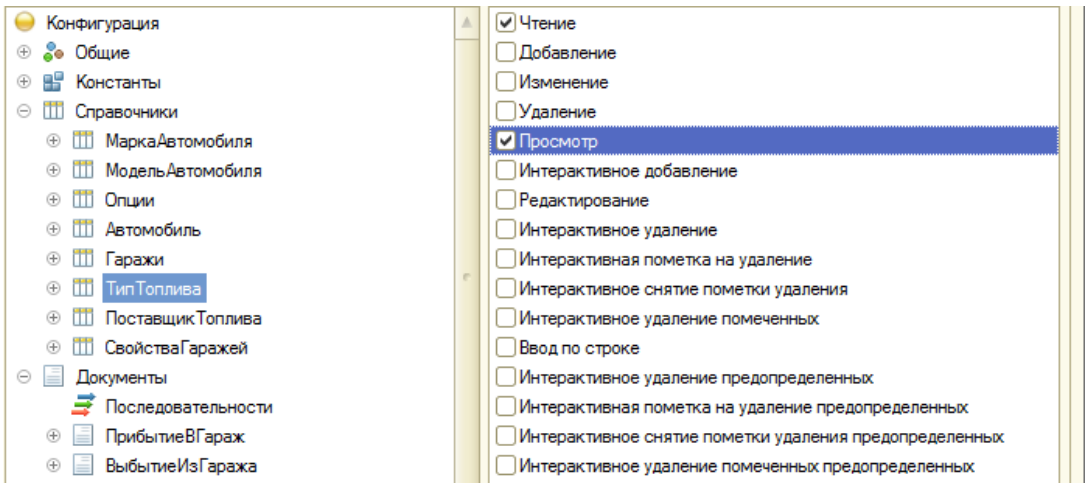


Рис. 4.1.185

Теперь самостоятельно установите права на объекты согласно таблице 4.1.2 для роли *Диспетчер*, потом создайте роль *УчетчикТоплива* (не забудьте поставить право конфигурации *Толстый клиент*) и также самостоятельно установите права для этой роли согласно всё той же таблице 4.1.2. Для новых прав создайте пользователей «Диспетчер» и «Учетчик» с соответствующими наборами ролей. Проверьте доступ к объектам из таблицы 4.1.2.

Общие картинки

Научимся добавлять в конфигурацию 1С общие картинки, которые потом можно использовать в декорациях или при создании подсистем.

Для добавления общей картинки необходимо выделить соответствующий пункт ветки «Общие», вызвать контекстное меню и в этом меню нажать на кнопку «Добавить».

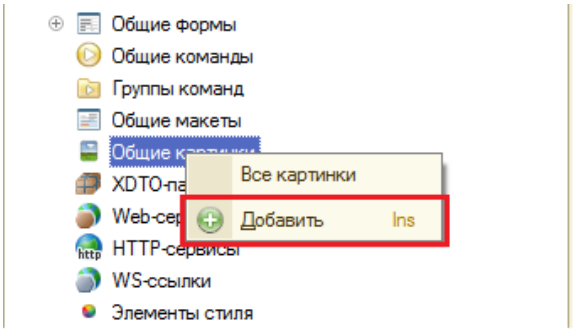


Рис. 4.1.186

После добавления откроется конструктор общей картинки, где можно внести название картинки, а также выбрать произвольную картинку из файловой системы, нажав на кнопку «Выбрать из файла» (см. рис. 4.1.187).

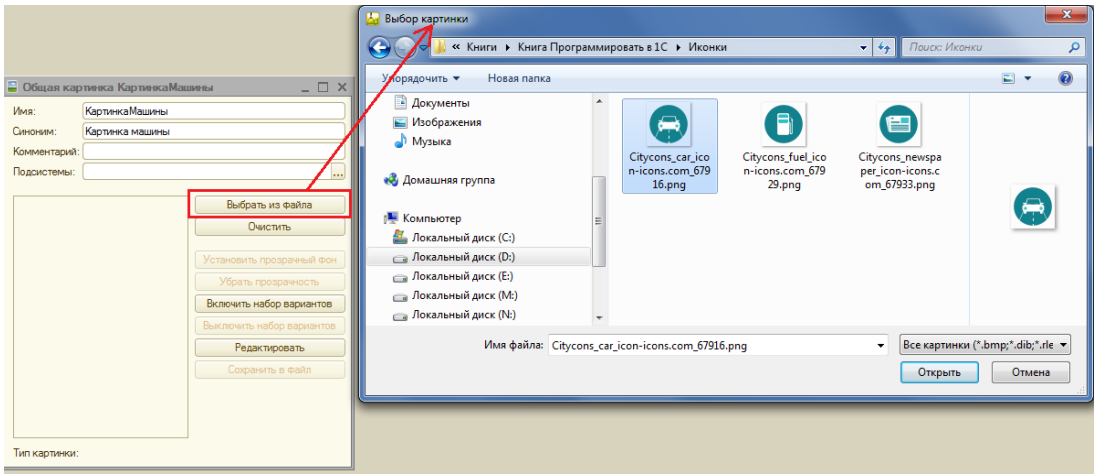


Рис. 4.1. 187

После выбора картинка появится в конструкторе.

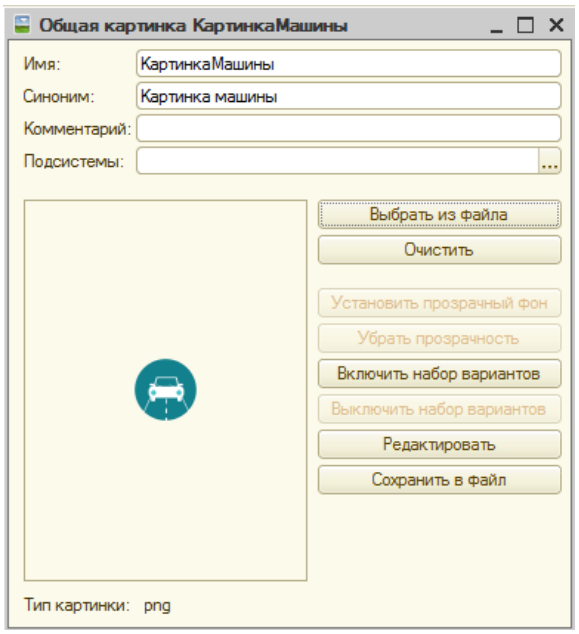


Рис. 4.1.188

И соответствующий пункт появится в ветке «Общие картинки».

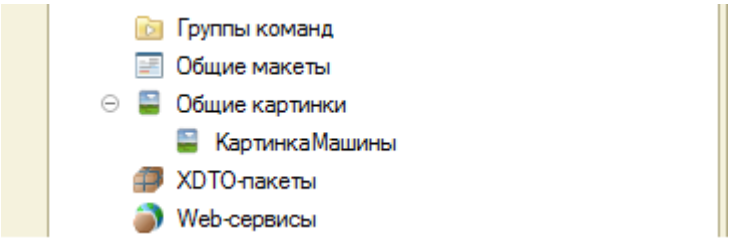


Рис. 4.1.189

В дальнейшем эту картинку можно использовать при работе с формами, подсистемами и пр.

Подсистемы

И напоследок мы изучим *Подсистемы*. Как Вы заметили, обращаться к справочникам, документам и прочим объектам конфигурации, которые свалены в меню функций текущего раздела, не совсем удобно. Логичнее все объекты разбить по каким-то разделам. Сделать это можно при помощи объекта *Подсистемы*. Очень подробно работа с этим объектом дается в моей книге «Основы разработки в 1С: Такси. Разработка в управляемом приложении за 12 шагов». Здесь мы коснемся только самых общих моментов, необходимых для дальнейшей работы.

Прежде чем начать создавать подсистемы, определим, сколько их будет, как они будут называться и какие объекты войдут в их состав. Самое логичное - конструировать подсистемы по прикладным областям. В нашей конфигурации можно сделать три подсистемы: «Учет автомобилей», «Учет топлива» и «НСИ» (нормативно-справочная информация). Причем обратите внимание: подсистемы «Учет автомобилей» и «Учет топлива» по своей сути соответствуют ролям

«Диспетчер» и «Учетчик топлива». Я распределю объекты конфигурации по подсистемам следующим образом (по аналогии с ролями):

Подсистема	Объект конфигурации
Учет автомобилей	Справочники: Марки автомобилей Модели автомобилей Опции Автомобиль Гаражи Свойства гаражей Документы: Прибытие в гараж Выбытие из гаража Журнал: Прибытие-выбытие автомобилей Планы видов характеристик: Дополнительные свойства гаражей Регистры сведений: Основной гараж автомобиля Значение свойств гаражей Регистры накопления: Пробег автомобиля
Учет топлива	Справочники: Тип топлива Поставщик топлива Документы: Заправка топлива Отчет о расходе топлива Установка цен на топливо Регистры сведений: Цены на топливо Регистры накопления: Топливо в автомобилях
НСИ	Справочники: Марки автомобилей Модели автомобилей Опции Автомобиль Гаражи Свойства гаражей Тип топлива Поставщик топлива Планы видов характеристик: Дополнительные свойства гаражей Регистры сведений: Основной гараж автомобиля Значение свойств гаражей Константы и форма констант, перечисления, общие картинки, параметры сеанса, предопределяемые типы, роли

Табл. 4.1.3

Как видите, объекты в подсистемах могут пересекаться. Создадим нашу первую подсистему «Учет автомобилей». Для этого в дереве метаданных конфигурации в ветке «Общие» необходимо выделить ветку «Подсистемы», вызвать правой кнопкой контекстное меню, в котором нажать на пункт «Добавить»

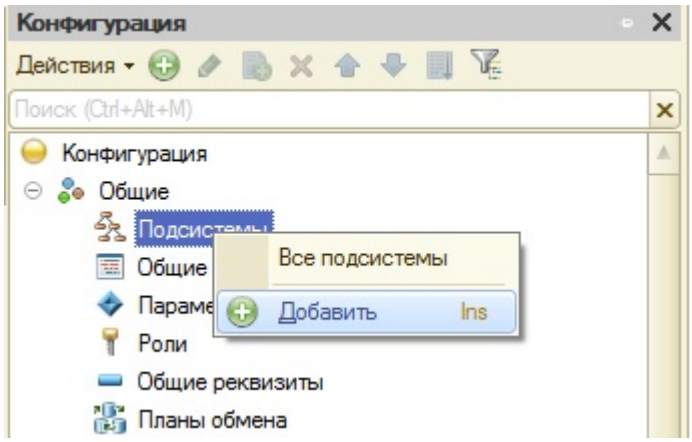


Рис. 4.1.190

Откроется окно редактора подсистем, где мы вводим название и, самое главное, оставляем включенным флаг «Включать в командный интерфейс».

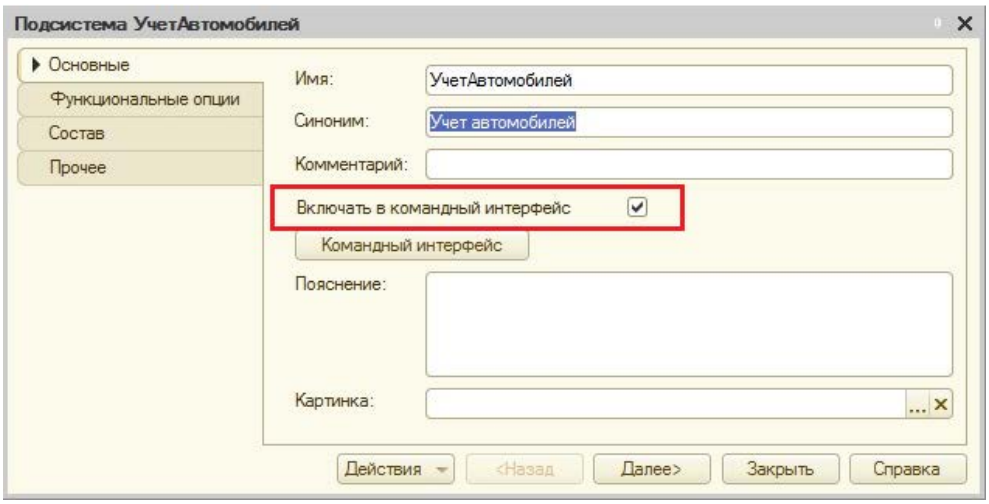


Рис. 4.1.191

Если этот флаг включен, то Ваша подсистема будет отображена в виде закладке в пользовательском режиме «1С:Предприятия».

Следующим шагом мы привяжем наши объекты из таблицы выше к подсистеме. Для этого необходимо перейти на закладку «Состав» и в верхнем окне проставить флажки рядом с нужными объектами, в нижнем они отобразятся автоматически.

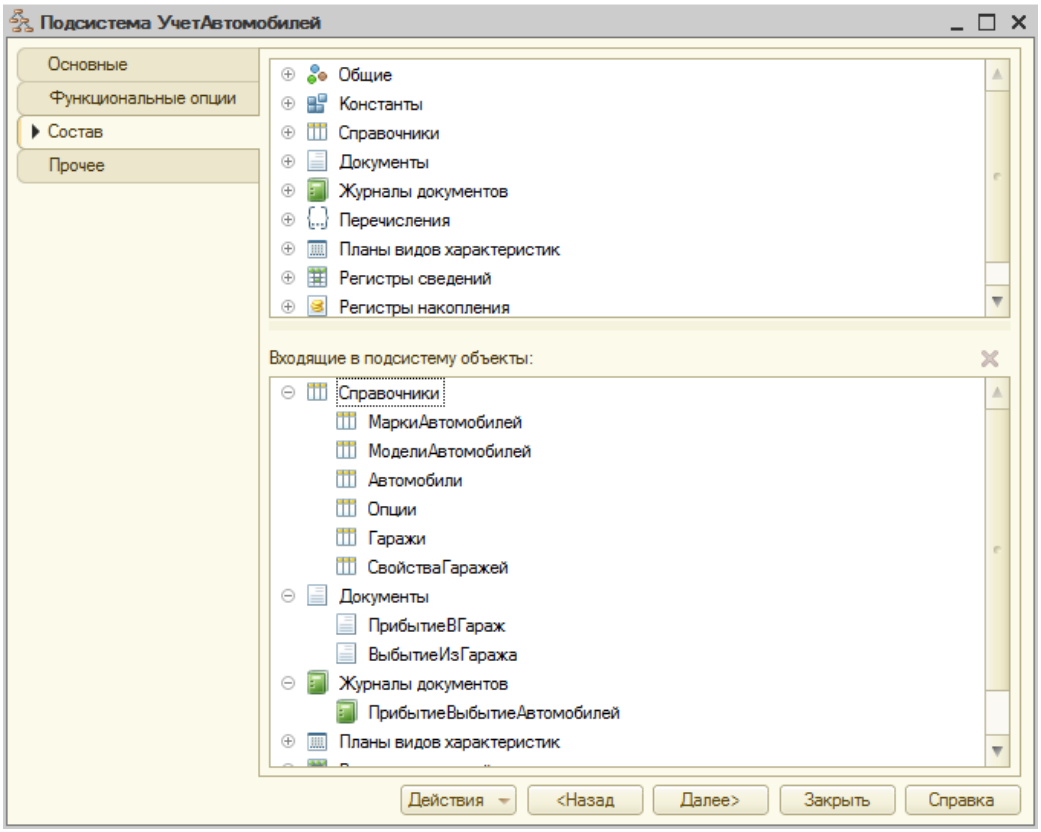


Рис. 4.1.192

Самостоятельно создайте остальные подсистемы и настройте их состав согласно таблице выше. Если Вы забудете какой-либо объект включить в подсистему, то при обновлении базы данных выйдет соответствующее сообщение в конфигураторе.

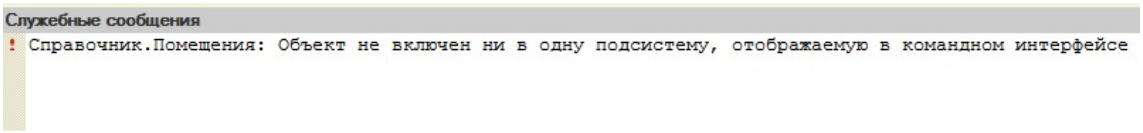


Рис. 4.1.193

И последний шаг: обязательно проверьте у роли «Полные права», установилось ли у всех новых подсистем *право на просмотр* для этой роли, иначе зайдя под администратором, Вы не увидите закладок, которые привязаны к подсистемам.

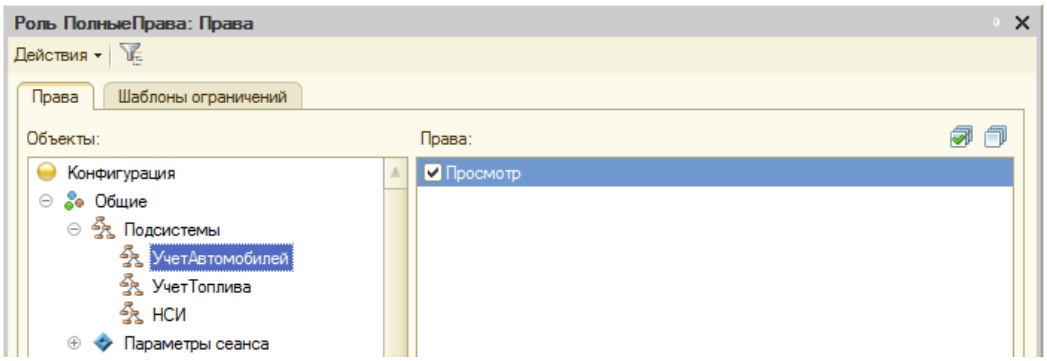


Рис. 4.1.194

После обновления базы данных и перезапуска пользовательского приложения «1С:Предприятия» Вы увидите пустой интерфейс.

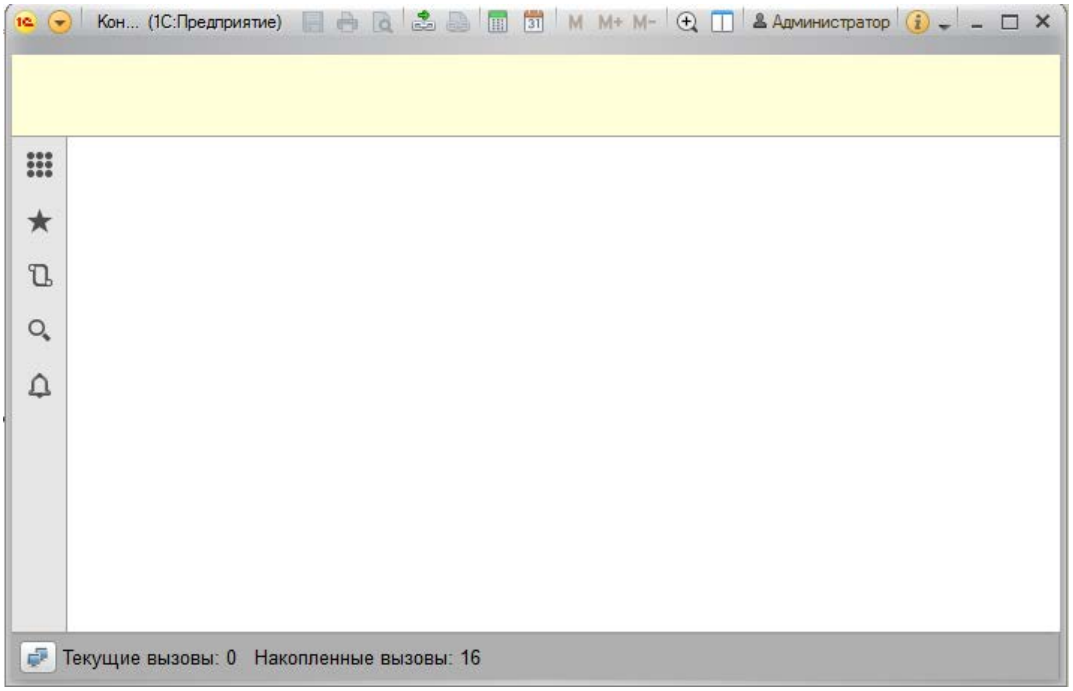


Рис. 4.1.195

Для отображения разделов необходимо добавить соответствующую панель.

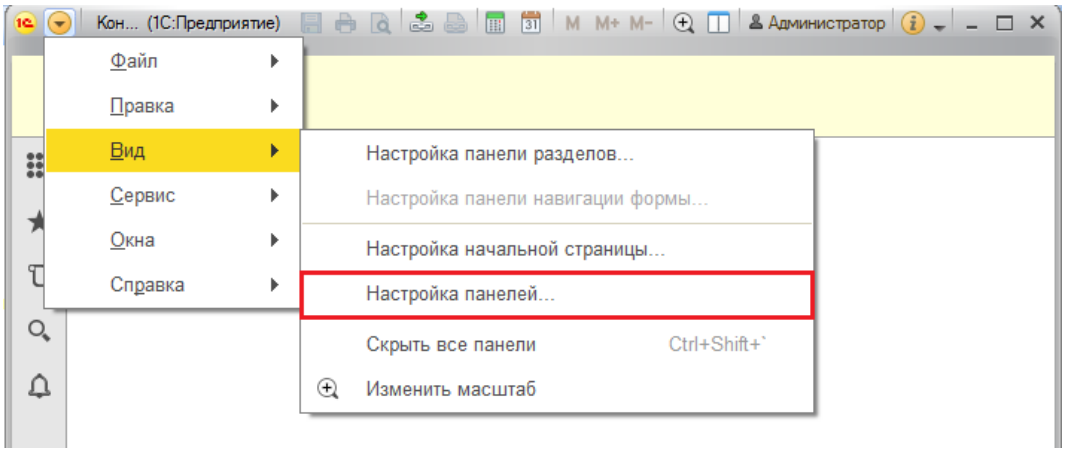


Рис. 4.1.196

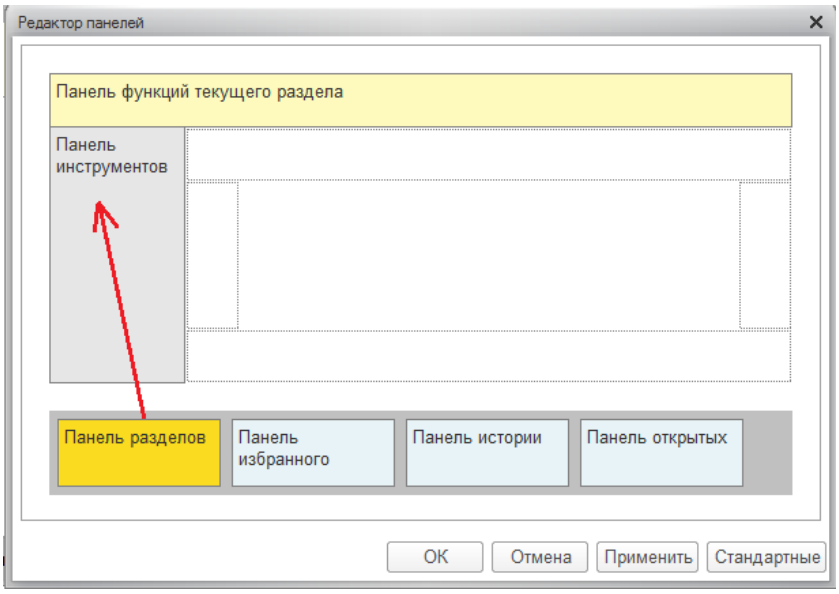


Рис. 4.1.197

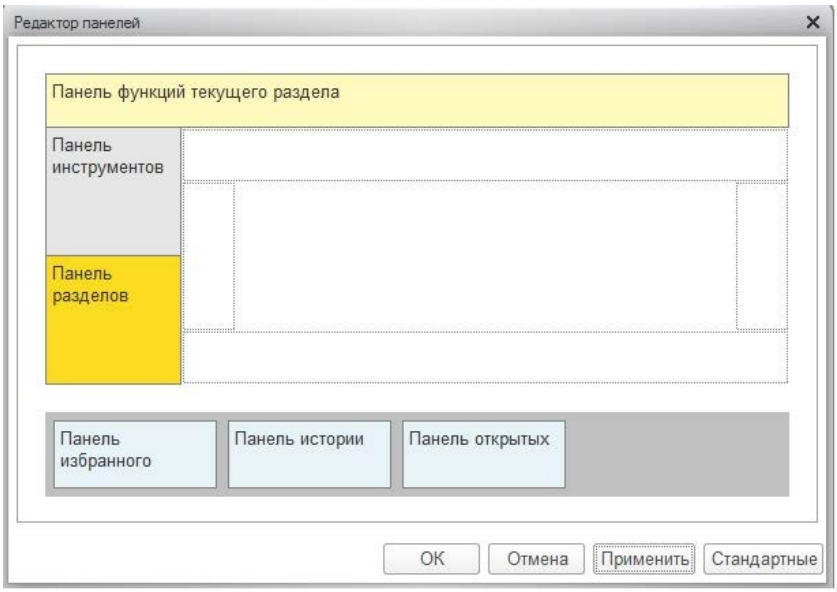


Рис. 4.1.198

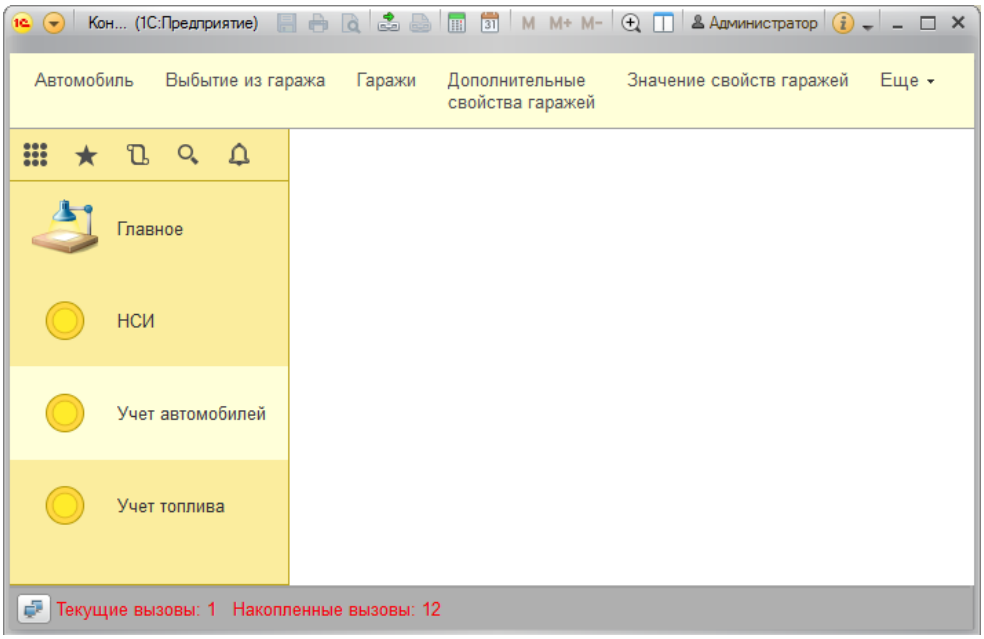


Рис. 4.1.199

Установите для роли «Диспетчер» самостоятельно право на просмотр следующих подсистем: «Учет автомобилей» и «НСИ», а для роли «Учетчик топлива» - «Учет топлива» и «НСИ», и проверьте, как будет отображаться интерфейс при заходе в программу под пользователями «Диспетчер» и «Учетчик».

Более подробно о работе с подсистемами читайте в книге «Основы разработки в 1С: Такси. Разработка в управляемом приложении за 12 шагов».

Резюме

На этом мы закончим изучать основы конфигурирования программы 1С. Знаний, которые предоставлены в этой главе, Вам вполне хватит, чтобы окунуться в мир программирования 1С и начать осваивать работу с конфигуратором. В следующей части мы изучим модули.

Часть 2. Модули.

Во второй части этой главы мы разберем модуль управляемого приложения, модуль объекта, модуль менеджера объекта и модуль формы. Выполняйте все примеры, приведенные в листингах.

Прежде чем рассматривать работу модулей, разберем небольшую тему - это локальные и глобальные переменные.

Локальные и глобальные переменные

Рассмотрим такое важное понятие в языке программирования, как *Локальные и Глобальные переменные.*

Что такое *Локальные переменные*? Это любые переменные, описанные внутри процедуры или функции. Как задаются переменные внутри процедур и функций, мы знаем: либо через слово *Перем*, либо если переменной сразу будет присвоено какое-нибудь значение (см. листинг 4.2.1).

```
Процедура НашаПроцедура()

    Перем А,Б;

    В = 10;
    Г = "Привет, мир!"

КонецПроцедуры
```

Листинг 4.2.1

В листинге 4.2.1 переменные *А* и *Б* заданы явным способом, а *В* через значение. Локальные переменные создаются при запуске процедуры или функции и уничтожаются, когда процедура или функция заканчивает свою работу. Т.е. Вы не сможете к переменным, заданным в процедуре *НашаПроцедура*, обратиться в другой процедуре или просто в модуле.

Теперь - как задать переменные глобально. Рассмотрим пример в рамках модуля формы (создайте новую обработку и форму). Для этого в самом верху модуля формы пишем слово *Перем* и перечисляем переменные через запятую, а в конце пишем точку с запятой. Так же, как и в случае процедур или функций, в модуле формы *управляемого приложения* перед переменными необходимо указывать директивы компиляции *&НаКлиенте* или *&НаСервере*. В этой части все переменные и методы в модуле формы будут выполняться в клиентском контексте (под директивой *&НаКлиенте).*

Директивы компиляции будем изучать в пятой главе.

Смотрите листинг 4.2.1. Переменные А,Б,В – заданы локально, а переменные А0,Б0,В0 – глобально.

```
&НаКлиенте
Перем А0, Б0;
&НаКлиенте
Перем В0;

&НаКлиенте
Процедура НашаПроцедура()

    Перем А,Б;

    В = 100;

КонецПроцедуры
```

Листинг 4.2.2

После того, как мы задали глобальные переменные в начале модуля, мы можем обращаться к ним в любой процедуре или функции модуля. Выполните пример с листинга 4.2.3. Самостоятельно создайте команду и обработчик команды в модуле формы.

```
&НаКлиенте
Перем А, Б;

&НаКлиенте
Процедура ИзменитьЗначенияПеременных()
    А = А + 100;
    Б = А;
КонецПроцедуры

&НаКлиенте
Процедура ВыполнитьКоманду(Команда)
    Сообщить("А = " + А);
    Сообщить("Б = " + Б);
    ИзменитьЗначенияПеременных();
    Сообщить("А = " + А);
    Сообщить("Б = " + Б);
КонецПроцедуры

А = 10;
Б = -100;
```

Листинг 4.2.3

Разберем данный код. В начале нашего модуля мы задаем переменные *А* и *Б.* В конце модуля присваиваем им значения. Почему значения присваиваются в конце модуля? Потому что процедуры и функции должны описываться непосредственно перед кодом, который пишется в самом модуле. Да, в модуле формы мы можем написать почти любой код, и он выполнится во время открытия формы.

После мы создаем процедуру *ИзменитьЗначенияПеременных*, в которой манипулируем с нашими переменными. И в обработчике команды *ВыполнитьКоманду* работаем с нашими глобальными переменными.

Рис. 4.2.1

Т.е. сначала переменные *А* и *Б* были такими, какими Вы задали их в конце модуля, но после выполнения процедуры *ИзменитьЗначенияПеременных* они изменились.

А что будет, если имея глобальную переменную, внутри процедуры или функции задать локальную переменную с таким же именем?

Выполните пример из листинга 4.2.4, и Вы увидите, что как *А* была равна *1*, так и осталась. Потому что, задав явно переменную внутри процедуры, мы создали локальную переменную, пусть с тем же названием, но абсолютно другую. Как только процедура выполнилась, данная локальная переменная уничтожилась.

Задание локальных переменных работает только в том случае, если Вы ее задали именно явно, через ключевое слово *Перем*.

```
&НаКлиенте
Перем А;

&НаКлиенте
Процедура ВыполнитьКоманду(Команда)
    Сообщить("А = " + А);
    ИзменитьПеременную();
    Сообщить("А = " + А);
КонецПроцедуры

&НаКлиенте
Процедура ИзменитьПеременную()
    Перем А;
    А = 10000;
КонецПроцедуры

А = 1;
```

Листинг 4.2.4

Рис. 4.2.2

Модули

Переходим к модулям. Для работы с данной частью я рекомендую открыть Вам ту конфигурацию, которую Вы сделали в первой части этой главы.

Что такое *Модуль*? По-простому, *Модуль* - это непосредственно та область, в которой программист пишет свои алгоритмы, процедуры и функции. Но особенности программы 1С в том, что модуль это не только текстовый редактор, по сути, это программа на языке программирования 1С. И в зависимости от того, в каком модуле Вы опишете ту или иную процедуру, она будет выполняться на том или ином уровне.

К примеру, процедуры, написанные в модуле формы, будут работать только тогда, когда открыта или создана (это не одно и то же) данная форма. Вы не сможете обратиться к процедурам формы элемента справочника, если обратитесь программно к справочнику как объекту.

Поэтому разрабатывая или дорабатывая программное обеспечение, Вы должны четко понимать, на каком уровне Вы будете писать Ваши алгоритмы. А какие есть уровни, мы выясним в этой части главы.

Каждый программный модуль связан с остальными объектами конфигурации, эта связь называется *Контекстом*. Всего существует два вида контекста: *Глобальный* и *Локальный. Глобальный контекст* виден для всех, он, по сути, создает общую среду работы программы. *Локальный контекст* действует только в рамках данного объекта. Т.е., если мы создадим процедуру в рамках глобального контекста и назовем ее, к примеру, *РасчетСебестоимости*:

РасчетСебестоимости()

- то мы в любом месте программы можем написать имя данной процедуры, и она без проблем выполнится. А если мы создадим процедуру, назовем ее *Рассчитать пробег*:

РассчитатьПробег()

для объекта справочника *Автомобили* в модуле объекта этого справочника и сделаем ее экспортной (про экспорт процедур и функций мы узнаем позже), то обратиться к ней мы сможем только через данный объект справочника:

```
ОбъектСправочника = СправочникСсылка.ПолучитьОбъект();

ОбъектСправочника. РассчитатьПробег();
```

Теперь изучим сами виды модулей, их много, но мы изучим основные, которые чаще всего необходимы в работе.

Начнем с самого низа - это уровень формы.

Модуль формы

Модуль формы должен содержать те процедуры и функции, которые необходимы только для работы данной формы. В *управляемом приложении* процедуры или функции модуля формы выполняются или в *серверном* контексте или в *клиентском* контексте. Смысл клиентского и серверного контекста мы будем проходить в следующей главе.

В модуле формы можно создавать и описывать обработчики событий элементов формы и самой формы, а также обработчики событий команд формы. Код в описанных обработчиках событий будет выполняться тогда, когда будет выполнена команда, к которой привязан обработчик события, или когда с элементом формы (самой формой), произойдет событие, с которым связан обработчик.

С модулем формы мы работали с самого начала курса, и Вы знаете, что попасть в модуль формы легко, для этого достаточно открыть форму и зайти в закладку *«Модуль»*.

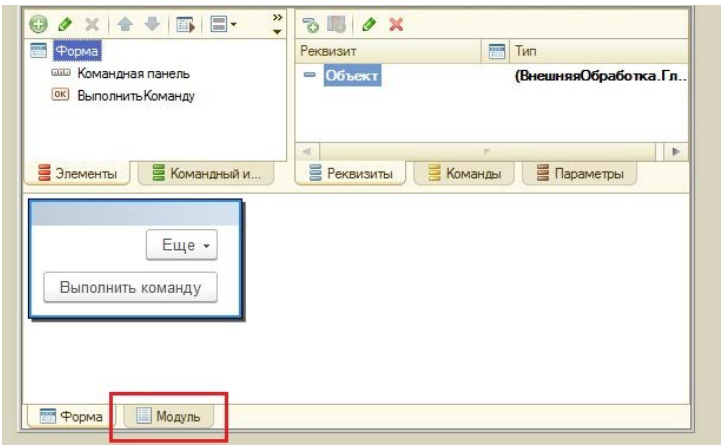


Рис. 4.2.3

А чтобы увидеть список всех процедур и функций модуля формы, необходимо сфокусироваться на модуле формы и выполнить команду «Процедуры и функции» в меню «Текст» («Главное меню» – «Текст»).

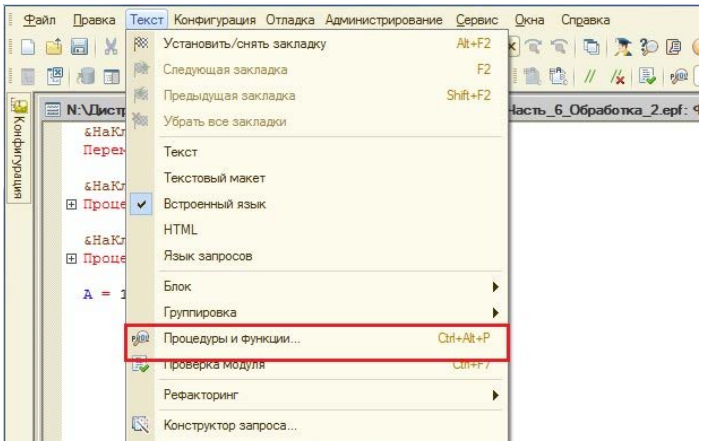


Рис. 4.2.4

После выполнения команды откроется список процедур и функций, которые уже есть в модуле формы. Причем в этом списке также присутствуют обработчики событий формы, которые не созданы (выделены треугольными скобками - < >). У созданных процедур и функций слева в списке отображается соответствующий значок.

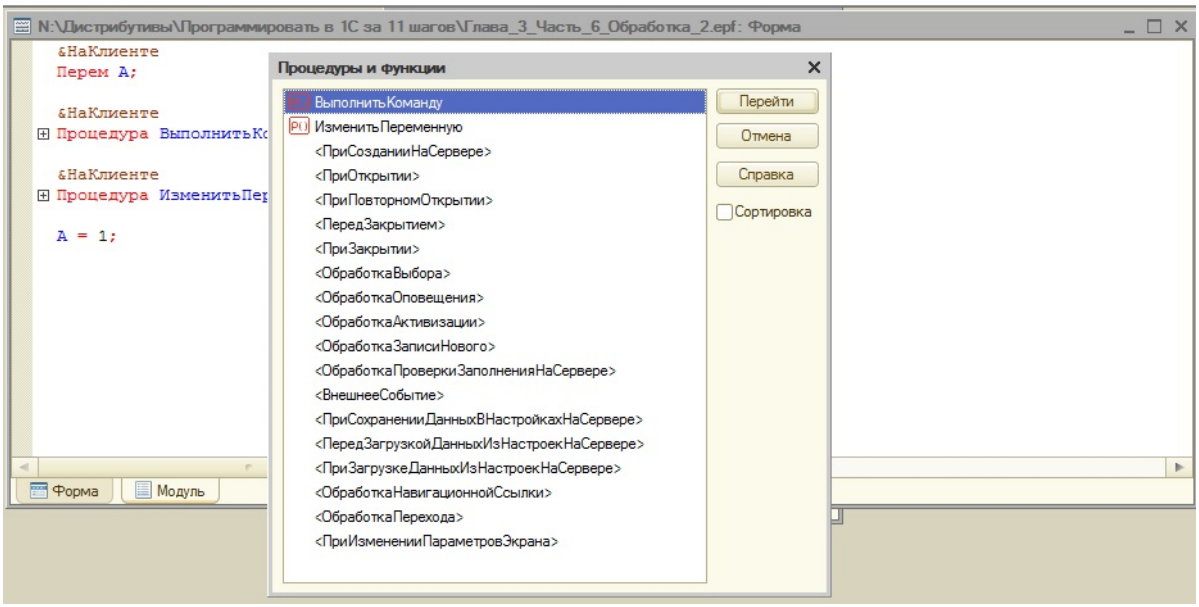


Рис. 4.2.5

Более подробно работу с формами мы будем проходить в пятой главе.

Модуль объекта

Модуль объекта предназначен для описания процедур и функций, которые необходимы для работы конкретного элемента объекта конфигурации. Это может быть конкретный элемент справочника, или элемент документа.

Поскольку создать элемент справочника, документа можно не только интерактивно, но и программно, то данный модуль призван описать те события, процедуры и функции, которые должны выполняться именно на уровне элемента объекта, без разницы, как он создан: программно или интерактивно пользователем.

Кроме того, в модуле объекта можно создавать и описывать обработчики событий, которые возникают при работе с объектом. Например, это может быть обработка события записи объекта, или обработка события проведения объекта (для документов).

Подробно обработчики событий объекта будем проходить в шестой главе. Пока же запомним, что все методы (процедуры и функции) модуля объекта можно разделить на две группы: это типовые обработчики различных событий объекта, и просто процедуры (функции) созданные разработчиками для осуществления каких-либо действий над экземпляром объекта (например, пересчет всех цен в табличной части после изменения типа цен в шапке документа).

Это что касается документов и справочников. Когда же Вы работаете с обработками или отчетами, в модуле обработки и отчета должны быть описаны те процедуры и функции, к которым понадобиться доступ тогда, когда Вы программно обратитесь к этому отчету или обработке.

Как попасть в модуль объекта? Для элементов конфигурации это достаточно просто: в дереве метаданных выделяете интересующий Вас объект, кликаете правой кнопкой мышки и нажимаете на пункт *«Открыть модуль объекта»*.

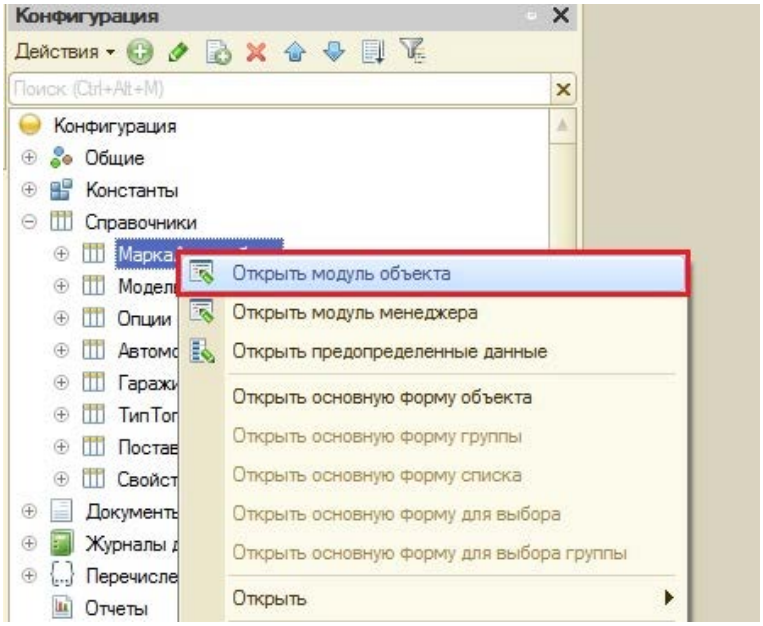


Рис. 4.2.6

Для внешних отчетов и обработок: находясь непосредственно в менеджере обработки, нажимаем на кнопку *«Действия»* и выбираем *«Модуль объекта»*.

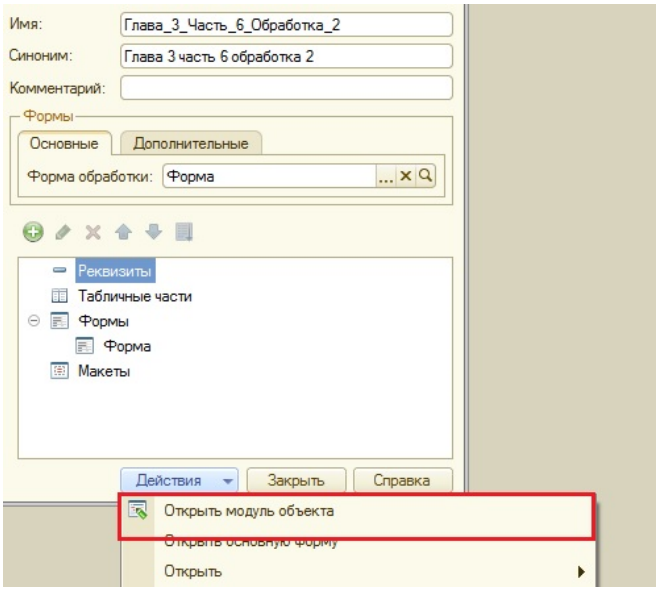


Рис. 4.2.7

Для того чтобы посмотреть на полный список процедур и функций модуля объекта, необходимо сфокусироваться на данном модуле, и перейти *«Текст» – «Процедуры и функции».*

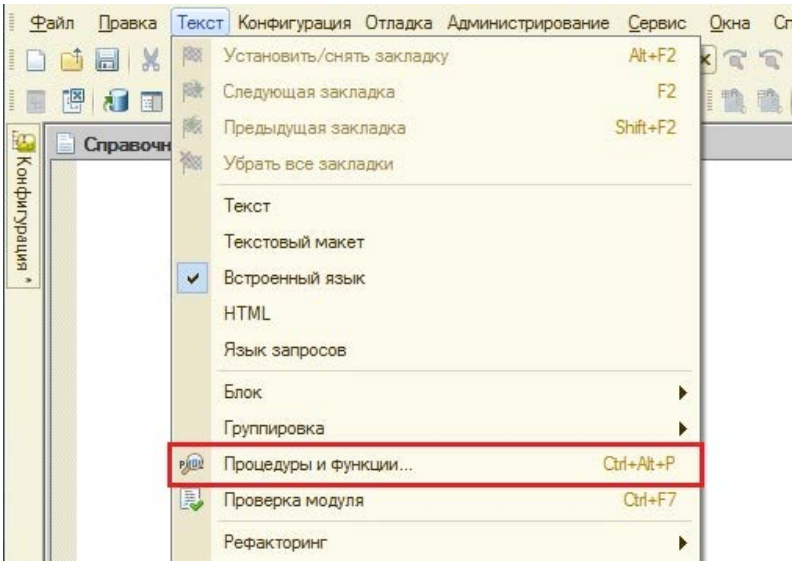


Рис. 4.2.8

Вы увидите все имеющиеся процедуры.

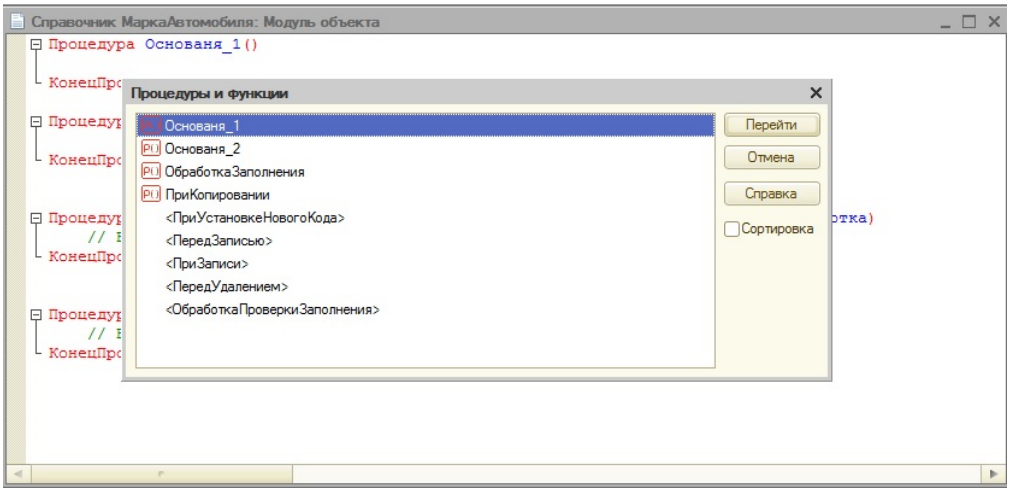


Рис. 4.2.9

Значками в левой части списка будут обозначены уже созданные процедуры и функции (в том числе обработчики событий), а треугольными скобками (< >) выделяются еще не созданные обработчики событий объекта.

Если Вам надо будет создать какую-нибудь процедуру обработчик события, просто выберите ее в этом списке и дважды кликните левой клавишей мышки.

***Обращаю Ваше внимание**, что для документов и справочников, если Вы будете делать какие-нибудь проверки при записи объекта или проверки на заполнение реквизитов при сохранении объекта, делайте это все на уровне объекта, а не формы. Поскольку возможны случаи, когда Вам придется записывать этот документ программно, и тогда Ваша проверка на форме не сработает.*

Модули менеджера объекта

Если в модуле объекта описываются методы, предназначенные для работы с конкретным экземпляром объекта (подробно об экземплярах объектов в шестой главе), то в модуле менеджера объекта описываются методы, которые не привязаны к конкретному экземпляру объекта, но привязаны к объекту в целом.

Например, мы можем написать обработку печати товарной накладной документа в модуле объекта, и тогда для печати документа нам нужно будет каждый раз получать экземпляр этого документа. А можем написать обработку печати в модуле менеджера объекта и сделать её экспортной (об этом чуть позже), тогда достаточно передать ссылку (или массив ссылок) на распечатываемый документ в процедуру печати.

В случае, когда процедура печати в **модуле объекта**, код для печати будет выглядеть так:

```
ОбъектСправочника = СправочникСсылка.ПолучитьОбъект();
ОбъектСправочника.Печать();
```

А когда процедура печати в **модуле менеджера объекта**, код будет следующий:

```
Документы.ВыбытиеИзГаража.Печать(СсылкаНаДокумент);
```

Получить доступ в модуль менеджера объекта можно двумя способами (см. рис. 4.2.10 и 4.2.11).

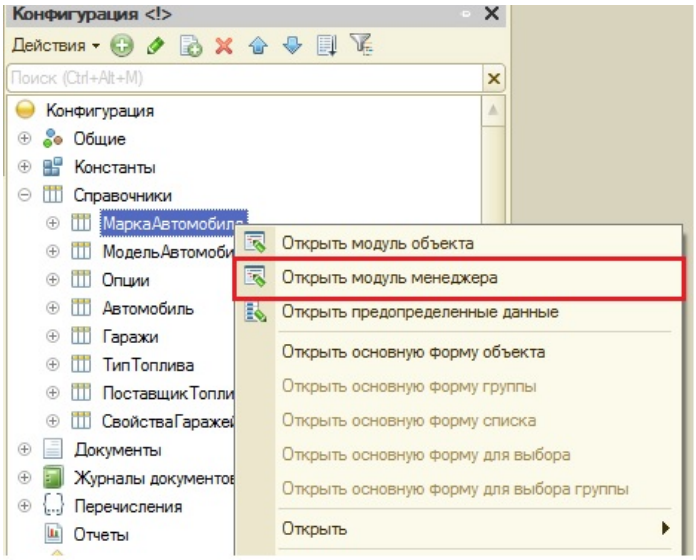


Рис. 4.2.10

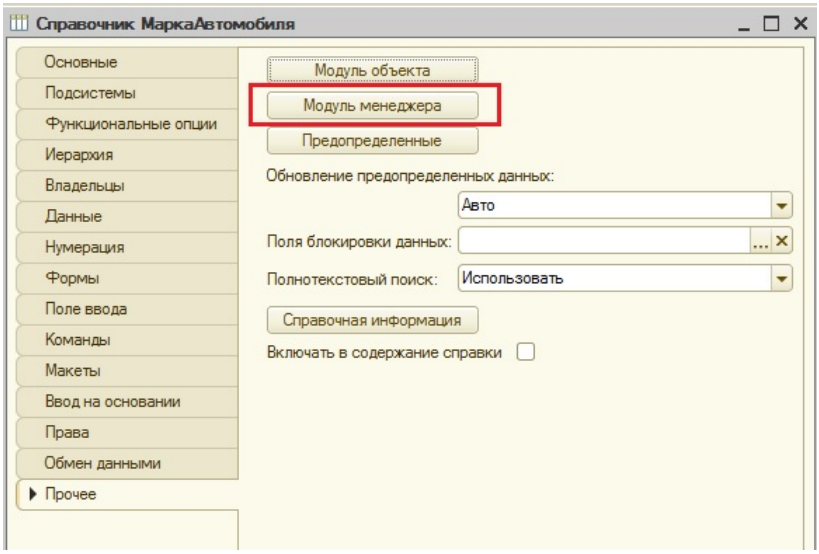


Рис. 4.2.11

Кроме того, в модуле менеджера можно описывать обработчики некоторых типовых событий, которые могут возникать при работе с менеджером объекта (например, при получении данных выбора).

Общие модули

Общие модули можно найти в отдельной ветке конфигурации:

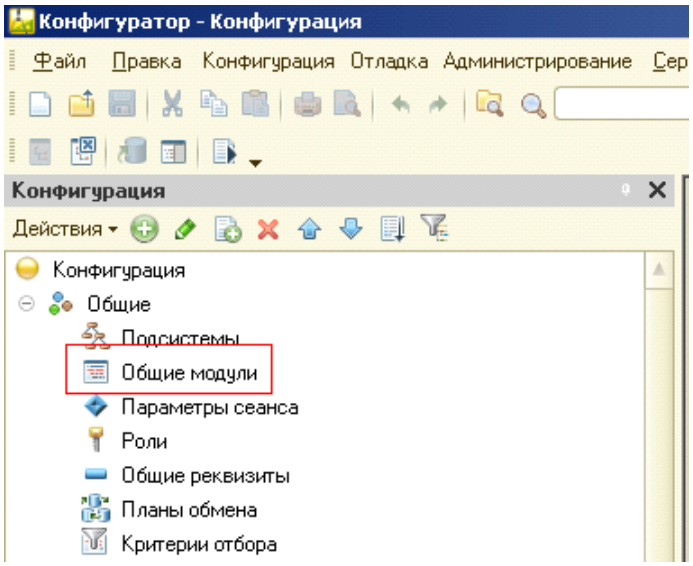


Рис. 4.2.12

В общем модуле Вы можете описать процедуры и функции, которые будут доступны из любого места конфигурации. В принципе, это их основная функция, т.е. если Вам необходимо проверять остатки товаров на складе при проведении документов, которых может быть несколько видов (реализация товаров, списание товаров и т.д.), то процедуру проверки будет практично

написать именно в общем модуле, а потом вызывать в модуле объекта соответствующего документа при проведении.

Создайте общий модуль. Для этого необходимо выделить пункт *«Модули»* в ветке метаданных конфигурации «Общие», нажать правую кнопку мышки и выбрать команду *«Добавить»*.

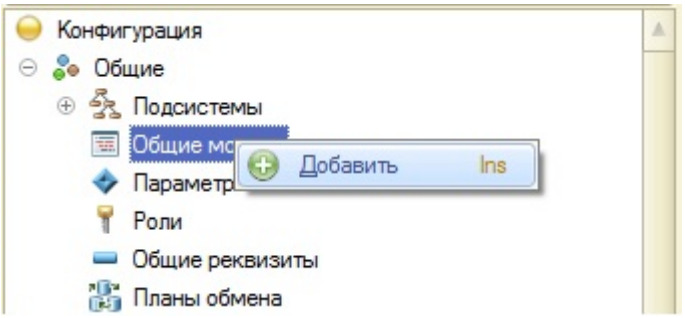


Рис. 4.2.13

Модуль будет создан и сразу же откроется окно свойств, где мы можем назвать его каким-нибудь именем (назовете его *Основной*) и установить некоторые настройки модуля.

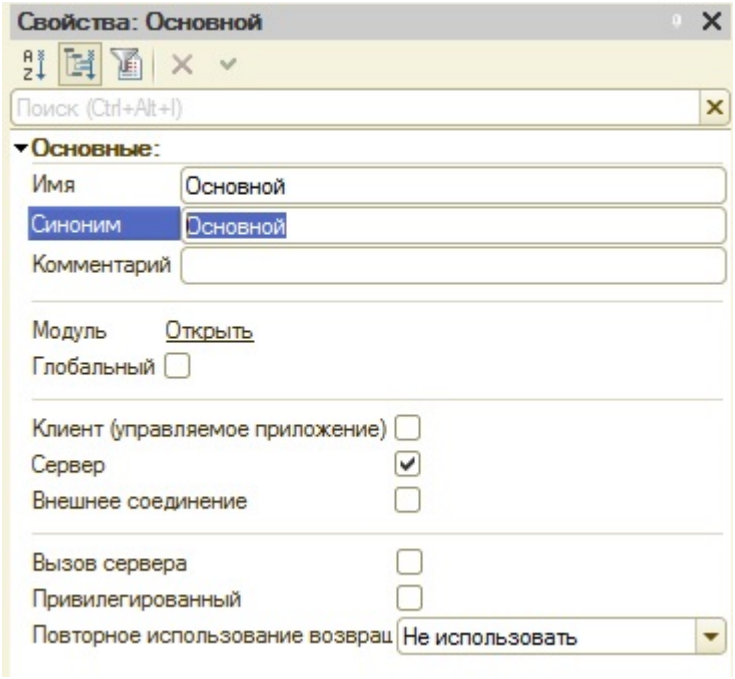


Рис. 4.2.14

Первый флажок, который нас интересует - это *Глобальный*. Это значит, что если установлен данный флажок, то процедуры и функции в этом модуле будут выполняться в рамках глобального контекста. Как мы уже узнали ранее, можно будет просто в любом месте программы написать название функции и процедуры, и она будет выполняться.

Если же этой галочки не будет, то обратиться к процедуре или функции этого модуля можно будет через сам модуль.

Т.е. написать:

```
Основной.ПроцедураРасчета();
```

Не рекомендую использовать глобальные общие модули, поскольку в этом случае требуется контроль уникальности в названиях процедур и функции во всех глобальных модулях. В ином же случае, контроль уникальности в названиях процедур и функций ведется только в рамках модуля, где эти методы описаны. Т.е. вполне возможна такая запись:

```
Основной.ПроцедураРасчета();
Дополнительный.ПроцедураРасчета();
```

К тому же, масштабное использование *глобальных* общих модулей может негативно отобразиться на производительности программы, поскольку все глобальные модули компилируются при старте приложения. В отличие от не глобальных модулей, которые компилируются в момент обращения к процедуре или функции этого модуля.

Флажок *Клиент (управляемое приложение)* устанавливается в том случае, когда процедуры или функции данного модуля должны быть доступны в клиентском контексте при работе *толстого клиента*.

Флажки *Сервер* и *Внешнее соединение* устанавливаются в том случае, когда процедуры и функции данного модуля будут доступны в серверном контексте и во внешнем соединении.

Это значит, что если Вы не поставите флажок *Внешнее соединение*, а в результате работы какое-нибудь внешнее соединение (например, обмен) будет использовать процедуру из этого модуля (к примеру, это может быть, если в результате обмена проводятся какие-нибудь документы, где при проведении используется Ваша функция), то программа выдаст ошибку.

То же самое и с серверным контекстом. Если на стороне сервера будет выполняться процедура из этого модуля, то компилятор выдаст ошибку, в случае если нет соответствующего флажка.

Подробно о серверном и клиентском контексте - в пятой главе.

Если Вы установите флаг *Вызов сервера*, то Вы сможете методы этого модуля использовать в клиентском контексте управляемой формы. Это значит, что в клиентском контексте (директива компиляции &НаКлиенте) формы (или команды) у Вас будет доступен вызов процедур и функций этого общего модуля (без разницы, под *толстым клиентом* запущена форма или под *тонким*).

Ещё раз. Если мы установили флаг *Клиент*, то в клиентском контексте управляемой формы мы сможем использовать процедуры и функции общего модуля только под *толстым клиентом*. Если управляемая форма будет работать как под толстым клиентом, так и под тонким клиентом (или веб-клиентом), то для использования на этой форме процедур и функций общего модуля у этого модуля необходимо установить флаг *Вызов сервера*.

Когда установлен флаг *Привилегированный*, при выполнении процедур и функций общего модуля будут отключены проверки прав пользователей. Данный режим удобно использовать при выполнении каких-то глобальных и масштабных алгоритмов (например, контроль остатков по партиям). Поскольку очень часто при работе глобальных алгоритмов могут использоваться объекты (справочники, регистры сведений и т.д.), на которые у пользователя нет прав, поэтому

чтобы огульно не давать права на все используемые объекты, необходимо использовать режим *Привилегированный* общего модуля.

И последний параметр общего модуля – повторное использование возвращаемых значений.

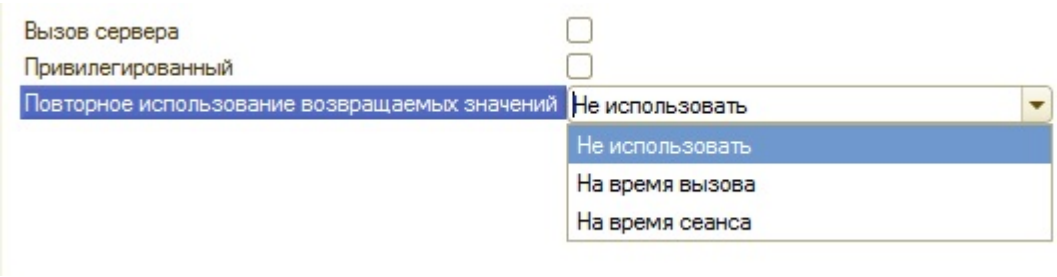


Рис. 4.2.15

Для каких целей данный параметр? Иногда бывают такие ситуации, что в процессе работы в функцию или процедуру могут передаваться несколько раз одинаковые параметры. Например, нам нужно вычислить курс какой-то валюты к рублю, мы это делаем в отдельном, для этого предназначенном модуле в какой-то функции и передаем в нее такие параметры, как дата и ссылка на валюту. Делаем запрос к базе данных и получаем какое-то значение. Если нам опять в этот же день понадобится вычислить курс этой же валюты к рублю, то мы опять передадим дату и ссылку на эту валюту, а потом снова сделаем запрос к базе данных и вычислим курс. И так каждый раз, что, естественно, скажется на производительности приложения. В целях оптимизации производительности можно кэшировать значения функций, т.е. при первой передаче параметров будет обращение к базе и вычисление функции, а при последующей передаче параметров возвращаемое значение будет браться из кэша, нет необходимости каждый раз заново вычислять значение.

Чтобы функции общего модуля кэшировались, в параметр *Повторное использование возвращаемых значений* необходимо установить или значение «На время вызова» (кэш будет использоваться, пока работает функция, которая вызвала метод общего модуля), или значение «На время сеанса» (кэш будет использоваться, пока работает пользователь).

И напоследок: у всех общих модулей есть одна особенность: Вы не сможете задать глобальные переменные этого модуля.

С общими модулями закончили.

Следующий уровень контекста - это *модуль сеанса*.

Модуль сеанса

Модуль сеанса необходим для установки параметров сеанса. В первой части этой главы мы создали единственный параметр сеанса в нашей конфигурации – «Имя текущего компьютера».

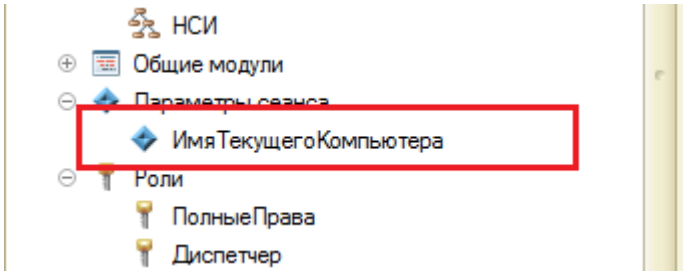


Рис. 4.2.16

Научимся устанавливать этот параметр сеанса при запуске сеанса пользователя. Для этого откроем модуль сеанса. Делается это достаточно просто: выделяется самый верхний узел конфигурации «Конфигуратор», вызывается контекстное меню, в котором необходимо кликнуть на пункт «Модуль сеанса».

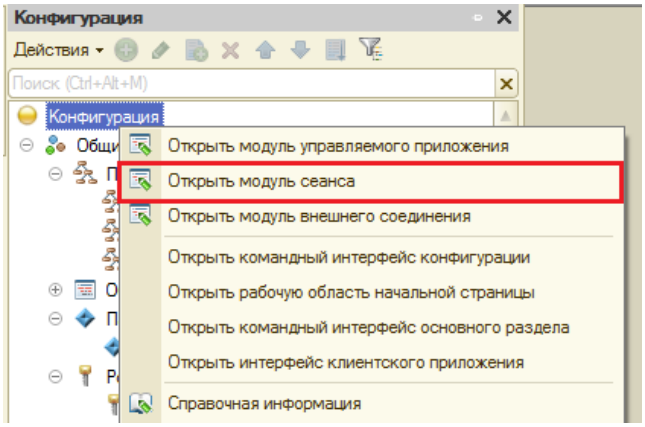


Рис. 4.2.17

Впоследствии откроется модуль сеанса, в котором нас интересует один-единственный обработчик событий «Установка параметров сеанса». Создадим процедуру – обработчик события, для этого после открытия модуля выполним команду «Процедуры и функции» меню «Текст» (модуль должен быть активен).

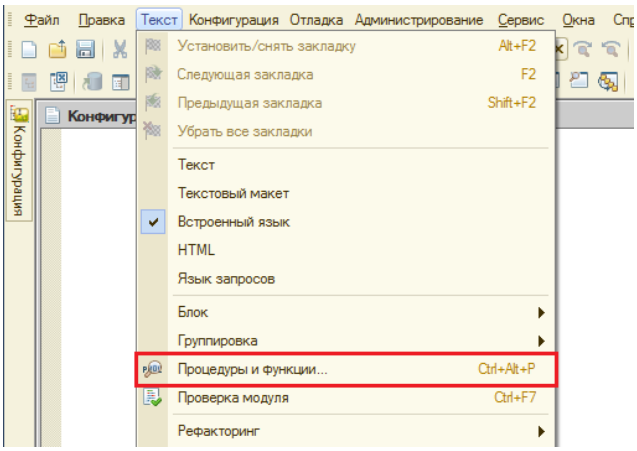


Рис. 4.2.18

В открывшемся окне выделяем единственную процедуру и нажимаем кнопку «Перейти».

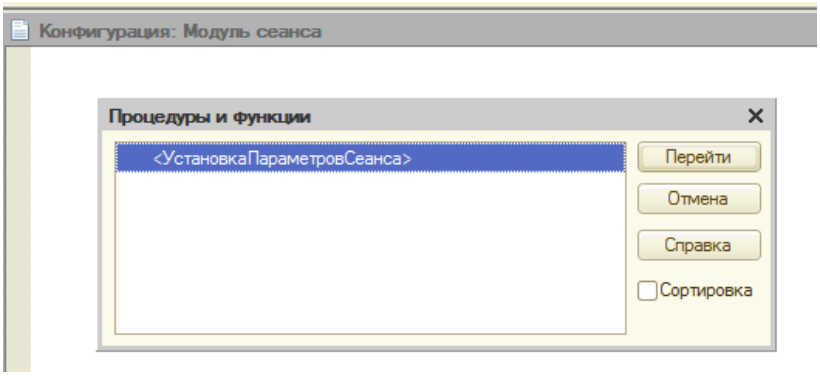


Рис. 4.2.19

Будет создана процедура «УстановкаПараметровСеанса»

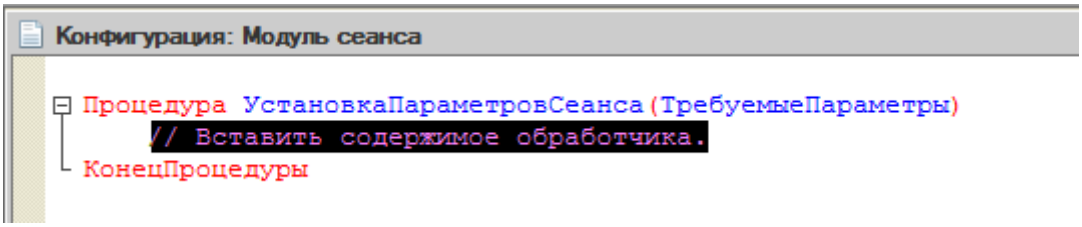


Рис. 4.2.20

Напишем в новой процедуре следующий код:

```
Процедура УстановкаПараметровСеанса(ТребуемыеПараметры)

    ПараметрыСеанса.ИмяТекущегоКомпьютера = ИмяКомпьютера();

КонецПроцедуры
```

Листинг 4.2.5

Где **ИмяКомпьютера()** – функция глобального контекста, которая возвращает сетевое имя текущего компьютера.

А **ПараметрыСеанса** – коллекция, которая содержит параметры сеанса.

Проверим, как работает установка параметров сеанса. Для этого создайте обработку, форму и команду формы «ПолучитьПараметр», а также обработчик команды на клиенте и на сервере.

В обработчиках команды напишем следующий код:

```
&НаСервере
Процедура ПолучитьПараметрНаСервере()
    Сообщить(ПараметрыСеанса.ИмяТекущегоКомпьютера);
КонецПроцедуры

&НаКлиенте
Процедура ПолучитьПараметр(Команда)
    ПолучитьПараметрНаСервере();
КонецПроцедуры
```

Листинг 4.2.6

Сохраним конфигурацию, обновим базу данных, перезапустим «1С:Предприятие» в пользовательском режиме. Запустим обработку и посмотрим, как выполнится команда формы обработки.

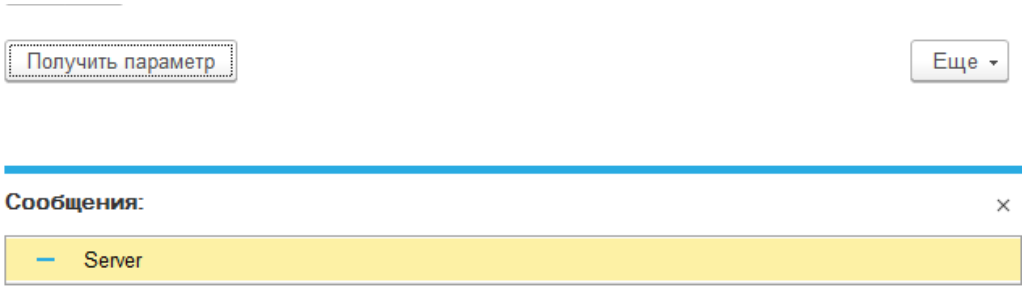


Рис. 4.2.21

Модуль приложения

Рассмотрим модули самого верхнего уровня контекста - это модуль **управляемого** приложения и модуль **обычного** приложения.

В модулях управляемого и обычного приложения располагаются, как правило, процедуры и функции, связанные с началом и окончанием работы программы.

Методы в модуле управляемого приложения срабатывают тогда, когда идет запуск в режиме управляемого приложения. А методы в модуле обычного приложения отрабатывают тогда, когда идет запуск в режиме обычного приложения. В этом курсе мы рассматриваем работу

управляемого приложения, и поэтому будем работать только с модулем управляемого приложения.

Естественно, данный модуль выполняется в рамках глобального контекста, тем самым все процедуры и функции, которые Вы объявите в модуле обычного приложения, будут доступны из любого места программы. Также они должны быть экспортными. Как попасть в модуль управляемого приложения? Есть два способа.

Первый. Необходимо выделить слово *«Конфигурация»* на самом верхнем уровне конфигурации и выбрать *«Открыть модуль управляемого приложения».*

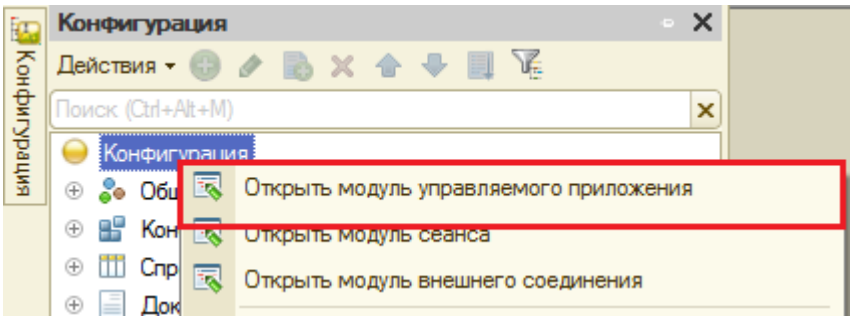


Рис. 4.2.22

Второй. Зайти в палитру свойств конфигурации и нажать на гиперссылку «Открыть» свойства «Модуль управляемого приложения».

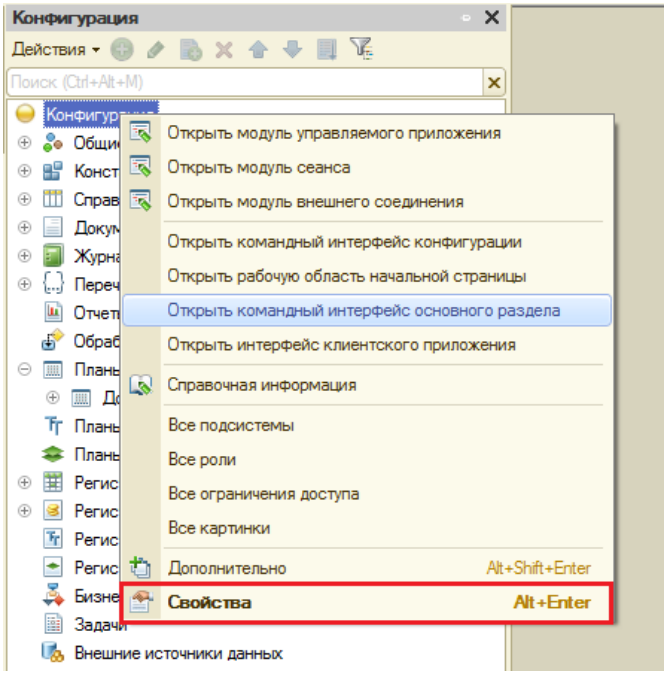


Рис. 4.2.23

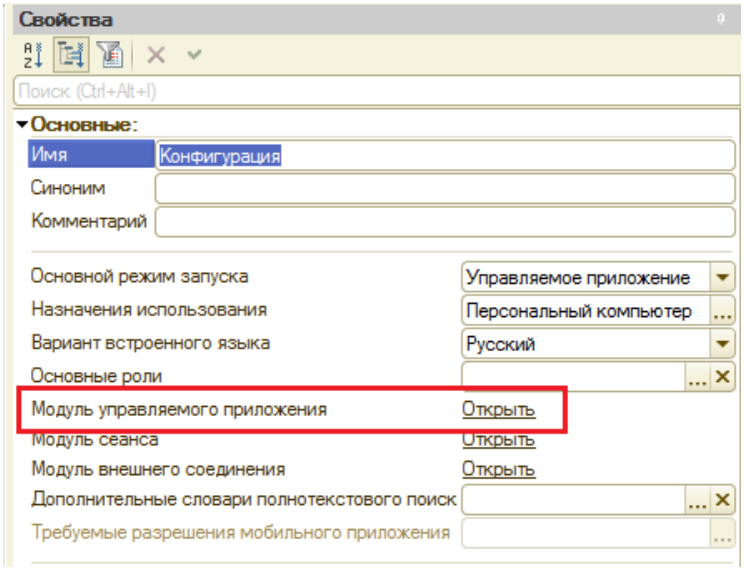


Рис. 4.2.24

Чтобы создать процедуры, которые будут выполняться при работе программы, перейдите в меню *«Текст» – «Процедуры и функции»*, и увидите список процедур.

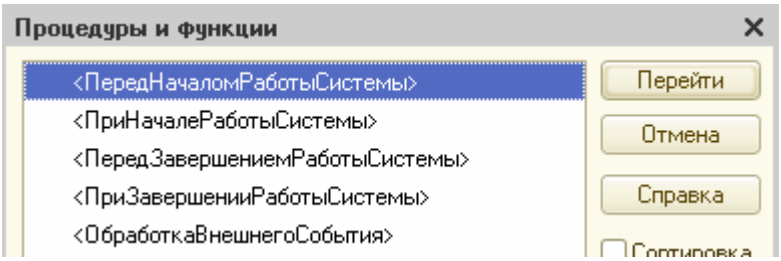


Рис. 4.2.25

Их всего пять, функции первых четырех ясны из названия, а обработка внешнего события позволяет перехватить внешнее событие при подключении оборудования, Вам пока это событие не интересно.

Создадим методы «ПередНачаломРаботыСистемы» и «ПриНачалеРаботыСистемы».

```
Процедура ПередНачаломРаботыСистемы(Отказ)
    // Вставить содержимое обработчика.
КонецПроцедуры

Процедура ПриНачалеРаботыСистемы()
    // Вставить содержимое обработчика.
КонецПроцедуры
```

Листинг 4.2.7

Вы видите, что у метода **ПередНачаломРаботыСистемы** есть параметр «Отказ», это значит, что у нас есть возможность отказаться от запуска программы. В этот метод можно ставить различные проверки (например, проверка лицензирования программы).

Метод **ПриНачалеРаботыСистемы** необходим для выполнения различных действий при запуске программы. Это может быть подключение оборудования или открытие формы рабочего стола.

Точно так же создадим методы **ПередЗавершениемРаботыСистемы** и **ПриЗавершенииРаботыСистемы:**

```
Процедура ПередЗавершениемРаботыСистемы(Отказ, ТекстПредупреждения)
    // Вставить содержимое обработчика.
КонецПроцедуры

Процедура ПриЗавершенииРаботыСистемы()
    // Вставить содержимое обработчика.
КонецПроцедуры
```

Листинг 4.2.8

Метод **ПередЗавершениемРаботыСистемы** возникает перед завершением работы управляемого приложения до закрытия главного окна. В этом методе есть параметр *Отказ*, который имеет тип *Булево*. Если данному параметру установить значение *Истина*, то работа программы не будет завершена.

Метод **ПриЗавершенииРаботыСистемы** вызывается самым последним. В этом методе необходимо выполнять действия, которые необходимы при выходе из программы: запись логов и т.п.

Экспорт процедур, функций и переменных

Следующим пунктом мы изучим экспорт процедур и функций. Рассмотрим на примере общего модуля.

Для этого создадим общий модуль, который назовем «МатематическиеВычисления».

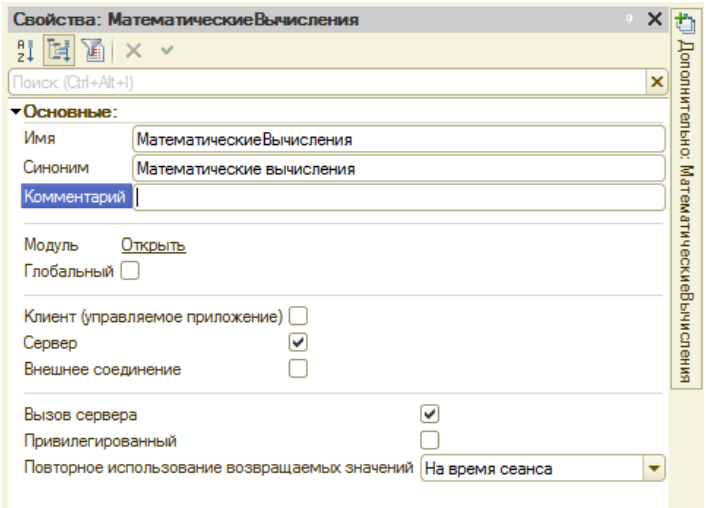


Рис. 4.2.26

Установим у этого модуля флаг «Вызов сервера», для того чтобы методы из этого модуля можно было использовать в клиентском контексте формы. А также включим кэширование (параметр «Повторное использование возвращаемых значений»).

Напишем в общем модуле функцию, которая вычисляет квадратный корень модуля какого-нибудь числа.

```
Функция КореньМодуляЧисла(ЧислоПодКорнем)

    МодульЧисла = ?(ЧислоПодКоронем >= 0, ЧислоПодКорнем,ЧислоПодКорнем*(-
1));

    Возврат Sqrt(МодульЧисла);
КонецФункции
```

Листинг 4.2.9

Естественно, возникнет вопрос: как обратиться к этой функции на какой-либо форме? Для этого необходимо сделать функцию экспортной. Осуществляется это посредством написания слова *Экспорт* после параметров, функция должна выглядеть так:

```
Функция КореньМодуляЧисла(ЧислоПодКорнем) Экспорт

    МодульЧисла = ?(ЧислоПодКоронем >= 0, ЧислоПодКорнем,ЧислоПодКорнем*(-
1));

    Возврат Sqrt(МодульЧисла);
КонецФункции
```

Листинг 4.2.10

Теперь создадим обработку, в которой будем вычислять квадратный корень модуля числа, используя функцию из общего модуля. Сделайте все как Вы уже умеете: создайте обработку, форму, команду формы (разместите её на форме), а также обработчик для этой команды. Перед этим не забудьте сохранить конфигурацию и обновить Вашу базу данных.

В обработчике команды будем получать некоторое число при помощи метода «ВвестиЧисло», вычислять корень этого числа и выводить сообщение.

```
&НаКлиенте
Процедура ВычислитьКореньЧисла(Команда)
    Перем ЧислоПодКорнем;

    Если ВвестиЧисло(ЧислоПодКорнем,"Введите число") Тогда

        КореньЧисла =
МатематическиеВычисления.КореньМодуляЧисла(ЧислоПодКорнем);

        Сообщить(Окр(КореньЧисла,2));

    КонецЕсли;
КонецПроцедуры
```

Листинг 4.2.11

Запустите обработку и посмотрите, как она будет работать. После уберите слово *Экспорт* у названия метода **КореньМодуляЧисла** в общем модуле и попробуйте заново посчитать. Компилятор выдал ошибку.

Резюмирую: если процедура или функция экспортная (имеется ключевое слово *Экспорт* после параметров), то к этой функции можно обращаться вне данного модуля. В противном случае она может быть использована только в том модуле, в котором описана.

Заметьте, что компилятор дает подсказки: если после названия модуля написать точку и нажать клавишу «пробел», то выйдет список из всех экспортируемых процедур этого модуля (см. рис. 4.2.27).

```
Если ВвестиЧисло(ЧислоПодКорнем,"Введите число") Тогда

    КореньЧисла = МатематическиеВычисления.

    Сообщить(Окр(КореньЧисла,2));

КонецЕсли;
```

Рис. 4.2.27

Экспорт переменных осуществляется точно так же, как экспорт функций и процедур: задается переменная в начале модуля и после описания переменной пишется слово *Экспорт*, после ставится точка с запятой.

Создадим в *модуле управляемого приложения* переменную «РазрядностьПоУмолчанию», в которой будем указывать число знаков дробной части при округлении. Присваивать значение этой переменной будем в процедуре **ПриНачалеРаботыСистемы** (см. листинг 4.2.12).

```
Перем РазрядностьПоУмолчанию Экспорт;

Процедура ПриНачалеРаботыСистемы()
    РазрядностьПоУмолчанию = 3;
КонецПроцедуры
```

Листинг 4.2.12

Переделаем код из листинга 4.2.11:

```
&НаКлиенте
Процедура ВычислитьКореньЧисла(Команда)
    Перем ЧислоПодКорнем;

    Если ВвестиЧисло(ЧислоПодКорнем,"Введите число") Тогда

        КореньЧисла =
МатематическиеВычисления.КореньМодуляЧисла(ЧислоПодКорнем);

        Сообщить(Окр(КореньЧисла,РазрядностьПоУмолчанию));

    КонецЕсли;
КонецПроцедуры
```

Листинг 4.2.13

И напоследок изучим еще одну особенность модулей. Помимо процедур и функций, Вы в модуле можете написать любой код, который Вам нужен. Он будет выполняться тогда, когда будет активирован этот модуль. Если это модуль формы, то когда открыта форма, если модуль объекта - то тогда, когда было обращение к этому объекту (без разницы, программное или интерактивное).

Например, в имеющейся у нас конфигурации в модуле объекта справочника автомобилей напишите следующий код:

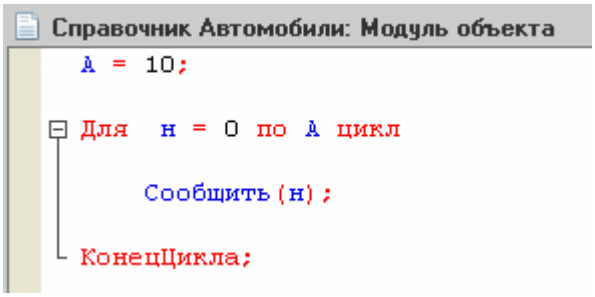


Рис. 4.2.28

Откройте теперь любой элемент справочника. Вышли сообщения, т.к. при обращении к объекту сразу идет выполнение модуля, если в нем есть код.

Но есть исключения. Общие модули, модуль сеанса и модуль менеджера служат только для описания процедур и функций.

И, по правилам языка, все процедуры и функции должны идти перед основным кодом модуля.

Вот и всё. Данная часть была больше информационного характера, чтобы Вы имели ясное представление, как пишется конфигурация 1С. В этой книге мы не будем использовать общие модули и модуль обычного приложения, а будем работать только с модулем объекта и модулем формы.

В следующей главе мы научимся работать с формами, наконец-то узнаем, что такое клиентский и серверный контекст, и научимся размещать элементы на формах.

Глава 5. Работа с управляемыми формами

В этой главе мы научимся работать с управляемыми формами. Информация в этой главе дается в таком объеме, чтобы Вы смогли начать работать с формами: создавать формы, размещать элементы и программировать на форме. Более подробно работа с управляемыми формами в частности и с управляемым приложением вообще разбирается в моей книге «Основы разработки в 1С: Такси. Программирование управляемого приложения за 12 шагов».

Часть 1. Конструирование форм

Создание форм - это самый базовый процесс, с которым столкнется любой, кто будет вести разработку под управляемым приложением.

Вам уже приходилось создавать различные формы (как правило, формы обработок) в процессе изучения этой книги, но я особо не вдавался в подробности этих действий. В этой главе мы изучим вопросы создания форм более основательно. Но перед тем, как начать упражняться с управляемыми формами, узнаем, что такое основные формы объектов.

Основные формы объектов

У всех объектов есть определенный перечень основных форм. Причем количество и состав основных форм может отличаться в зависимости от прототипа объекта. Например, у справочников это – форма элемента, форма списка, форма группы, форма выбора и форма выбора группы (причем, форма группы и форма выбора группы будут доступны только *иерархическим* справочникам!). А у документов – форма документа, форма списка и форма выбора. По сути, для каждого типа формы должна быть одна основная форма. Но в то же время форм одного типа может быть несколько. Например, можно создать несколько форм списков справочника *Автомобили* и выбрать одну из этих форм основной.

Для чего необходимо устанавливать основные формы объекта? Основная форма всегда будет отображаться пользователю при выполнении стандартных команд (команды, которые определяют основные действия с объектом (подробнее см. «Основы разработки в 1С: Такси», часть 2, глава 2). Говоря простым языком: основная форма будет всегда открываться по умолчанию из интерфейса «1С:Предприятия». Например, основная форма списка справочника *Автомобили* отобразится, когда будет выполнена стандартная команда навигации, которая открывает форму списка номенклатуры (см. рис. 5.1.1).

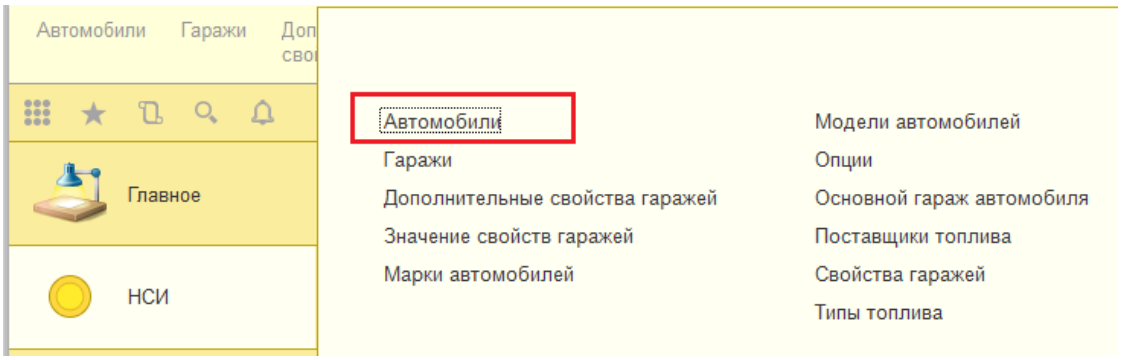


Рис. 5.1.1

Таким образом, если мы создали несколько различных форм одного типа, к примеру, формы списков, но хотим, чтобы по умолчанию при выполнении стандартной команды открывалась только одна конкретная, то её и нужно сделать основной.

Если основная форма у данного объекта не задана, то при выполнении стандартных команд будет открываться **автоматически** сгенерированная форма. Такое действие платформы в некоторых случаях разработчика может устраивать (например, если мы создали простой справочник). Но очень часто необходимо создать собственную основную форму.

Узнаем, где в конфигурации можно посмотреть на перечень основных форм объекта. Сделать это можно несколькими способами. Мы разберем один из них.

Самый простой способ - это с помощью редактора объекта метаданных (в основном это касается объектов таких метаданных, как справочники, документы, регистр, планы видов характеристик и т.д.). Открыть его можно дважды кликнув по объекту мышкой. В этом редакторе нас интересует закладка формы (см. рис. 5.1.2), где в верхней части обычно и расположены поля с основными формами этого объекта.

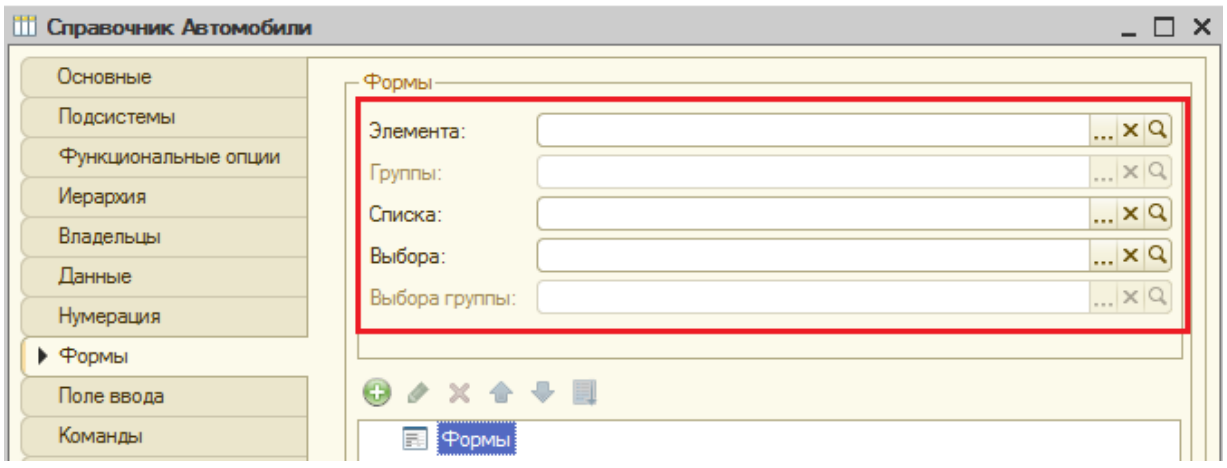


Рис. 5.1.2

На рис. 5.1.2 все поля основных форм пустые, потому что для справочника «Автомобили» не была создана ни одна форма.

Я не буду в этой книге глубоко разбирать, какие типы форм бывают и чем они отличаются друг от друга. По сути, названия всех типов форм интуитивно понятны, и их предназначение без труда ищется в справочной информации.

Создание новых форм

Что такое основные формы, мы узнали. Теперь научимся создавать различные типы форм. Начнем мы с форм справочников и документов. Все формы создаются с помощью конструктора форм. Для каждого прототипа объекта (документы, справочники и т.д.) есть свой конструктор форм. Для документов – конструктор форм документов, для справочников – конструктор форм справочников и т.д. Вызвать этот конструктор можно несколькими способами (см. рис. 5.1.3, 5.1.4, 5.1.5), но в любом случае вызывать его нужно из того объекта, для которого Вы создаете форму.

Рассмотрим три способа вызова конструктора:

1) В дереве конфигурации выделить узел «Формы» ветки нужного объекта, вызвать контекстное меню и нажать кнопку «Добавить» (см. рис. 5.1.3);
2) В дереве конфигурации выделить нужный объект, вызвать контекстное меню, найти подменю «Добавить» и в этом подменю нажать на кнопку «Форма» (см. рис. 5.1.4);
3) В редакторе объекта, на закладке «Формы», в дереве форм нажать кнопку «Добавить» (см. рис. 5.1.5).

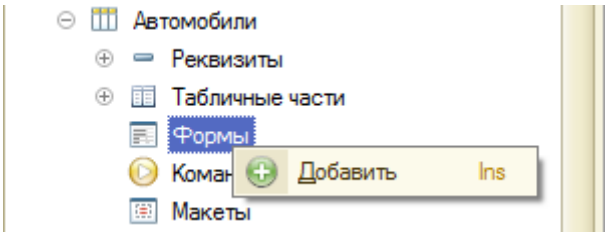


Рис. 5.1.3

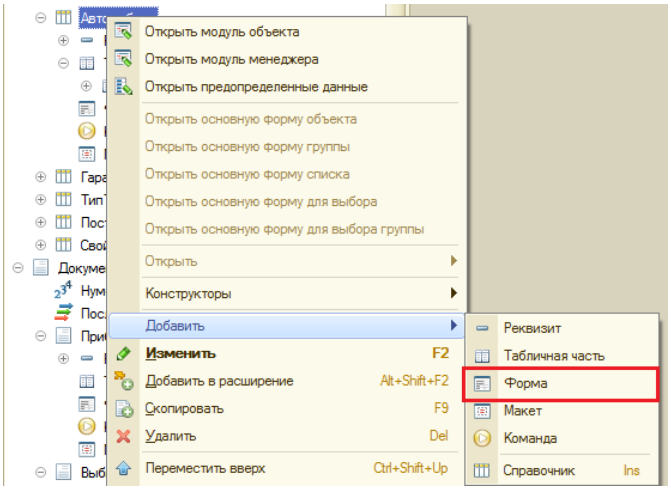


Рис. 5.1.4

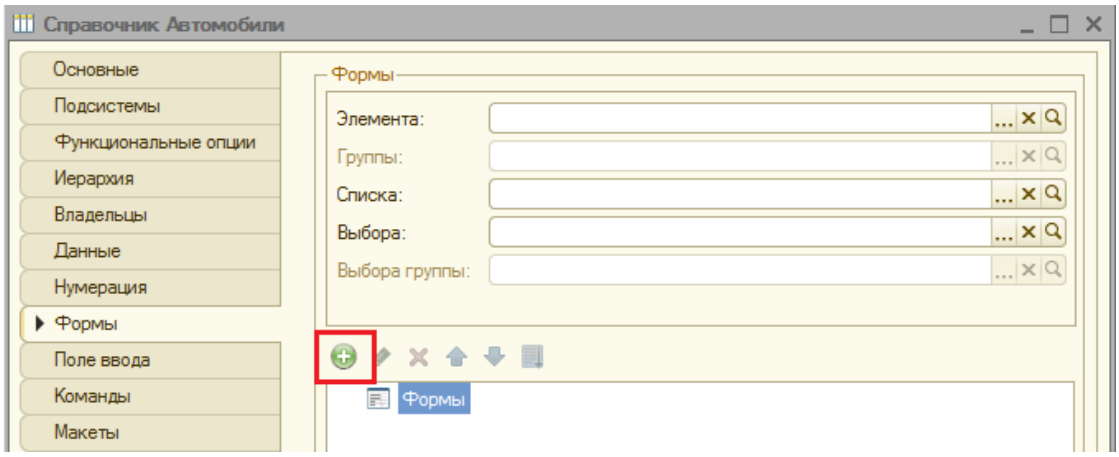


Рис. 5.1.5

После вызова откроется конструктор формы. Для разных типов объектов конструктор может быть разной формы. Например, на рис. 5.1.6 представлен конструктор формы для справочника, а на рис. 5.1.7 – для документа.

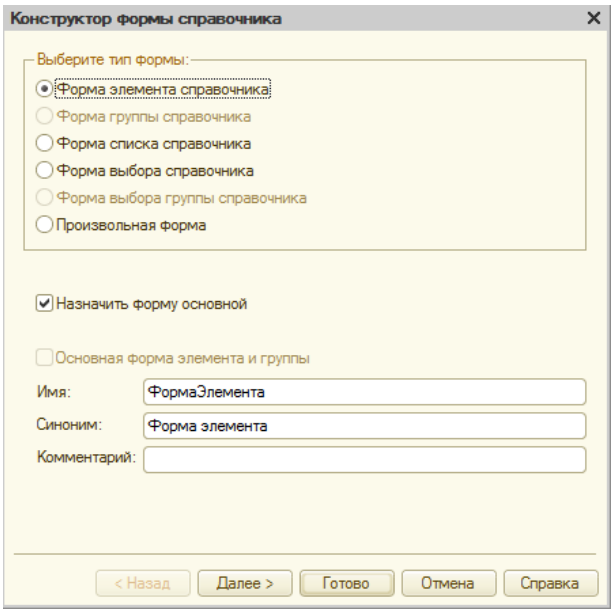


Рис. 5.1.6

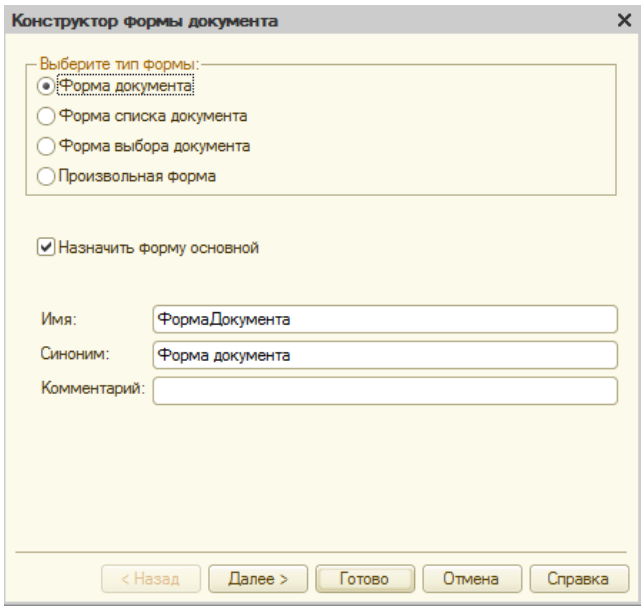


Рис. 5.1.7

Разберем первую страницу конструктора. Вверху формы нам предложено выбрать тип формы, которая будет создана. Как уже понятно, в зависимости от прототипа объекта (и даже вида прототипа) список типов форм может быть разный. Для справочников это один перечень форм, а для документов другой. Причем обратите внимание на рис. 5.1.5: иногда некоторые типы форм могут быть не доступны. Например, в случае справочника *Автомобили* недоступны следующие типы форм: форма группы справочника и форма выбора группы справочника, из-за того, что у этого справочника отключена иерархия. Если мы попытаемся создать форму для справочника «Гаражи» (у которого иерархия включена), то эти типы будут доступны.

С помощью свойства «Назначить форму основной» мы можем установить, что вновь созданная форма станет основной. Если у нас уже есть какая-то основная форма, то она никуда не

денется, просто перезапишется основная форма объекта того типа, форму для которого мы создаем.

Имя – это индикатор формы. В рамках одного объекта не может быть двух форм с одинаковым именем.

Синоним - это свойство, определяет, как форма будет отображаться в пользовательском интерфейсе по умолчанию (может быть несколько форм с одинаковыми синонимами).

Раньше мы всегда на первой странице конструктора останавливались и нажимали кнопку «Готово». В этот раз мы продолжим изучать конструктор новых форм. Создадим форму документа *УстановкаЦенНаТопливо*. Тип формы выберем – «Форма документа», назначим будущую форму основной, остальное оставим по умолчанию и нажмем на кнопку «Далее» (см. рис. 5.1.8).

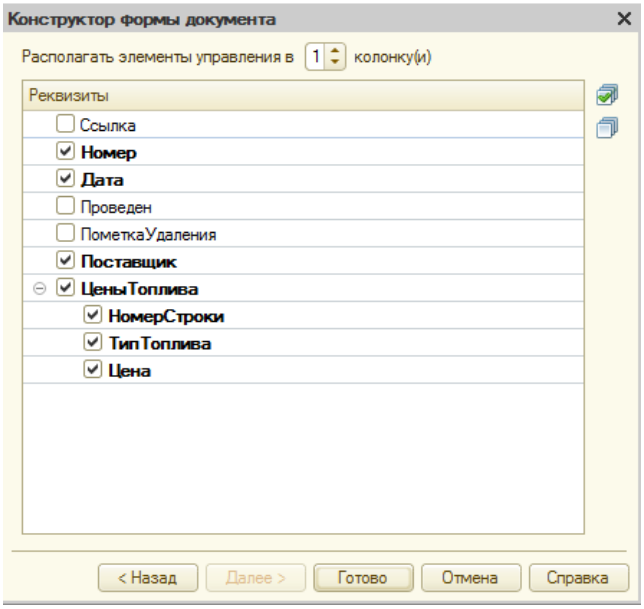


Рис. 5.1.8

В следующем окне конструктора формы мы увидим все реквизиты, табличные части и реквизиты табличных частей объекта. Нам предлагается с помощью флажков отметить, какие реквизиты попадут на нашу форму, а какие нет. Если мы создаем форму документа, форму элемента справочника, форму обработки и т.д., то в списке реквизитов будут присутствовать и реквизиты шапки объекта, и табличные части объекта с реквизитами этих частей. А если мы создаем формы списка или выбора, то будут присутствовать только реквизиты шапки.

После выбора нужных реквизитов необходимо нажать на кнопку «Готово», и форма будет создана (см. рис. 5.1.9).

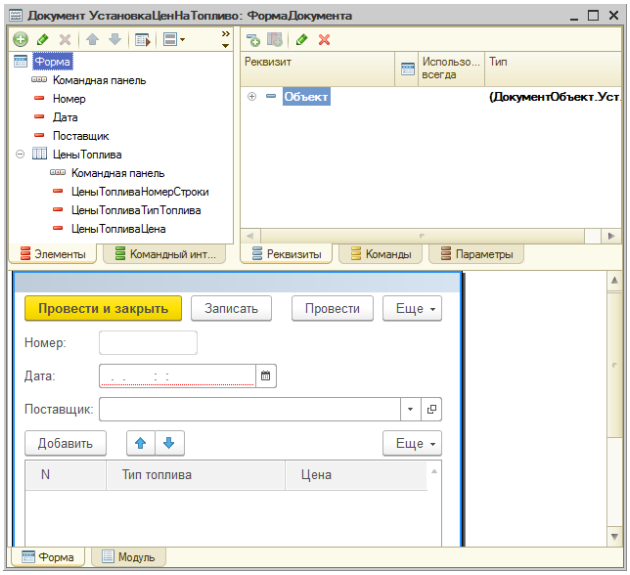


Рис. 5.1.9

Обратите внимание: после того, как мы создали форму элемента, в свойство «Основная форма документа» записалась наша вновь созданная форма (см. рис. 5.1.10).

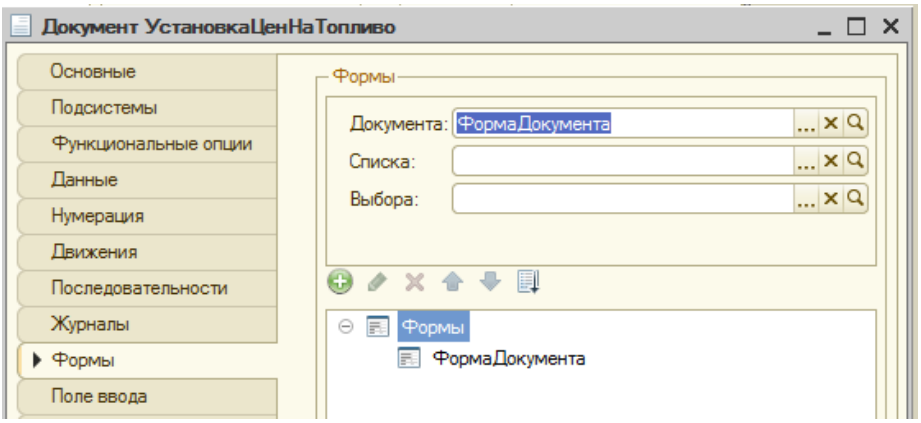


Рис. 5.1.10

Вы можете создавать какое угодно количество форм справочников и документов разных типов. Платформа также предоставляет возможность создания форм копированием. Для этого нужно выделить нужную форму, вызвать контекстное меню и в этом контекстном меню выбрать пункт «Копировать» (см. рис. 5.1.11). После этого будет создана новая форма, аналог скопированной, но под другим именем (см. рис. 5.1.12). В этом случае редактор создания форм открываться не будет. И новая форма не станет основной. Если разработчик желает сделать новую форму основной, то ему необходимо перевыбрать форму в соответствующем поле закладки формы редактора объекта (см. рис. 5.1.13).

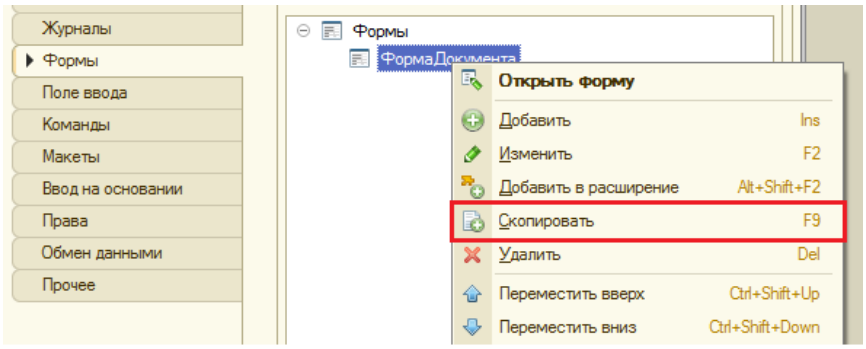


Рис. 5.1.11

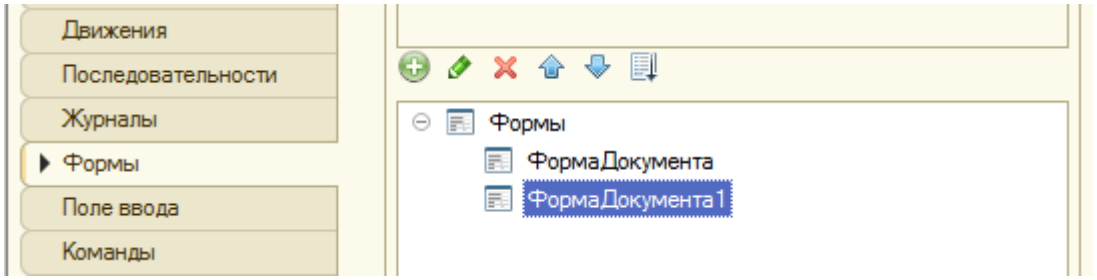


Рис. 5.1.12

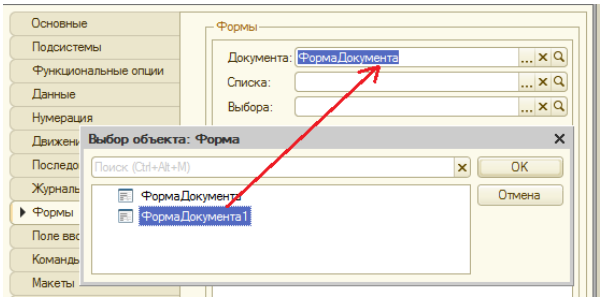


Рис. 5.1.13

С созданием форм все. Создайте самостоятельно форму справочника «Автомобили». А после приступайте ко второй части этой главы, где мы будем изучать командный интерфейс формы.

Часть 2. Командный интерфейс формы

В предыдущей части мы научились создавать формы объектов и узнали, что такое основная форма, как её задать и поменять. В этой части мы познакомимся с командным интерфейсом формы. Командный интерфейс формы разберем в самых общих чертах, чтобы иметь правильное представление об этом.

Но перед тем как начать изучать командный интерфейс форм, узнаем в общем виде, что такое команды и какие виды команд бывают.

Более подробно вопросы работы с командами в целом и командами формы разбираются во 2-й части книги «Основы разработки в 1С: Такси».

Что такое команда? Команда в понятиях платформы 1С это объект, с помощью которого система выполняет некоторые действия.

Все команды можно разделить на две большие группы: глобальные команды и локальные команды формы. Глобальные команды предназначены для работы с приложением в целом. А локальные команды формы действуют только в контексте конкретной формы. Например, команда открытия формы списка справочника или документа – это глобальная команда, её можно выполнять практически из любого места конфигурации. А команда, которая обрабатывает нажатие на кнопку «Закрыть» формы документа, это локальная команда. Она действует только в контексте определенной формы.

Все команды можно разделить на два вида, это навигационные команды и команды действия.

Навигационные команды - это, как правило, команды, необходимые для перемещения по приложению. Это команды открытия форм списков документов, справочников и журналов документов.

Команды действия – побуждают пользователя совершить какое-то новое действие. Создать новый документ или справочник, открыть форму отчета или запустить обработку.

Также все глобальные команды можно разделить на независимые и параметризируемые.

Независимые глобальные команды – это команды, которые выполняются для приложения в целом. Например, открытие формы списка справочника.

Параметризируемые глобальные команды – это команды, для выполнения которых нужен какой-то определенный параметр (или несколько). Без этого параметра они выполняться не могут. Например, открытие формы регистра накопления с отбором по регистратору (параметр в этом случае – регистратор, какой-то документ).

После того, как Вы создали форму документа, справочника или любого другого объекта, на рабочем столе разработчика откроется редактор формы. На рис. 5.2.1 Вы видите редактор управляемой формы документа *УстановкаЦенНаТопливо* нашей конфигурации.

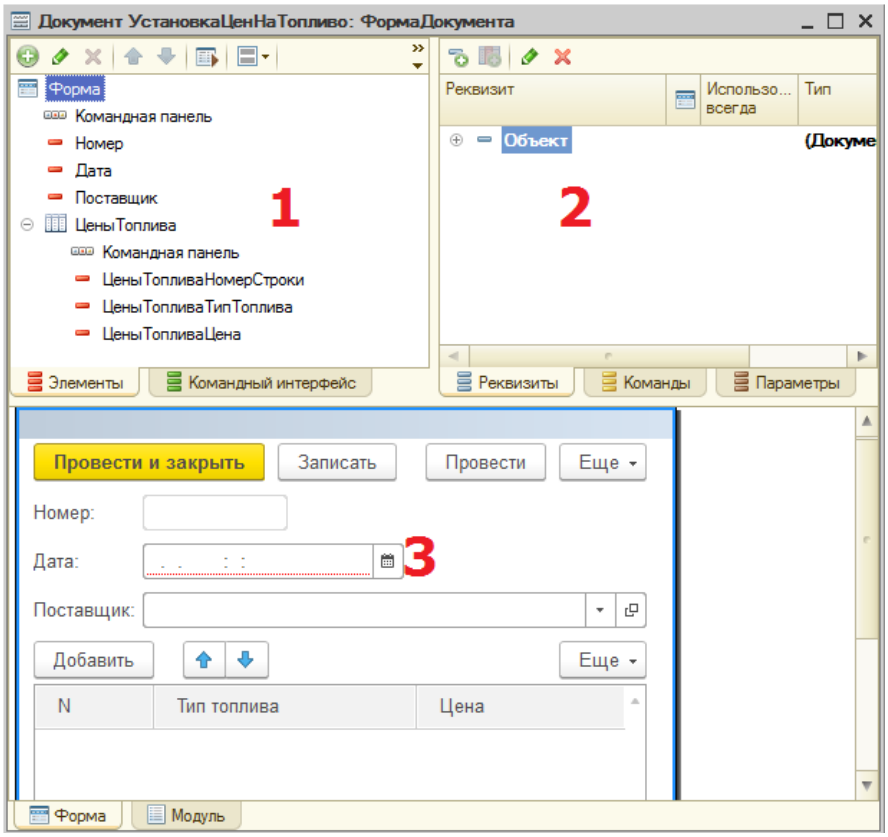


Рис. 5.2.1

Редактор управляемой формы разбит на три окна (см. рис. 5.2.1).

- **Окно 1** разбито на две закладки: «Элементы» и «Командный интерфейс»
  - «Элементы» - все элементы формы в том порядке, в каком они расположены на форме.
  - «Командный интерфейс» - команды, которые могут выполняться на форме.
- **Окно 2** разбито на три закладки: «Реквизиты», «Команды» и «Параметры».
  - «Реквизиты» - содержит реквизиты формы (что это, мы узнаем позже);
  - «Команды» - состав команд, которые потенциально можно применить на форме (**локальные и глобальные** команды).
  - «Параметры» - содержит параметры формы (что это, мы узнаем позже)
- **Окно 3** разбито на две закладки - «Форма» и «Модуль»;
  - «Форма» - внешний вид формы, как его увидит пользователь;
  - «Модуль» - модуль формы для описания событий на встроенном языке.

Определим основные термины:

- Реквизиты формы – это данные, с которыми осуществляется основная работа формы. Более подробно работать с реквизитами мы будем в следующей части, а пока отметим, что есть основной реквизит. На рис. 5.2.1 в окне 2 это реквизит «Объект» и выделен жирным. При создании типизированных форм (форма документа, элемента справочника, списка и т.п.) основной реквизит создается автоматически.
- Команды формы – с помощью них пользователь осуществляет действия на форме. В окне 2 в закладке «Команды» перечислены все возможные команды, которые пользователь может

использовать в форме. В окне 1 в закладке «Командный интерфейс» перечислены все команды, размещенные на форме.

- Элементы формы – реквизиты, которые размещены на форме.
- Параметры формы – это данные, с помощью которых осуществляется открытие формы (об этом подробно во второй части шестой главы).

В этой части мы будем изучать командный интерфейс формы, поэтому активно будем работать с первым окном (обеими закладками) и закладкой «Команды» второго окна.

Посмотрим, что собой представляет командный интерфейс формы (см. рис. 5.2.2). Он содержит два узла, это «Панель навигации» и «Командная панель».

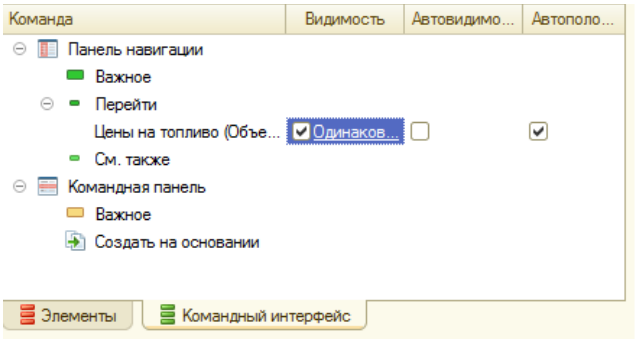


Рис. 5.2.2

Посмотрим, что из себя представляют эти объекты на форме в пользовательском виде (см. рис. 5.2.3).

Для того, чтобы появилась панель навигации на форме, необходимо установить флаг «Видимость» для ветки «Цены на топливо» узла «Панель навигации» (см. рис. 5.2.2).

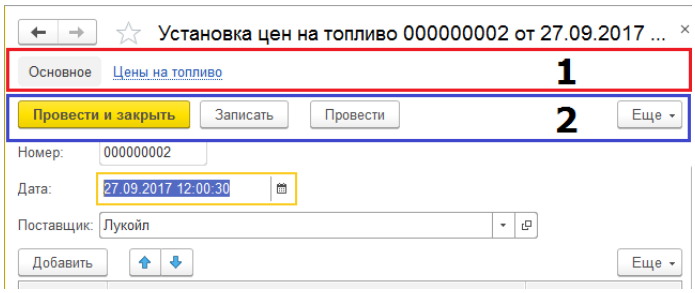


Рис. 5.2.3. Панель навигации и командная панель на форме (1 – панель навигации, 2 - командная панель)

В закладке «Командный интерфейс» формы помимо колонки, где расположены узлы панель навигации и командная панель, есть еще три колонки - «Видимость», «Автовидимость» и «Автоположение». Разберем, для чего они нужны.

При установке флажка в колонке «Видимость» команда, для которой поставлен этот флаг, будет однозначно всегда показываться на форме (если у пользователя есть соответствующее право).

Следующая колонка – «Автовидимость». Данная колонка доступна только для команд, добавляемых автоматически (о них ниже). При установке флажка в эту колонку автоматически снимается или устанавливается флаг в колонке «Видимость». Например, для команд открытия форм списков с отбором по регистратору всех регистров (сведений, накопления, бухгалтерии и расчетов) по умолчанию видимость отключена, поэтому когда Вы поставите флаг «Автовидимость» рядом с такой командой, то флаг «Видимость» этой команды снимется (как сняты флажки на рис. 5.2.4). Если разработчику необходимо, чтобы пользователь видел эти формы, то необходимо вручную поставить для них всех видимость, как это сделано на рис. 5.2.2.

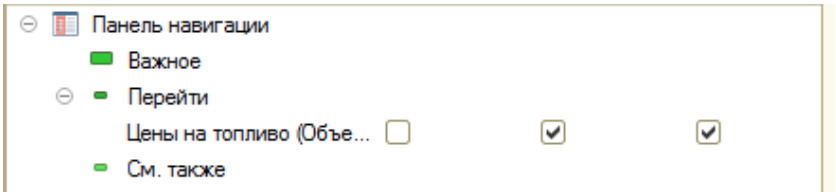


Рис. 5.2.4

Устанавливает ли признак «Автовидимость» флаг «Видимость» в значение *Истина*, или нет - определяется разработчиками платформы. В этой книге нет смысла перечислять все возможные варианты зависимостей. Просто при создании формы проверяйте установку флажка «Видимость», и если по умолчанию он снят, а Вам нужно чтобы у пользователя был доступ к этой команде, то просто установите этот флажок.

И последняя колонка «Автоположение» - определяет расположение команды по умолчанию. При снятии флажка с этой колонки команды можно перемещать внутри группы (сразу становятся доступны стрелки на командной панели вверху окна).

Разберемся с панелью навигации и командной панелью. Для этого самостоятельно создайте форму документа для документа *Прибытие в гараж* (если не сделали это ранее).

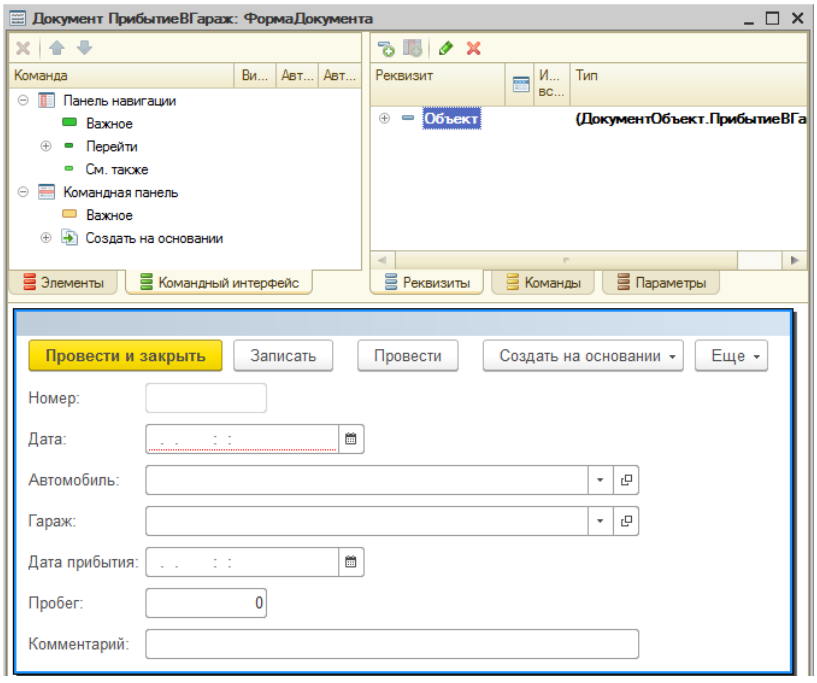


Рис. 5.2.5

Панель навигации служит для размещения навигационных команд (команды, необходимые для навигации по конфигурации), это какие-либо **глобальные** команды (независимые или параметризируемые). А командная панель предназначена для размещения команд формы и стандартных команд формы (закрытие, запись и т.д.).

Панель навигации формы состоит из трех групп, это группы «Важное», «Перейти» и «См. также». Разработчик может самостоятельно перемещать команды между группами с помощью мышки: необходимо выделить команду, удерживать клавишу мышки и тащить её куда нужно. Особых затруднений эта операция у Вас не должна вызвать, все точно так же, как и в командном интерфейсе подсистем. Для интерфейса «Такси» нет особой разницы, в какой группе размещена команда.

В панели навигации формы документа *Прибытие в гараж* автоматически разместилась команда на открытие формы списка регистра накопления *Пробег автомобиля* с выключенной видимостью (см. рис. 5.2.6).

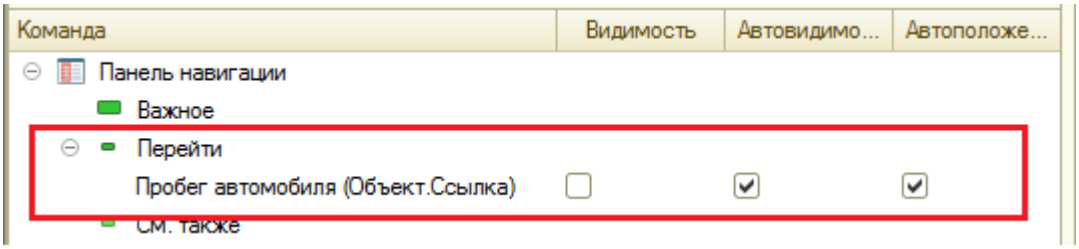


Рис. 5.2.6

Установим флаг *Видимость* (флаг *Автовидимость* автоматически снимется), сохраним конфигурацию, обновим базу данных и посмотрим на панель навигации формы документа *Прибытие в гараж:*

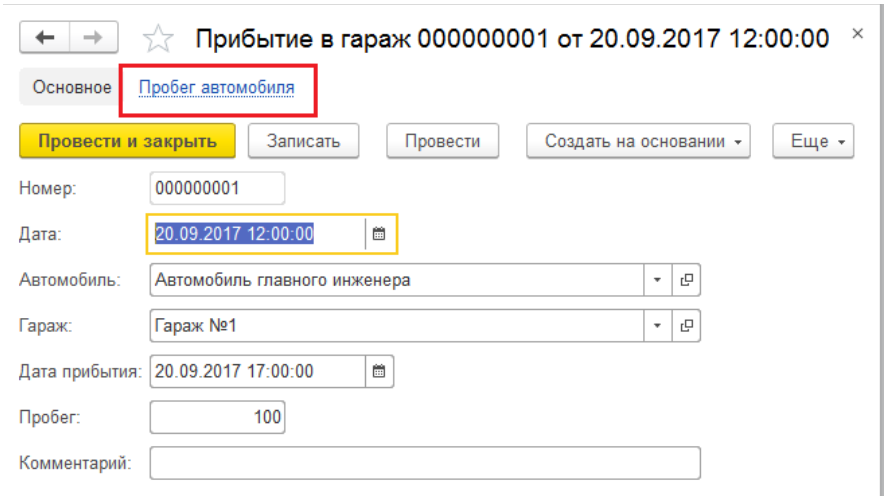


Рис. 5.2.7

Как Вы видите, в панели навигации формы документа *Прибытие в гараж* появилась ссылка, по которой можно увидеть движения этого документа по регистру накопления *Пробег автомобиля*.

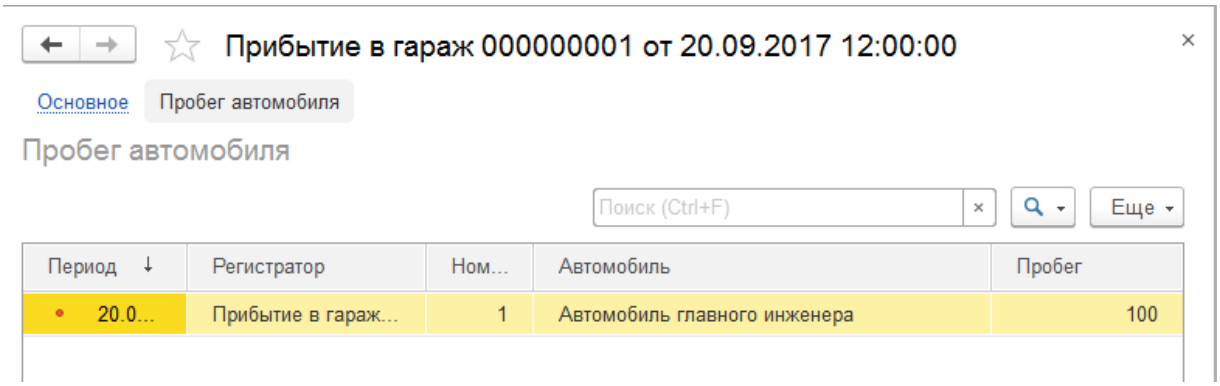


Рис. 5.2.8

Если в *панели навигации* мы можем размещать только команды навигации (глобальные), то в *командной панели* (см. рис. 5.2.9) формы мы можем размещать исключительно команды **действия** (создания на основании, печать отчетов и т.п.). Это могут быть как глобальные команды, так и локальные.

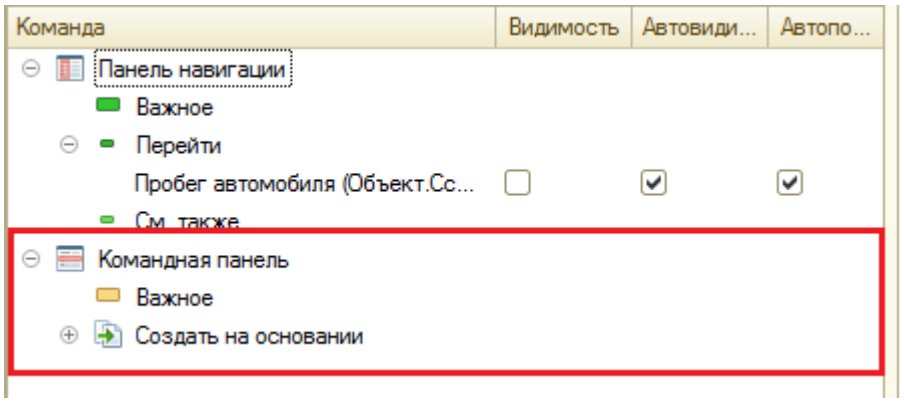


Рис. 5.2.9

В предыдущей главе мы создали возможность ввода документа *Прибытие в гараж* на основании документа *Выбытие из гаража* (см. стр. 182). Поэтому в командную панель формы автоматически должна добавиться глобальная команда создания на основании (см. рис. 5.2.10).

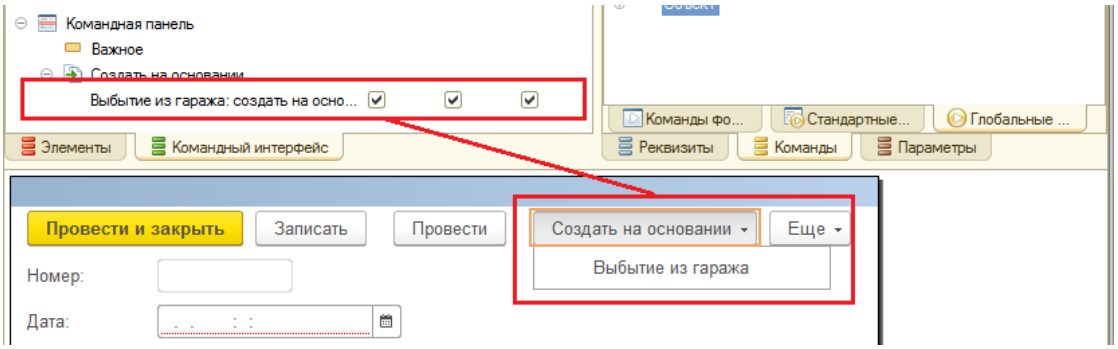


Рис. 5.2.10

Обратите внимание: флаг «Видимость» для команды создания на основании устанавливается автоматически.

Подробно о принципах формирования панели навигации и командной панели см. 2-ю часть книги «Основы разработки в 1С: Такси».

Глобальные команды, добавляемые автоматически

Как было уже сказано выше, в панель навигации добавляются **глобальные** команды. А в командную панель добавляются как глобальные, так и локальные команды. Причем Вы уже должны были обратить внимание, что некоторые команды добавляются автоматически (правда, у многих по умолчанию выключена видимость). Автоматически всегда добавляются глобальные **параметризируемые** команды, у которых в качестве параметра выступает ссылка на объект формы (реквизит «Объект»). В практике наиболее часто автоматически глобальные команды добавляются при создании форм элементов справочников и форм документов.

Например, создадим форму элемента справочника *МаркиАвтомобилей*. Для этого выделим узел «Формы» ветки «МаркиАвтомобилей», вызовем контекстное меню (см. рис. 5.2.11) и кликнем на единственный пункт «Добавить».

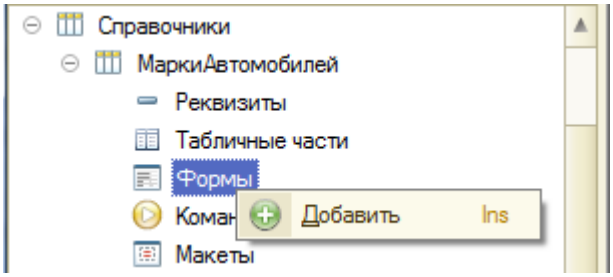


Рис. 5.2.11

В открывшемся конструкторе оставим переключатель на «Форма элементов справочника», а флаг «Назначить форму основной» оставим включенным.

Больше нам ничего от этого конструктора не нужно. Нажмем кнопку «Готово».

Во вновь созданной форме сразу перейдем на закладку «Командный интерфейс» и увидим, что в панели навигации, в группе «Перейти» уже есть команда, для которой, кстати, видимость по умолчанию включена (см. рис. 5.2.12). Разберемся, почему эта команда попала в панель навигации.

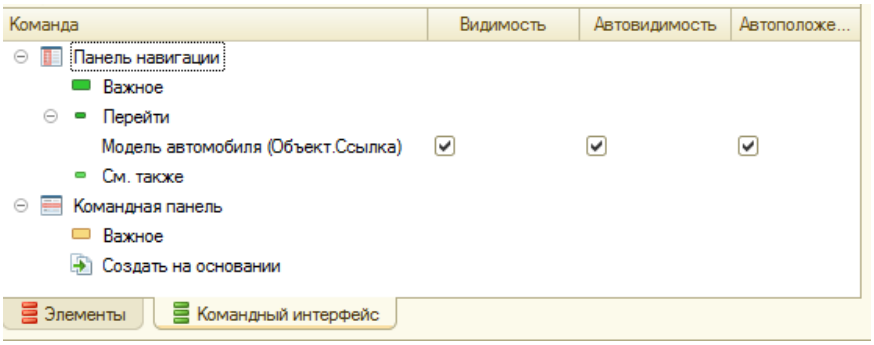


Рис. 5.2.12

В панель навигации автоматически добавилась параметризируемая команда на открытие формы списка справочника *МоделиАвтомобилей* с отбором по владельцу (если Вы помните, то справочник *МоделиАвтомобилей* подчинен справочнику *МаркиАвтомобилей*).

Команда на открытие формы списка справочника с отбором по владельцу является *стандартной глобальной параметризированной* командой, параметром этой команды является ссылка на элемент справочника. В нашем случае параметром команды на открытие формы списка справочника *МоделиАвтомобилей* является выражение «Объект.Ссылка». Реквизитом «Объект» формы элемента является объект элемента справочника *МаркиАвтомобилей* (см. рис. 5.2.13).

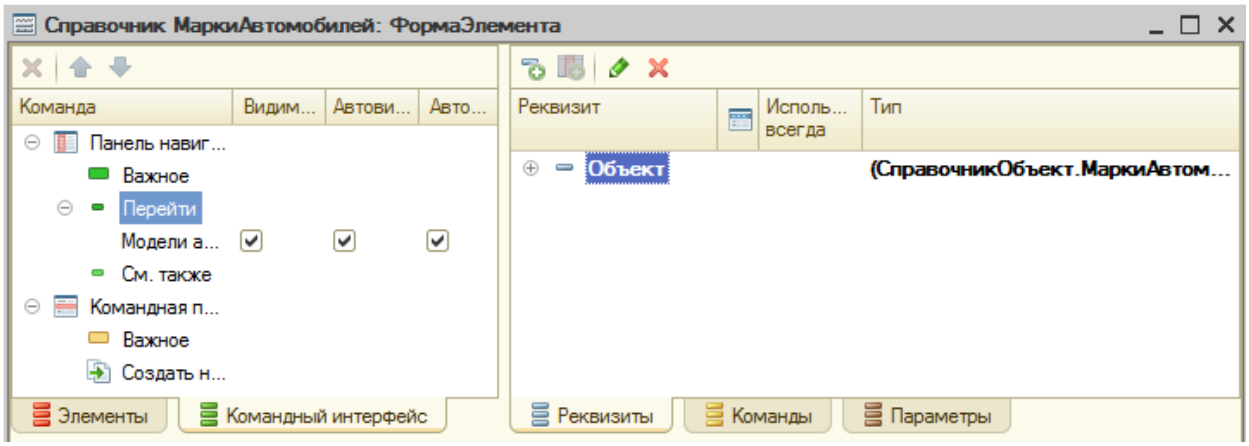


Рис. 5.2.13

Объекты и ссылки на объекты будем изучать в следующей главе.

Почему же добавилась команда на открытие формы списка с отбором по регистратору регистра накопления *ПробегАвтомобиля* на форму элемента документа *ПрибытиеВГараж*? Потому что документ *ПрибытиеВГараж* является регистратором вышеупомянутого регистра и поэтому существует стандартная параметризируемая глобальная команда на открытие формы списка регистра накопления с отбором по регистратору, которая и добавилась в командный интерфейс формы документа (с выключенной по умолчанию видимостью). Создайте самостоятельно формы документов *ЗаправкаТоплива* и *ОтчетОРасходеТоплива*, и в командном интерфейсе формы Вы обнаружите глобальную параметризируемую команду на открытие формы списка регистра накопления *ТопливоВАвтомобилях* с отбором по регистратору (см. рис. 5.2.14). Регистратор - это ссылка на документ, форма которого будет открыта.

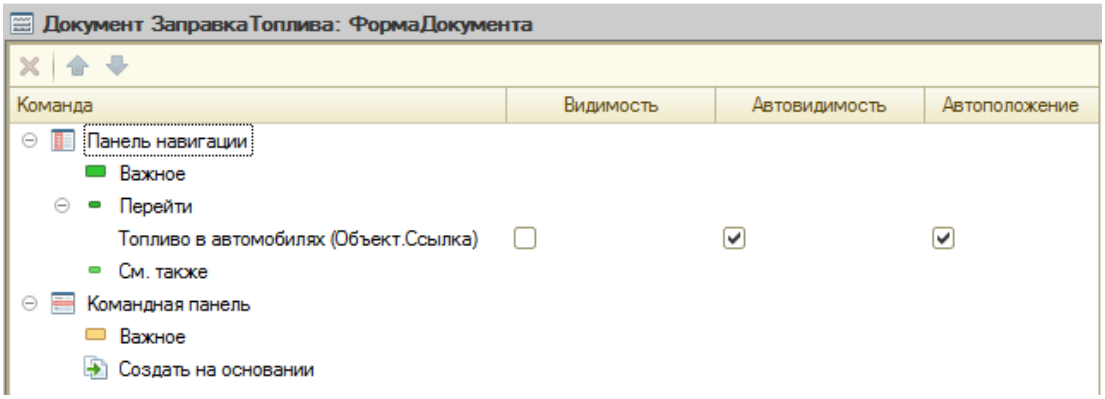


Рис. 5.2.14

Глобальные команды, добавленные вручную

Помимо возможности автоматического добавления глобальных команд, в панель навигации можно добавлять команды вручную, перенося их мышкой из вкладки «Глобальные команды» (см. рис. 5.2.15).

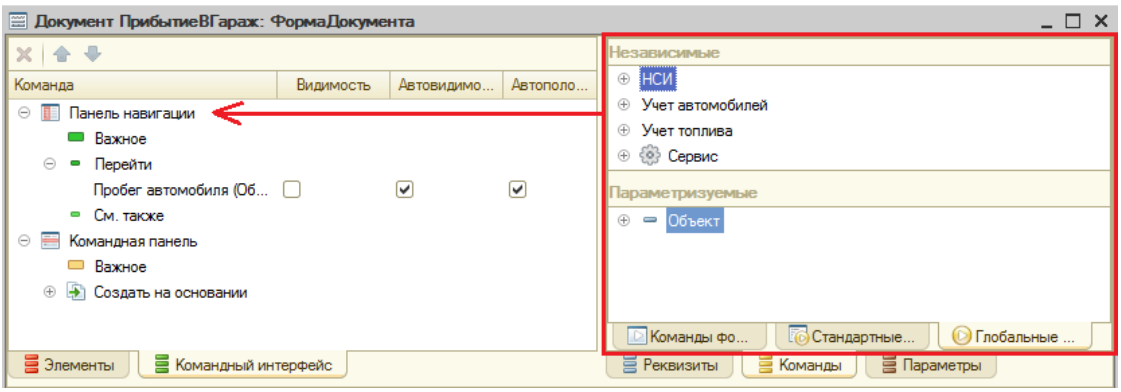


Рис. 5.2.15

В панель навигации можно добавлять только команды навигации, а в панель действия соответственно команды действия. Конструктор просто не даст Вам возможность сделать не правильно.

В этой книге я не буду подробно разбирать вопросы ручного добавления глобальных команд, более подробно эти моменты освещены во второй главе второй части книги «Основы разработки в 1С: Такси».

Локальные команды формы

Помимо глобальных команд, которые можно размещать в панели навигации (команды **навигации**) и в командной панели (команды **действия),** на форме можно размещать *локальные* команды. Локальные команды формы необходимы для выполнения различных действий на форме. Все локальнее команды расположены на закладке «Команды» в подзакладках *Команды формы* и *Стандартные команды*.

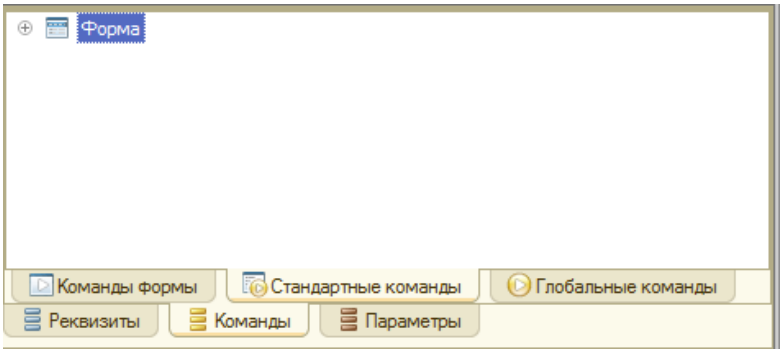


Рис. 5.2.16

В закладке *Стандартные команды* (см. рис. 5.2.17) перечислены команды формы, которые характеризуют стандартные события с формой или объектом формы. Например, это может быть команда закрытия формы, получения справки, проведения документа и т.п.

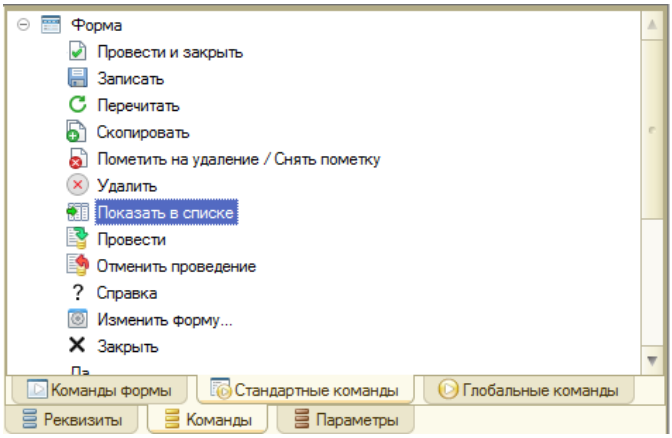


Рис. 5.2.17

В закладке *Команды формы* перечислены команды, созданные разработчиком самостоятельно. Разработчик может создать собственную локальную команду формы. Для этого необходимо на закладке *Команды формы* нажать на кнопку «Добавить» (+), после этого вновь созданная команда отобразится в списке и справа появится палитра свойств новой команды (см. рис. 5.2.18).

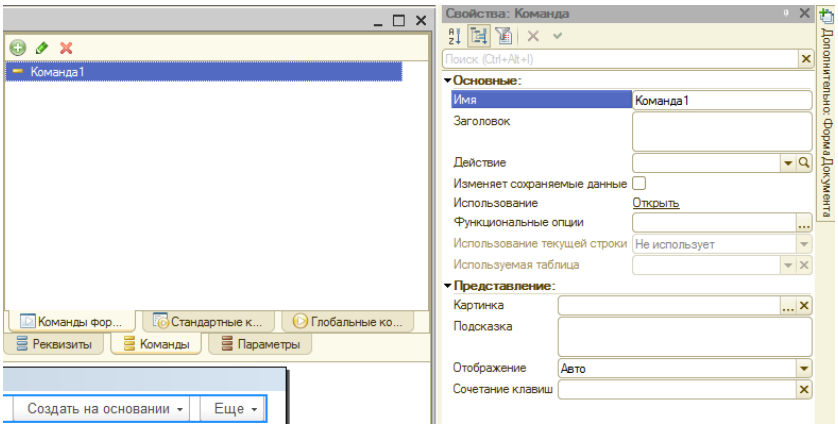


Рис. 5.2.18

В этой палитре свойств пользователь может изменить имя команды (имена должны быть уникальны в пределах формы), установить картинку, заголовок, но самое главное - это привязать какое-то действие к команде.

Назовем нашу команду «ВывестиСообщение».

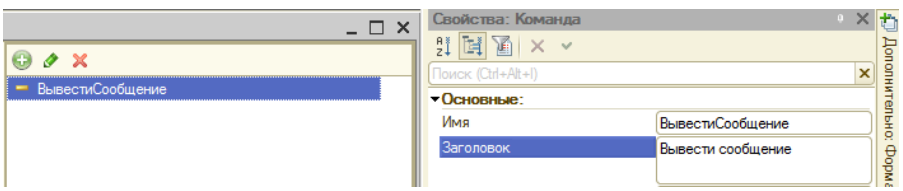


Рис. 5.2.19

В свойстве **Действие** указывается процедура, которая должна сработать, когда пользователь выполнит команду. Можно выбрать имеющуюся процедуру, а можно создать новую, нажав на кнопку «Лупа» свойства «Действие» (см. рис. 5.2.20).

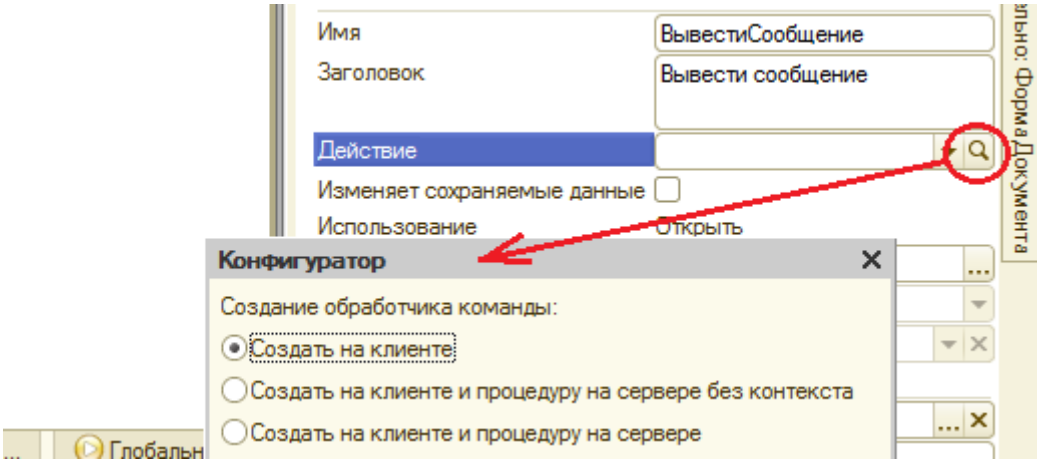


Рис. 5.2.20

После нажатия на кнопку «Лупа» будет предложено три варианта создания обработчиков команды: на клиенте, на клиенте и на сервере, на клиенте и на сервере без контекста. В четвертой части этой главы мы узнаем, в чем особенность каждого вида обработчика. Пока оставим переключатель в значение «Создать на клиенте» и нажмем кнопку «ОК», после этого в модуле формы будет создана процедура (см. рис. 5.2.21), а в свойстве *Действие* нашей команды появится название этой процедуры (см. рис. 5.2.22).

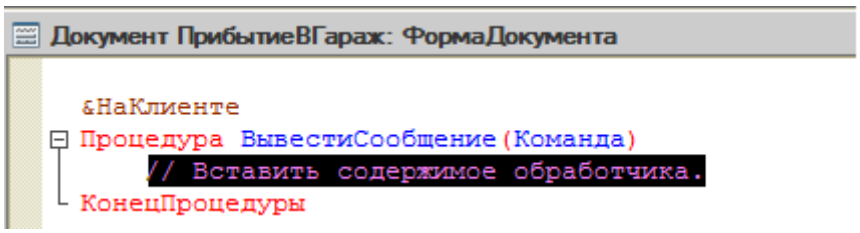


Рис. 5.2.21

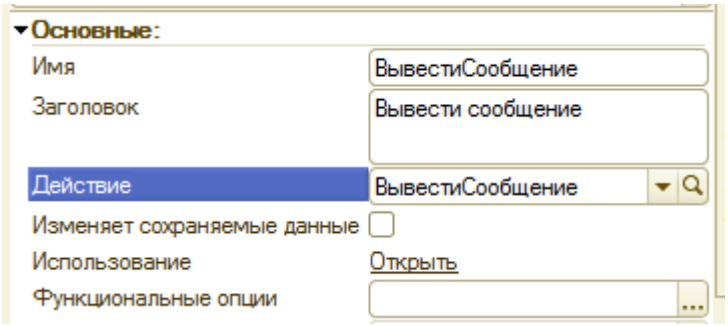


Рис. 5.2.22

Причем название процедуры будет совпадать с названием команды, поэтому не используйте в качестве названия команды какие-либо ключевые слова.

В новой процедуре самостоятельно напишите код, который будет выводить какое-либо сообщение. Теперь осталось разместить нашу команду на форме. Команду формы можно разместить как в командной панели, так и в элементах формы. Достаточно просто перетащить её в нужное место мышкой (см. рис. 5.2.23, 5.2.24). Естественно Вы не сможете разместить локальную команду в панели навигации.

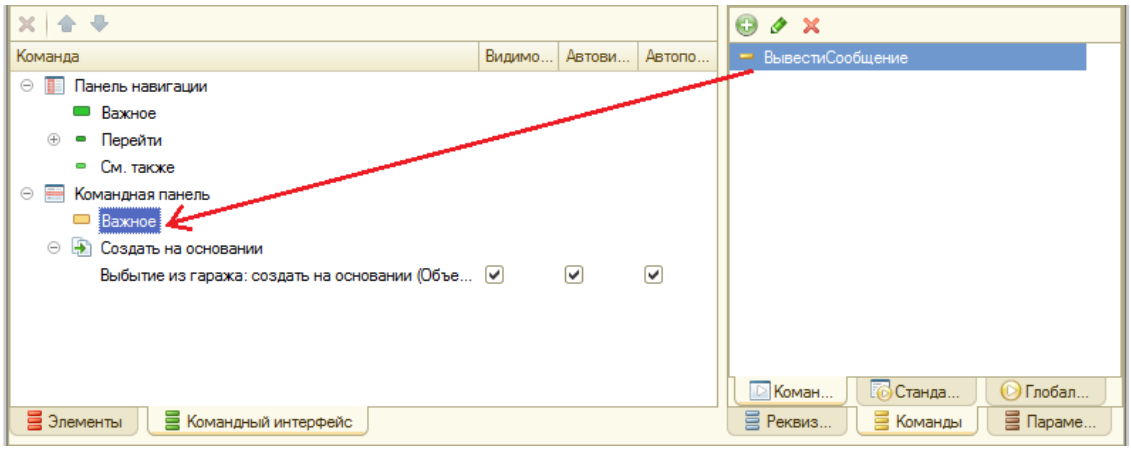


Рис. 5.2.23

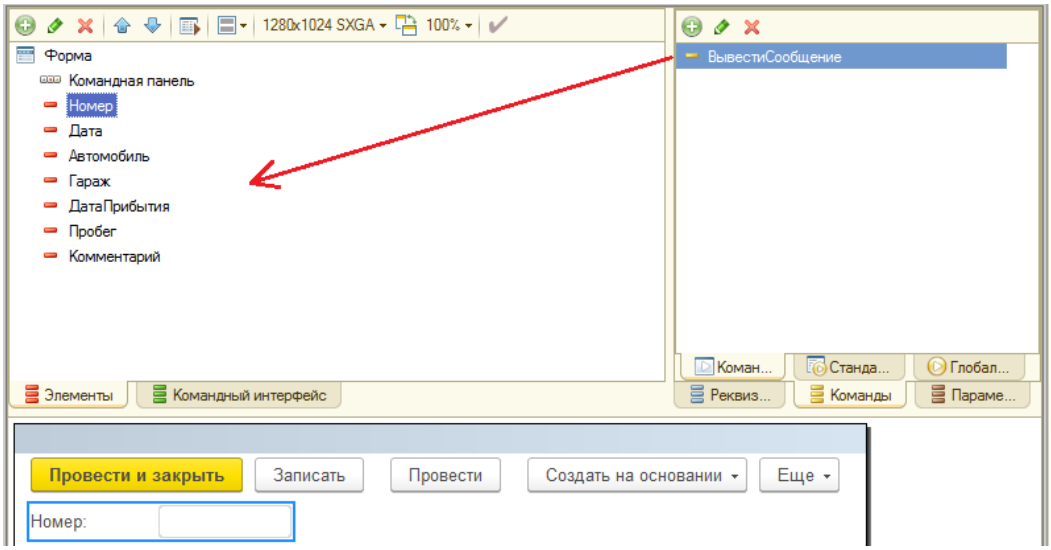


Рис. 5.2.24

После размещения кнопка, соответствующая команде, появится или на командной панели (см. рис. 5.2.25), или среди элементов формы (см. рис. 5.2.26).

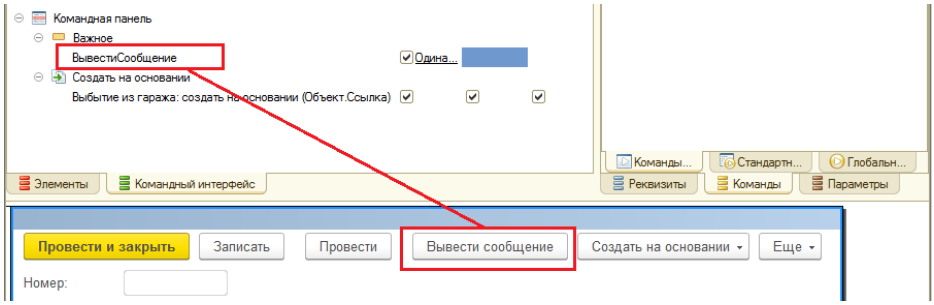


Рис. 5.2.25

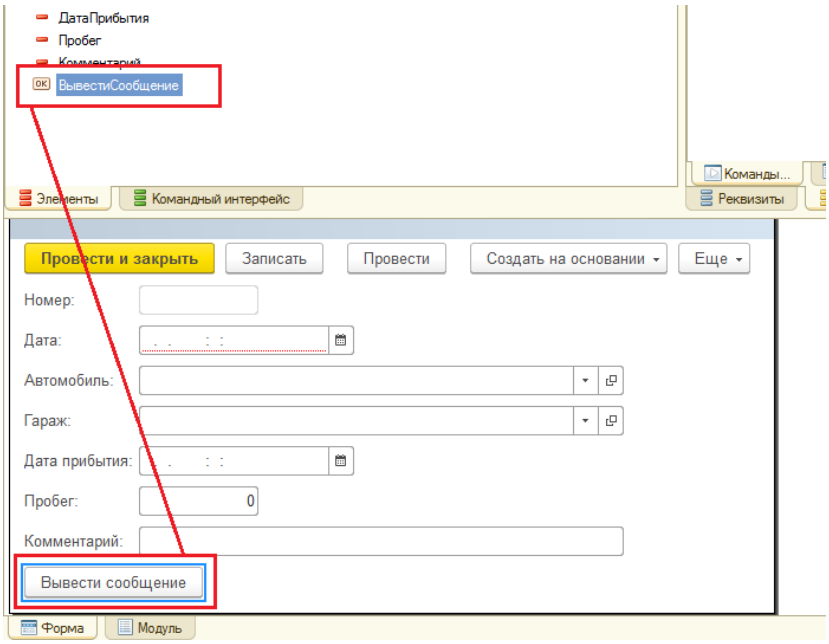


Рис. 5.2.26

При нажатии на эту кнопку выполнится код, который описан в процедуре, указанной в свойстве *Действие* связанной с этой кнопкой команды.

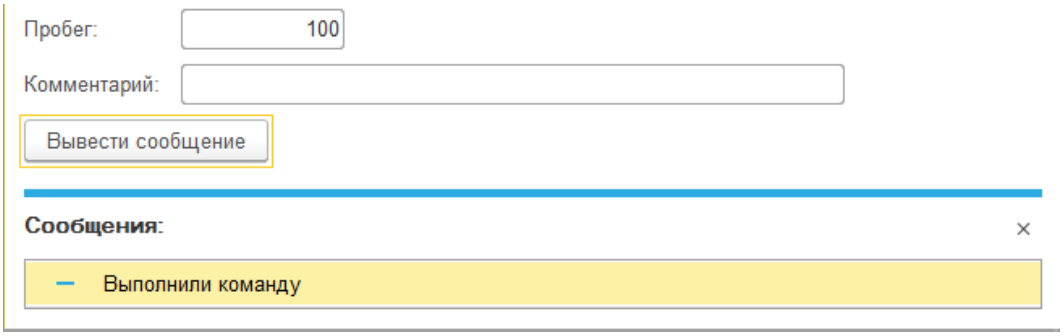


Рис. 5.2.27

Часть 3. Реквизиты и элементы формы

Реквизиты формы

Приступим к изучению реквизитов форм. Что такое реквизиты формы? Это данные, которые привязаны именно к форме: они создаются, хранятся, отображаются (по возможности) и редактируются на форме. Вне контекста формы эти данные не существуют. Сам по себе реквизит не может отображаться на форме. Для этого служат элементы формы, которые связаны с реквизитами форм (о них позже).

Очень часто разработчикам бывает необходимо разместить на форме какой-нибудь элемент. Это может быть какое-то стороннее поле ввода, или какая-то вспомогательная таблица значений. Эти элементы не должны храниться в базе данных и играют второстепенную вспомогательную роль (например, с их помощью выполняются какие-либо вычисления).

Поэтому в управляемом приложении (как и в обычном, кстати) есть возможность создать реквизит формы, который, по сути, будет **локальной** переменной формы. Реквизит формы можно разместить на форме, тогда добавится новый элемент формы, который будет привязан к этому реквизиту, а можно не размещать. Так или иначе, к реквизиту формы можно обращаться в любой процедуре или функции, написанной в модуле формы.

Это краткая теория. Теперь приступим к практической части.

В предыдущей части мы уже создали форму документа *УстановкаЦенНаТопливо*. Откройте заново эту форму и перейдите на закладку «Реквизиты» (см. рис. 5.3.1).

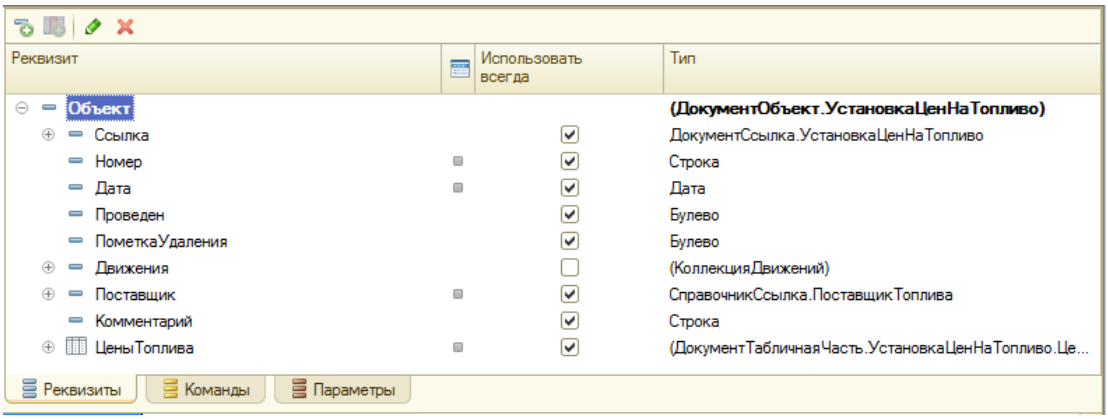


Рис. 5.3.1

С ней мы и будем работать. Сейчас у нас на форме только один реквизит, который называется «Объект», причем, как Вы могли заметить из рис. 5.3.1, он имеет иерархическую структуру.

Обратите внимание: этот реквизит выделен жирным шрифтом, и если Вы зайдете в палитру свойств этого реквизита, то свойство «Основной реквизит» будет установлено. Т.е. реквизит «Объект» нашей формы является основным реквизитом формы. У формы может быть только один основной реквизит, и с его помощью определяется функциональность формы.

Создадим несколько разных реквизитов для основной формы документа *УстановкаЦенНаТопливо*. Для того чтобы создать новый реквизит, необходимо нажать на кнопку «Добавить реквизит» закладки реквизитов (см. рис. 5.3.2).

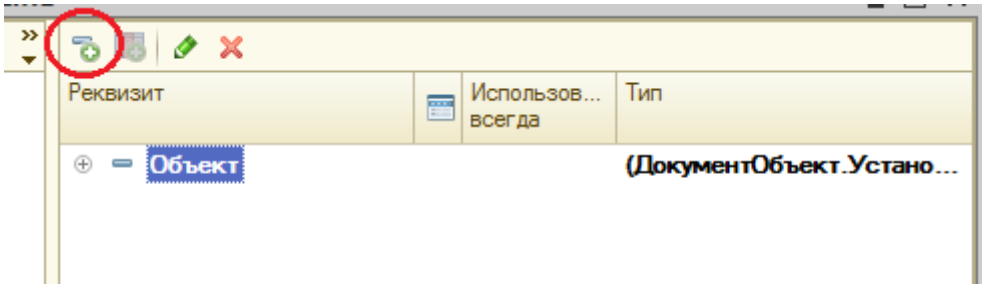


Рис. 5.3.2

После того, как Вы нажмете на эту кнопку, автоматически будет создан новый реквизит (по умолчанию тип нового реквизита - строка) и в правой части рабочей области конфигуратора откроется палитра свойств нового реквизита (см. рис. 5.3.3).

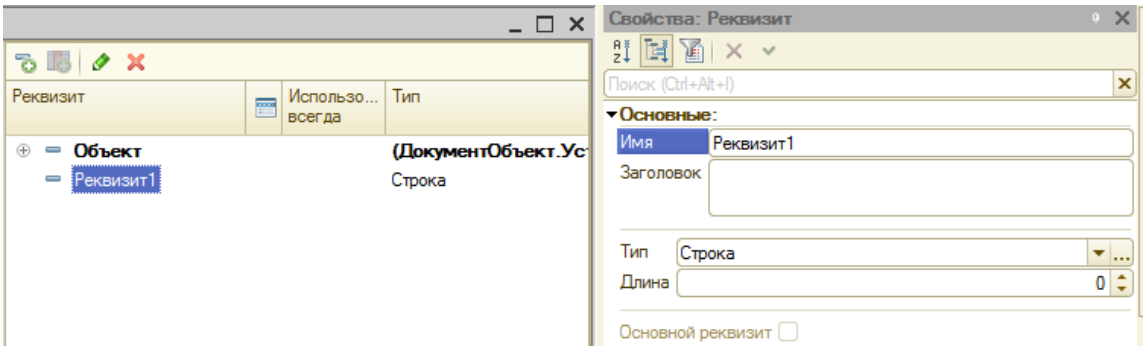


Рис. 5.3.3

Разберем основные свойства реквизита формы. И первое свойство, которое бросается в глаза, - это тип. Разработчик может назначить реквизиту практически любой из предоставленных платформой типов: примитивный, ссылочный, объект, таблицу значений, дерево значений и т.д. Причем тип также может быть составным. После нажатия на кнопку «…» рядом с типом откроется окно редактирования типов данных (см. рис. 5.3.4), посредством которого разработчик может назначить тип реквизиту.

Очень осторожно стоит использовать типы, содержащие прикладные объекты (ДокументОбъект, СправочникОбъект и т.п.). Начинающим программистам я рекомендую воздержаться от их использования (кроме случаев, когда данный реквизит является основным).

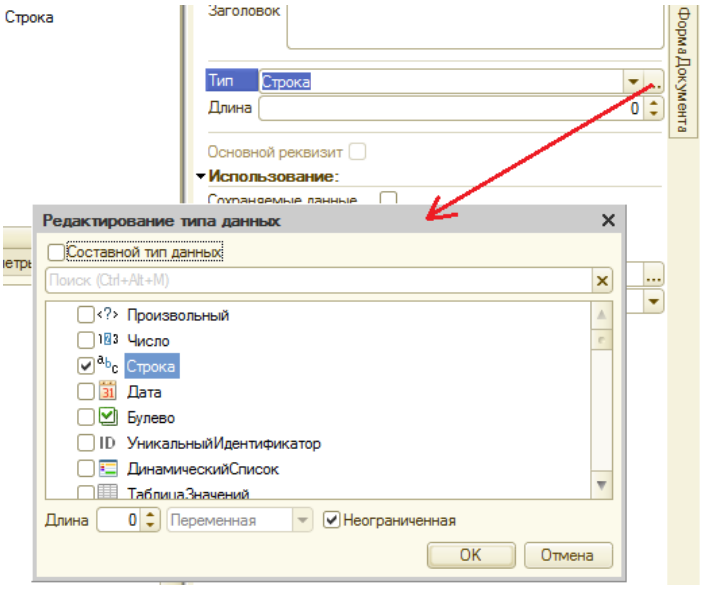


Рис. 5.3.4

Замечу, что при присвоении реквизиту формы таких типов, как ссылки, объекты и т.п., реквизит в окне отобразится в виде дерева. Где корнем будет название реквизита, а ветками этого дерева – реквизиты объекта выбранного типа.

Например, на рисунке 5.3.5 мы назначили новому реквизиту тип «ДокументСсылка.УстановкаЦенНаТопливо». И в списке реквизитов можно увидеть, что «Реквизит1» приобрел древовидную структуру, где в ветках дерева можно наблюдать реквизиты и табличную часть документа *Установка цен на топливо*.

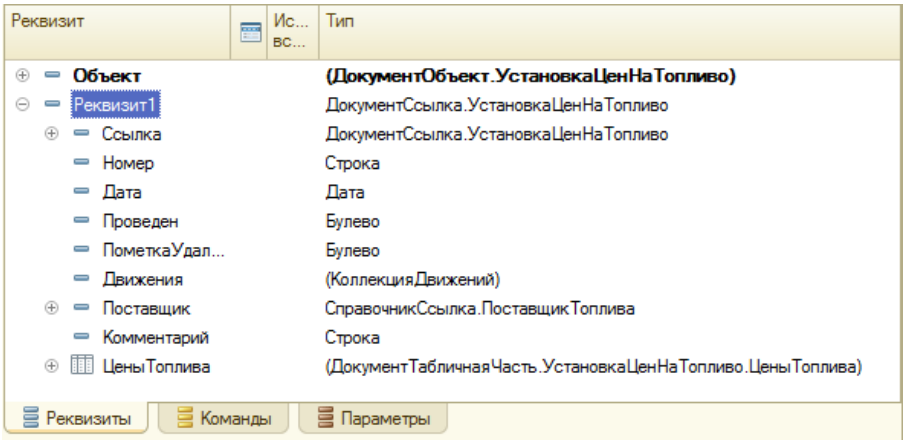


Рис. 5.3.5

Причем на форме можно будет разместить не только сам реквизит с ссылочным или объектным типом, но также и любой реквизит, который входит в состав объекта, образующего данный тип.

Ссылочные типы мы будем изучать в следующей части, а пока научимся работать с реквизитами примитивных типов. Поменяйте у реквизита *Реквизит1* тип на «Число», для этого зайдите в свойство реквизита *Тип*, нажав на кнопку «…», и установите новый тип.

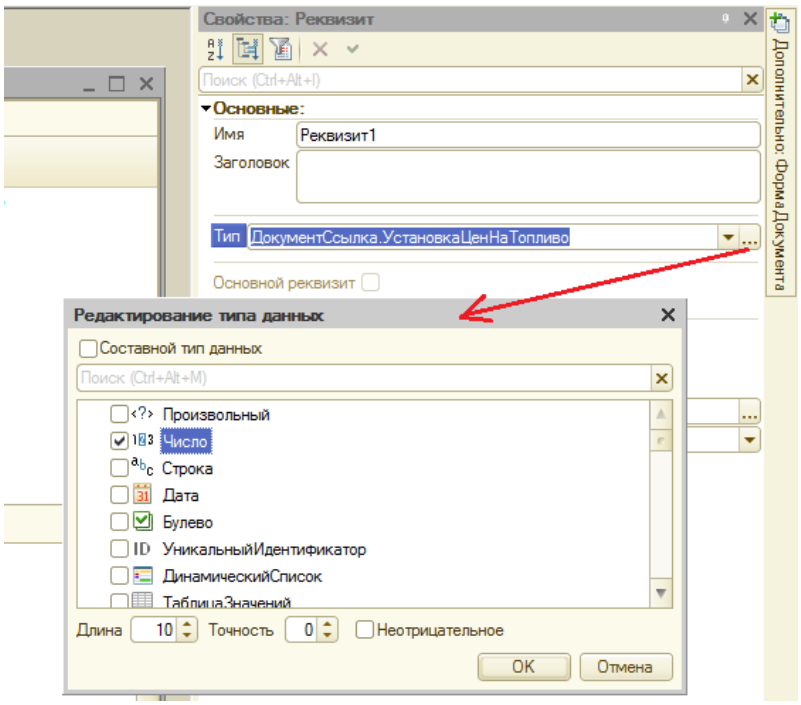


Рис. 5.3.6

Вернемся к свойствам реквизитов (см. рис. 5.3.7). Следующим после типа идет свойство «Сохраняемые данные». Если это свойство установлено, то при попытке редактирования элемента, связанного с этим реквизитом, будет установлен признак модифицированности формы (символ * рядом с названием).

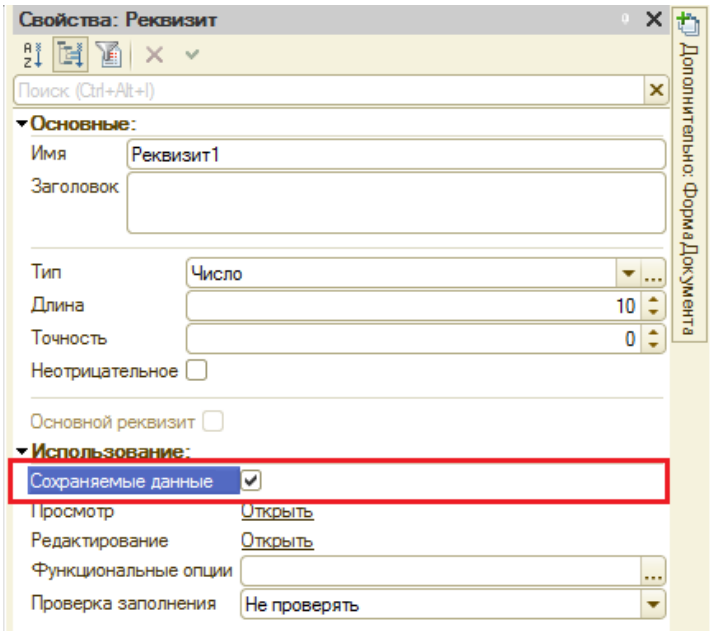


Рис. 5.3.7

Ещё одно интересное свойство – это «Проверка заполнения» (см. рис. 5.3.8).

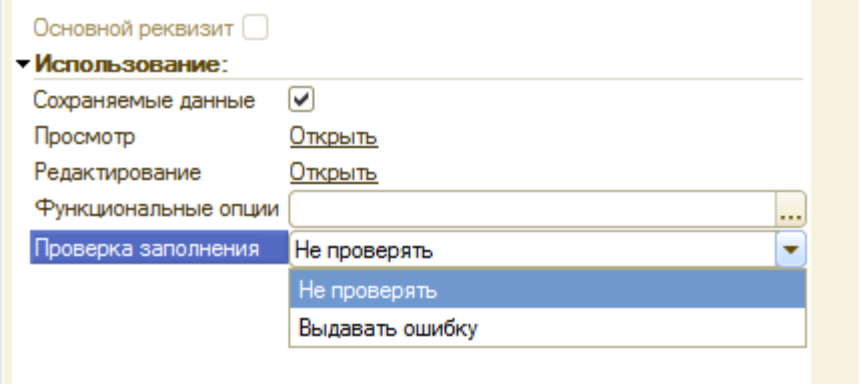


Рис. 5.3.8

Это свойство устанавливается для реквизитов с примитивными и ссылочными типами, а также для реквизитов с типами «Таблица значений» и «Список значений» (эти типы будем проходить в 7-й и 8-й главах).

После установки этого свойства реквизита в значение «Выдавать ошибку», незаполненный элемент формы, который привязан к реквизиту, будет подчеркиваться красной линией. А также если функциональность формы предусматривает сохранение данных (например, когда основной реквизит формы тип со значением какого-нибудь объекта), то при попытке сохранения данных, в случае незаполненного элемента формы, привязанного к этому реквизиту, программа выдаст ошибку и сохранение не произойдет.

Свойства «Просмотр» и «Редактирование» позволяют настроить видимость реквизита и возможность его редактирования в зависимости от ролей. После нажатия на гиперссылку «Открыть» реквизита «Просмотр» («Редактирование») появится окно настройки доступа (см. рис. 5.3.9). Если для роли установлен флаг, то пользователь, имеющий такую роль, будет видеть этот реквизит (сможет отредактировать), если флаг снят – то пользователь не будет видеть этот реквизит (не сможет редактировать), а если флаг – серый, то видимость этого реквизита (возможность редактировать) будет определяться флагом в шапке формы.

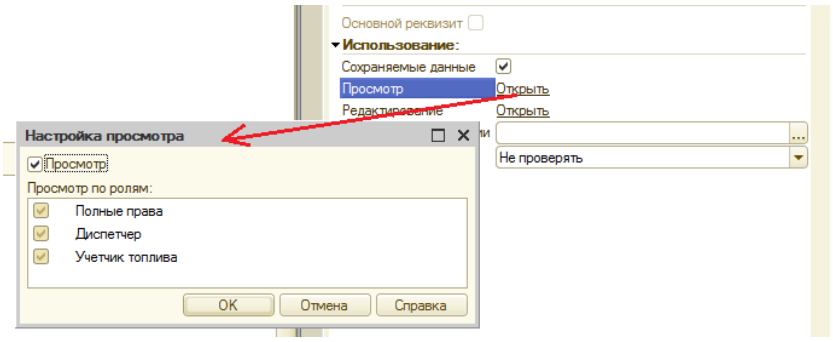


Рис. 5.3.9

Причем установка видимости элемента, к которому привязан реквизит, является вторичной по отношению к видимости этого реквизита: если у определенного реквизита запрещена видимость для определенной роли, то пользователь с этой ролью не увидит элемент, привязанный к этому реквизиту, в любом случае. Даже если на этот элемент пользовательская видимость установлена.

Вернемся к списку реквизитов на форме (см. рис. 5.3.10):

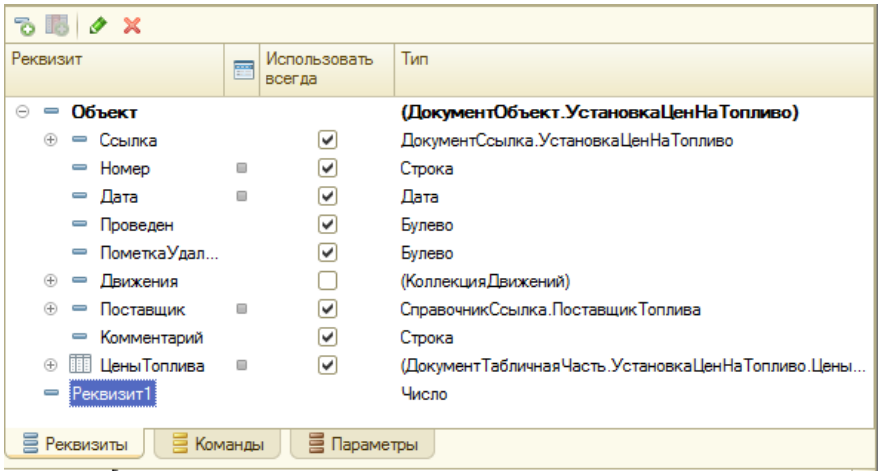


Рис. 5.3.10

Помимо колонок «Реквизит» и «Тип», у данной таблицы есть еще две колонки - «Расположение на форме» и «Использовать всегда». Если есть элемент, который привязан к реквизиту формы, то в колонке «Расположение на форме» появляется серый квадратик. С этим все просто. С полем «Использовать всегда» на данном этапе обучения не будем разбираться, чтобы не загружать Вас излишней информацией.

Элементы формы

Интерактивное управление формой осуществляется посредством элементов формы. Элементы расположены на форме в иерархическом порядке. Этот порядок определяет внешний вид формы и состав элементов управления формы. В конфигураторе работа с элементами формы осуществляется в редакторе формы на закладке «Элементы» (см. рис. 5.3.11), а сама же форма, её графический вид - на закладке «Форма» (см. рис. 5.3.11).

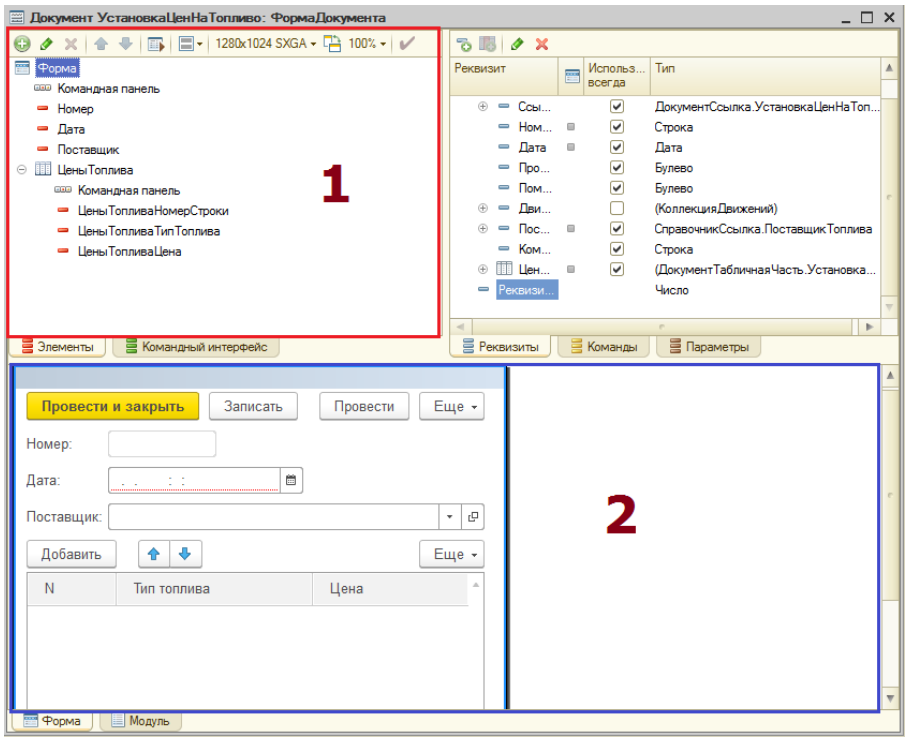


Рис. 5.3.11. Элементы формы (1) и вид формы (2)

Для тех, кто работал с обычными формами: нельзя работать с элементами формы (перемещать, удалять, растягивать и т.п.) непосредственно в графическом отображении формы. Вся работа с элементами ведется в закладке «Элементы».

Всего существует несколько видов элементов:

- Форма;
- Поле формы;
- Группа формы;
- Декорация формы;
- Таблица формы;
- Кнопка формы;
- Командные панели.

Позже мы разберем некоторые из них подробнее, пока отмечу, что элемент «Форма» существует только в *единственном* экземпляре. И располагается в самом верху иерархии. А элемент «Группа формы» может содержать в себе другие элементы формы.

Управление элементом возможно как с помощью мышки, так и с помощью командной панели, которая расположена в верхней части закладки «Элементы». Разберем некоторые кнопки командной панели (см. рис. 5.3.12).

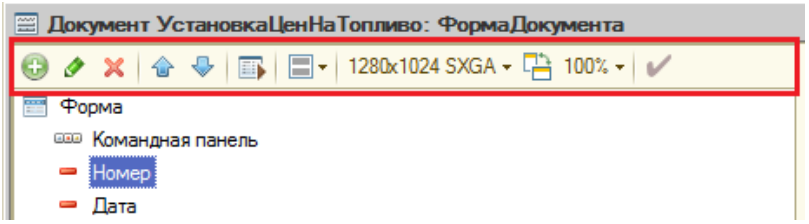


Рис. 5.3.12

Кнопка *Добавить* – первая кнопка на командной панели (см. рис. 5.3.12). При нажатии на неё открывается окно с выбором типа элемента (см. рис. 5.3.13).

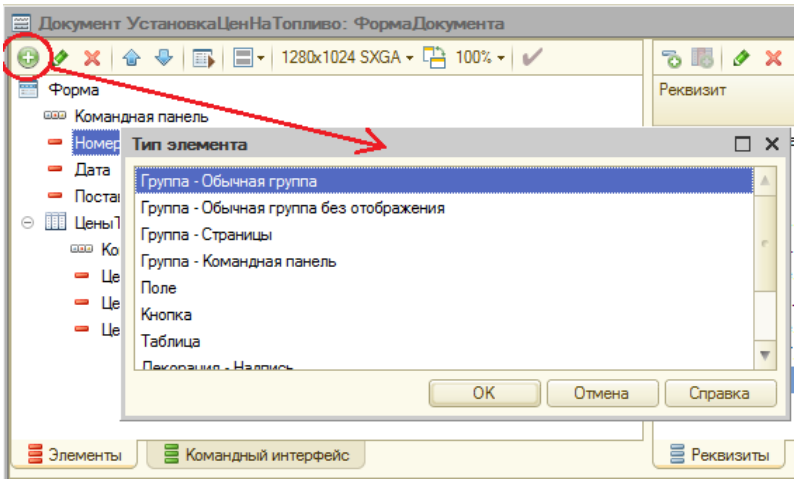


Рис. 5.3.13

После выбора нужного типа элемента и нажатия кнопки «ОК» этот элемент будет создан и помещен либо непосредственно на форму, либо в какую-нибудь группу, расположенную на форме. В зависимости от того, что у Вас на данный момент было активным.

При нажатии на кнопку «Изменить» (вторая на рисунке 5.3.12) откроется палитра свойств элемента.

С кнопками «Удалить», «Переместить вверх» и «Вниз» все понятно из названия, и, думаю, Вы в этом разберетесь самостоятельно.

При нажатии на кнопку «Проверить» (см. рис. 5.3.14) на экране показывается представление формы, какой её увидит пользователь.

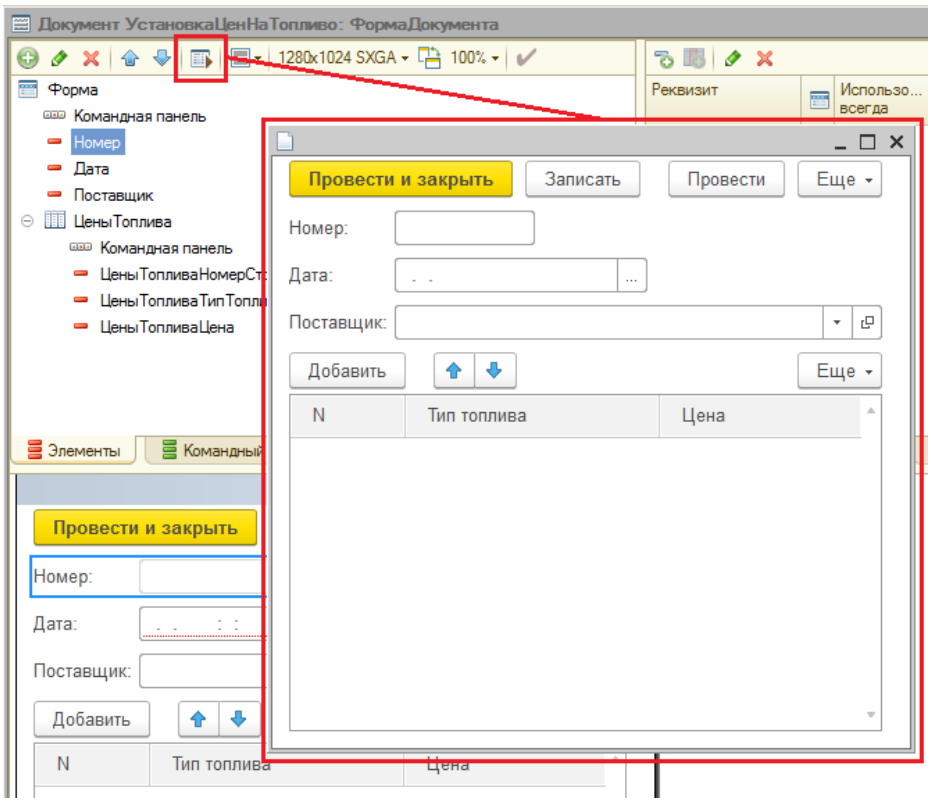


Рис. 5.3.14

Разберем основные элементы формы, с которыми Вам придется столкнуться в первое время.

Элемент форма

Самый главный элемент, без которого не обходится ни одна форма, это элемент «Форма». Данный элемент всегда располагается в верхней части дерева элементов и является родителем для всех элементов формы (см. рис. 5.3.15).

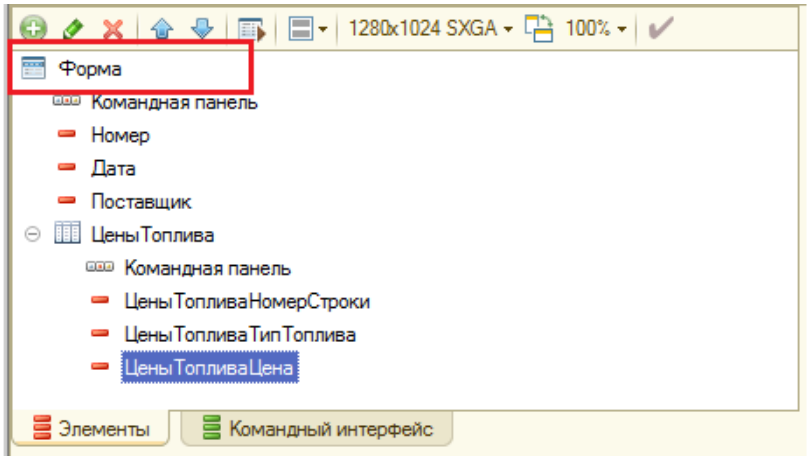


Рис. 5.3.15

Для того, чтобы попасть в палитру свойств формы, достаточно просто дважды кликнуть по элементу «Форма» мышкой, или, выделив форму, вызвать контекстное меню, где выбрать пункт «Свойства».

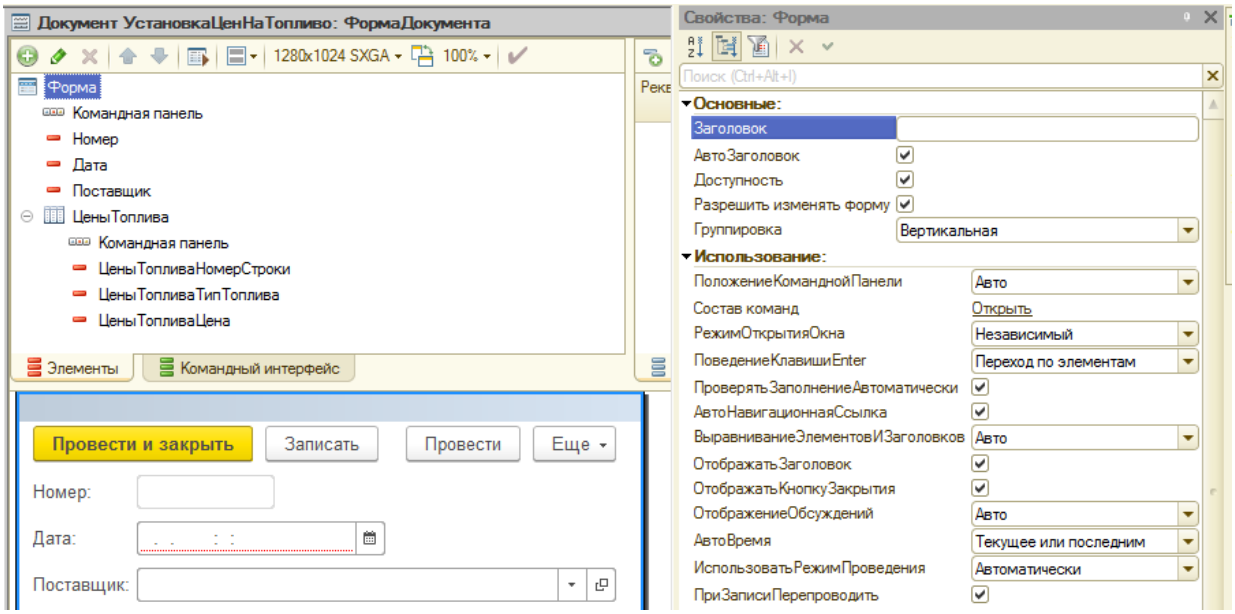


Рис. 5.3.16

Свойств у формы много, и я в рамках этой книги не буду все разбирать. Если у кого возникнет желание более глубоко изучить свойства формы, то он найдет всю нужную информацию в справке к программе 1С. С некоторыми свойствами Вы можете познакомиться в 1-й главе 3-й части моей книги «Основы разработки в 1С: Такси». В этой книге изучим только одно интересное свойство: *РежимОткрытияОкна*.

Режим открытия окна – очень важное свойство формы. Определяет, каким образом будет открываться наша форма. Может принимать три значения: независимый, блокировать окно владельца, блокировать весь интерфейс.

В этой книге рассмотрим режимы открытия окна только при работе с интерфейсом Такси.

При использовании *Независимого* режима открытия окна, новая форма откроется в рабочей области интерфейса. Она будет как бы «рядом» с тем окном, откуда было инициировано ей открытие.

Например, на рис. 5.3.17 мы открыли форму документа *Установка цен на топливо* из формы списка документов. Эта форма стала сама по себе (независимая) и мы как с помощью стрелок навигации, так и с помощью панели навигации можем перейти с этой формы на ту форму, откуда она была открыта.

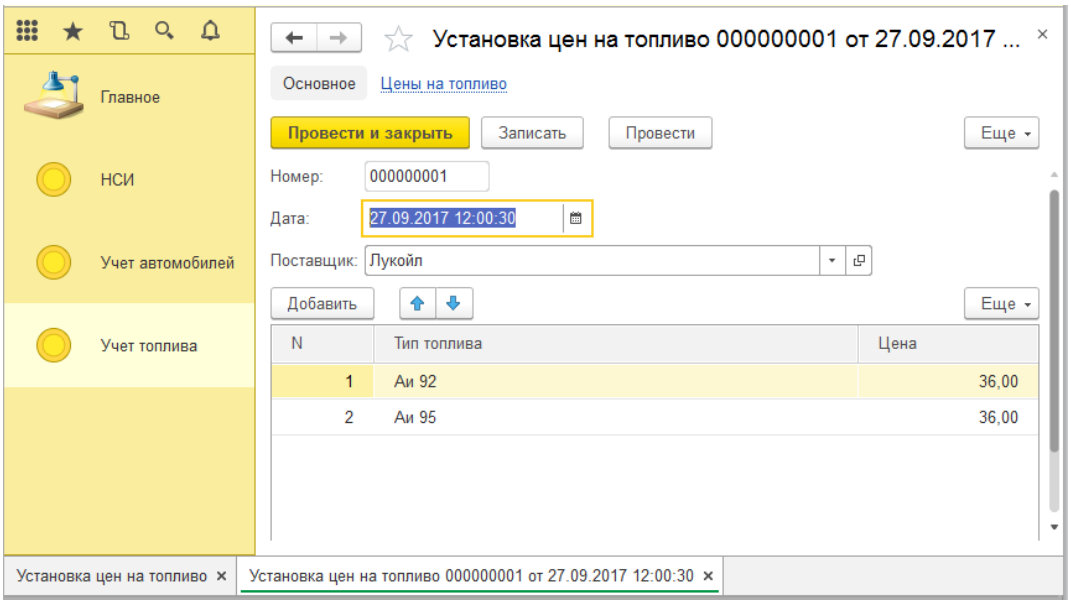


Рис. 5.3.17

Если используется режим *Блокировать окно владельца* (см. рис. 5.3.18), то форма открывается «поверх» окна, где было инициировано её открытие. Блокирует это окно, но в то же время пользователь сможет спокойно осуществлять навигацию по всему интерфейсу.

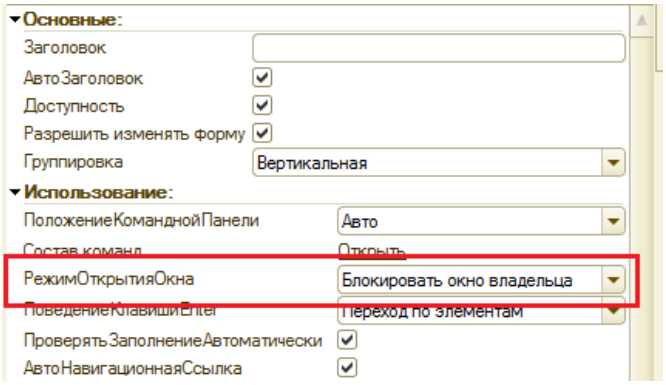


Рис. 5.3.18

Например, на рис. 5.3.19 мы открыли форму документа *УстановкаЦенНаТопливо* из списка документов с режимом *Блокировать окно владельца*. У нас есть возможность перемещаться по интерфейсу, открыть еще одно окно, но чтобы вернутся в список документов, необходимо закрыть окно документа.

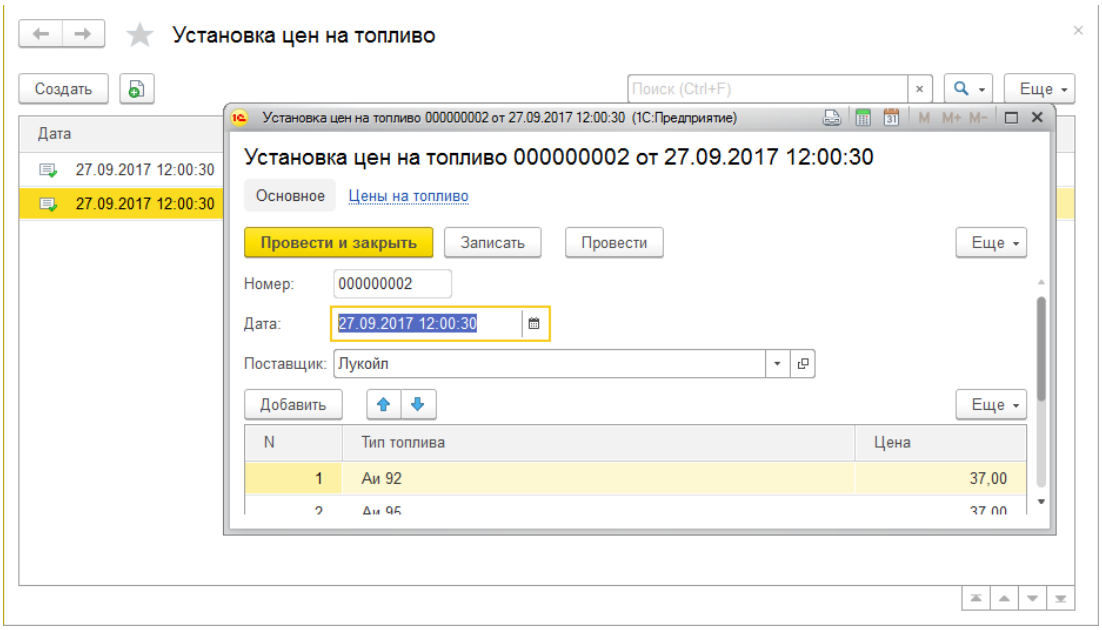


Рис. 5.3.19

Данный режим желательно применять, когда требуется открыть форму, в которую нужно ввести небольшое количество информации

По умолчанию этот режим устанавливается у следующих видов форм:

- Формы элементов и групп справочников
- Формы элементов и групп планов видов характеристик
- Форма счета
- Форма задачи и т.д.

В случае использования режима *Блокировать весь интерфейс* окно также открывается поверх окна, которое инициировало его открытие, и блокируется весь интерфейс «1С:Предприятия». Пока пользователь не закроет это окно, он не сможет получить доступ ни к одной команде «1С:Предприятия».

Группа формы

Этот элемент формы необходим для реализации всевозможных вариантов объединения остальных элементов формы. В управляемой форме мы не можем самостоятельно размещать элементы на форме, но иногда для удобства работы пользователей возникает необходимость расположить элементы относительно друг друга в различных вариациях. Например, у нас в документе *УстановкаЦенНаТопливо* поля «Дата» и «Номер» расположены друг под другом, а более удобно, например, чтобы они располагались рядом по горизонтали.

Всего можно создать 4 варианта группы: «Обычная», «Обычная без отображения», «Страницы» и «Командная панель» (см. рис. 5.3.20). Также можно сгруппировать колонки в

таблице формы (эту группу мы изучим, когда будем проходить таблицы на форме, см. глава 8, часть 1, стр. 509).

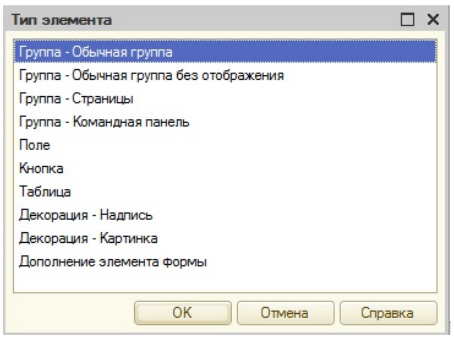


Рис. 5.3.20

Обычная группа (обычная группа без отображения)

Добавим на нашу форму документа *УстановкаЦенНаТопливо* обычную группу.

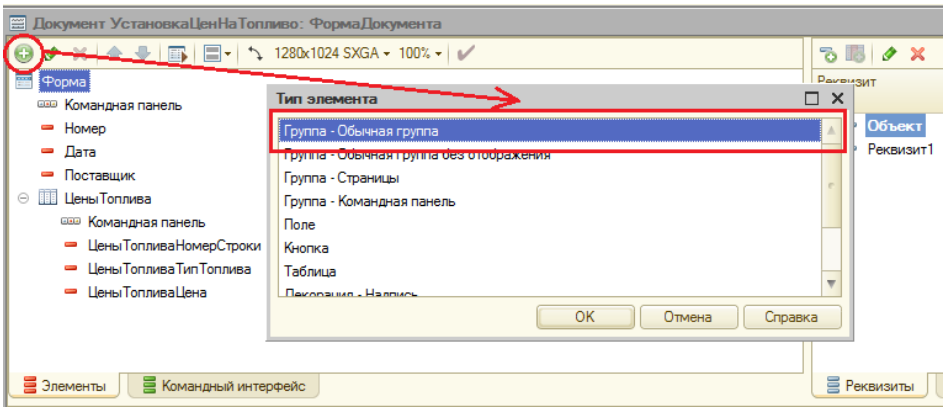


Рис. 5.3.21

По умолчанию группа добавится вниз элементов формы, перетащите её вверх мышкой или при помощи кнопки «Переместить вверх». В закладке *Элементы* группы отображаются в виде пиктограммы «Папка» (см. рис. 5.3.22).

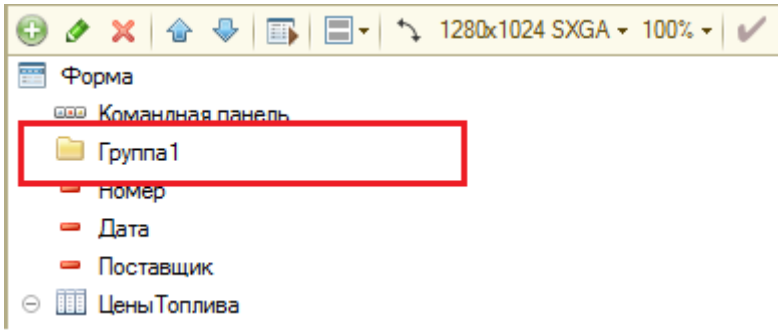


Рис. 5.3.22

Пока мы никак не увидим нашу группу на форме, для этого необходимо перетащить нужные элементы в группу. Делается это с помощью мышки: просто выделяется нужный элемент

и «тащится» в группу. Я перенесу в группу элементы «Номер» и «Дата», и на рис. 5.3.23 можно увидеть, как это отобразится на нашей форме.

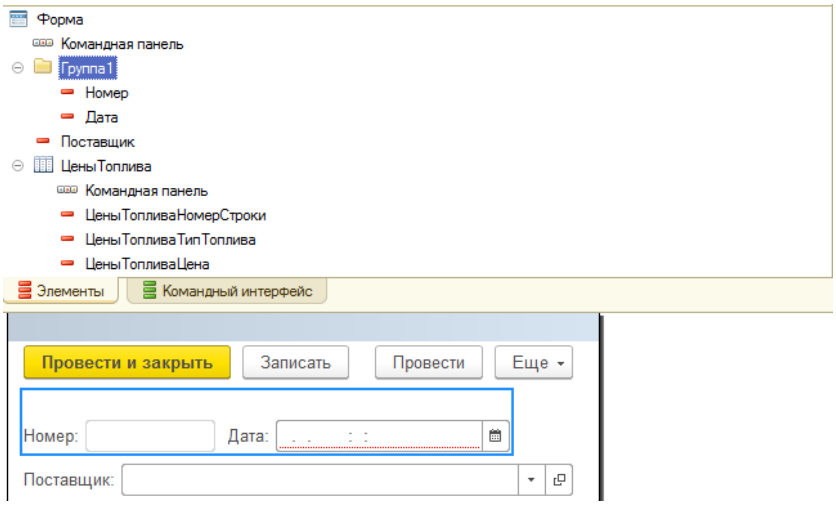


Рис. 5.3.23

Элементы в «Группа1» расположились согласно свойствам группы по умолчанию. Сейчас мы перейдем в палитру свойств группы и узнаем, какие свойства за что отвечают. Для того, чтобы перейти в палитру свойств группы, необходимо или дважды кликнуть мышкой по группе, или в контекстном меню группы выбрать пункт «Свойства». В открывшейся палитре (см. рис. 5.3.24) нас в основном интересуют свойства из закладки «Основные».

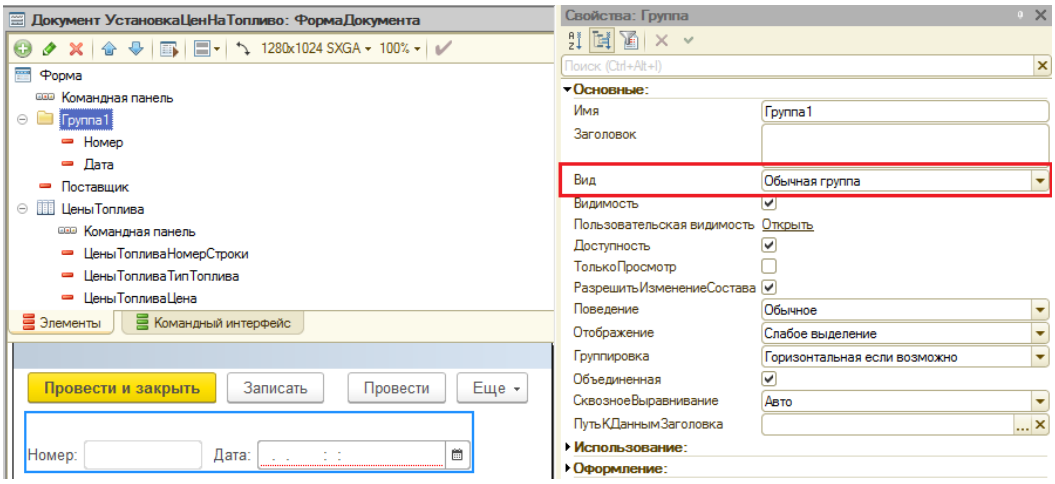


Рис. 5.3.24

И первым делом обратите внимание на свойство *Вид* – *Обычная группа*. Всего существуют три вида группы. В этом подразделе мы будем работать с *Обычной группой*. При выборе групп можно выбрать «Обычную группу» или «Обычную группу без отображения» (см. рис. 5.3.24), но в обоих случаях будет выбрана одна и та же группа с видом *Обычная группа*. Только во втором случае будет снято свойство *ОтображатьЗаголовок* (в категории Использование). Больше никаких различий между двумя этими видами в форме выбора нет. По сути, это одна и та же группа, разделение на «Обычную группу» или «Обычную группу без отображения» сделано больше для удобства.

Перейдем к следующему свойству, это «Группировка». Именно от значения в этом свойстве будет зависеть, как будут располагаться друг относительно друга элементы в группе. По умолчанию оно всегда установлено: «Горизонтальная если возможно». Если мы установим свойство «Вертикальная», то расположение элементов изменится (см. рис. 5.3.25).

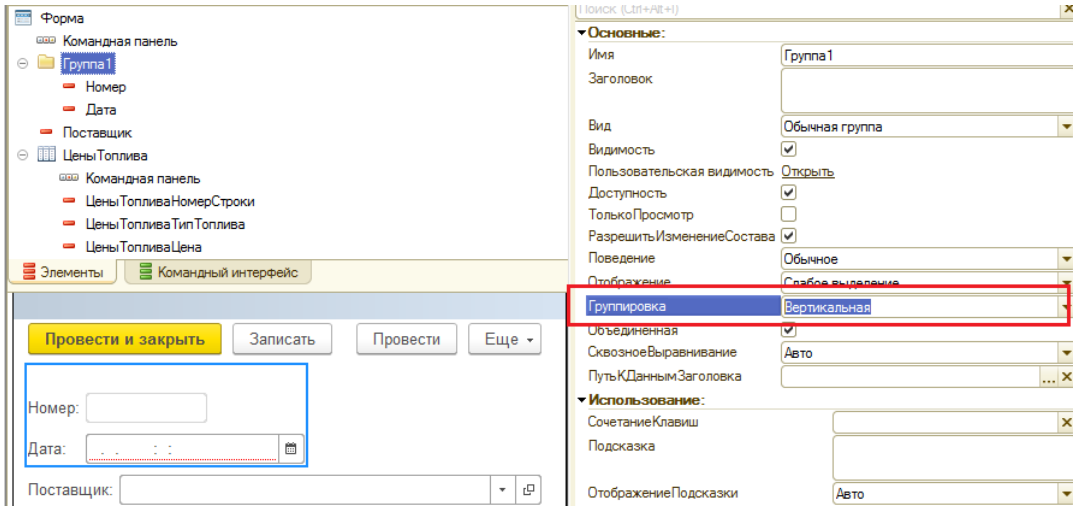


Рис. 5.3.25

Зададим заголовок группы, назовем её «Номер и дата» (см. рис. 5.3.26).

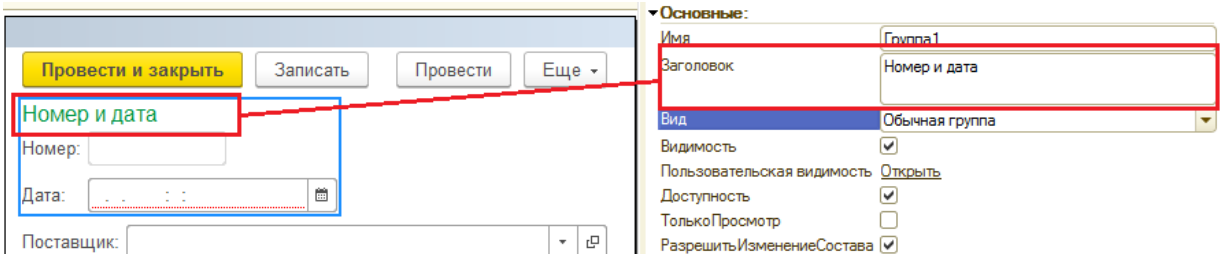


Рис. 5.3.26

Обратите внимание: если свойство *Отображать заголовок* снято, то заголовок не будет отображаться. Причем в любом случае, заполнено свойство «Заголовок» или нет.

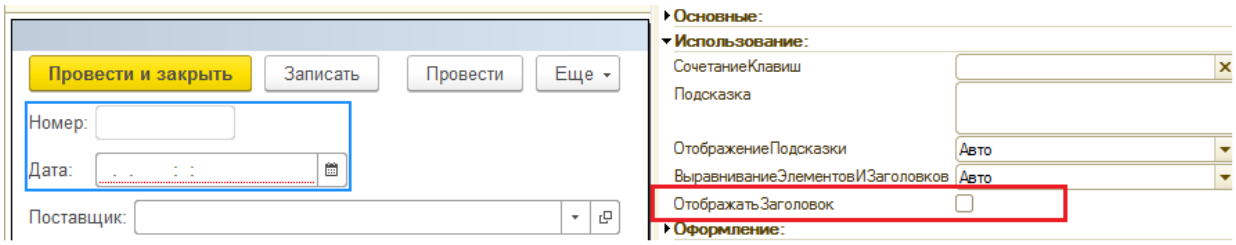


Рис. 5.3.27

Объем этой книги не позволяет изучить все свойства групп, да и на данном этапе обучения это не нужно. Пока научитесь работать со свойствами «Вид» (о других значениях позже), «Группировка», «Заголовок» и «Отображать заголовок».

Страницы

Рассмотрим следующий очень интересный вид группы - это *страницы*. С помощью этой группы можно создавать различного рода закладки и размещать по ним элементы. Например, нашу форму документа *УстановкаЦенНаТопливо* можно реализовать в таком виде:

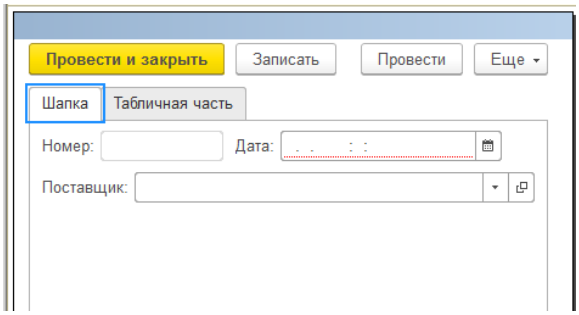


Рис. 5.3.28

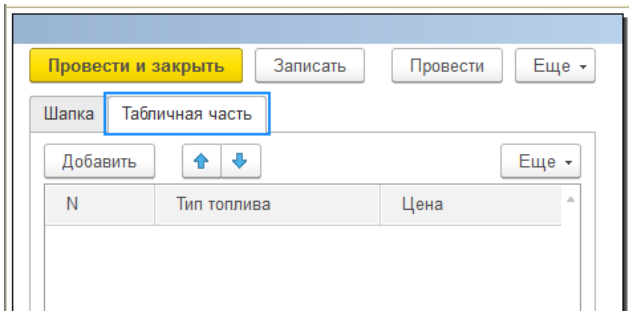


Рис. 5.3.29

Реализуем интерфейс формы документа *УстановкаЦенНаТопливо* с закладками как на рис. 5.3.28 и 5.3.29. Поставим курсор на узел «Форма», нажмем на кнопку «Добавить» командной панели закладки «Элементы» и выберем элемент с названием «Группа - Страницы» (см. рис. 5.3.30)

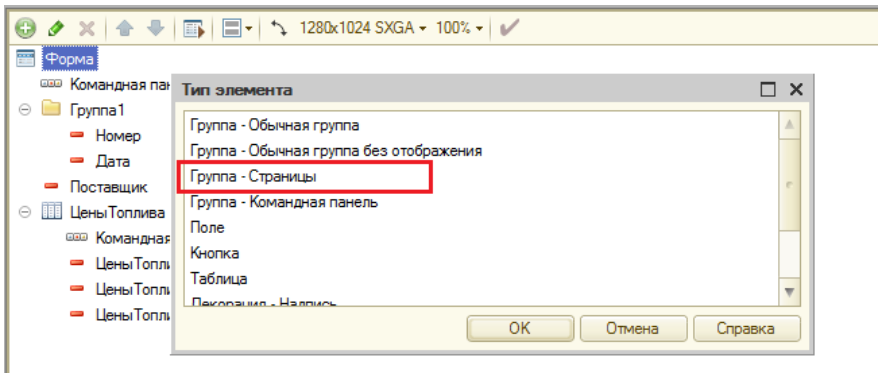


Рис. 5.3.30

После этого в дереве «Форма» закладки *Элементы* добавится строка с уже знакомой Вам пиктограммой «папка». Только свойство *Вид* этой группы будет *Страницы* (см. рис. 5.3.31).

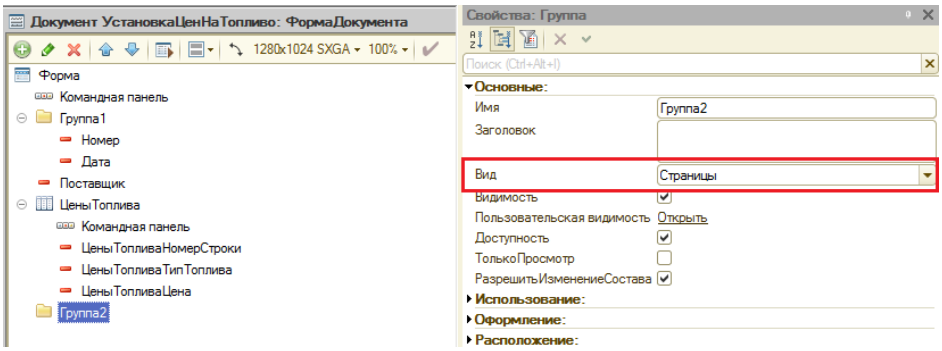


Рис. 5.3.31

Сейчас мы разместили общую группу с видом *Страницы*, но нам её еще необходимо разбить на страницы. У нас на форме должно быть две страницы, одна для реквизитов *шапки*, а вторая для табличной части *Цены*. Для того чтобы добавить новую страницу к группе, необходимо установить курсор на нужную группу (с видом *Страницы*, не забываем) и нажать на кнопку «Добавить» закладки элементы. В появившемся окне выбора элементов добавится элемент «Группа - страница» (см. рис. 5.3.32).

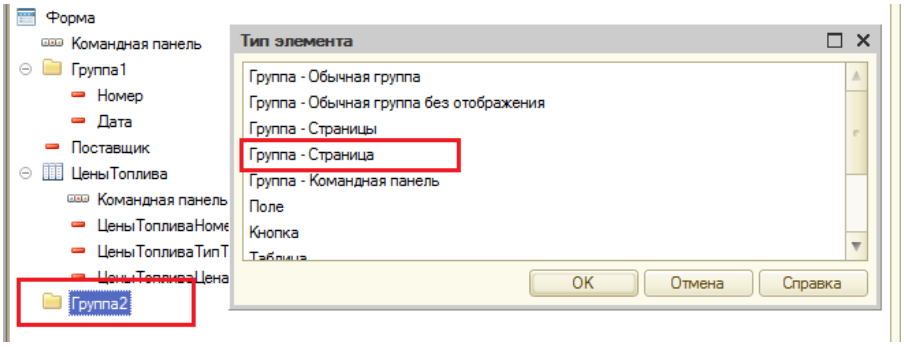


Рис. 5.3.32

Подобные действия нужно осуществить столько раз, сколько страниц в группе Вы хотите сделать (в нашем случае две). После этого элемент *Группа2* в дереве формы приобретет иерархический вид (см. рис. 5.3.33, самостоятельно переместите элемент *Группа2* в верх формы).

Рис. 5.3.33

Обратите внимание, что в группах с видом «Страница» тоже можно устанавливать различные группировки с помощью свойства «Группировка» (см. рис. 5.3.34).

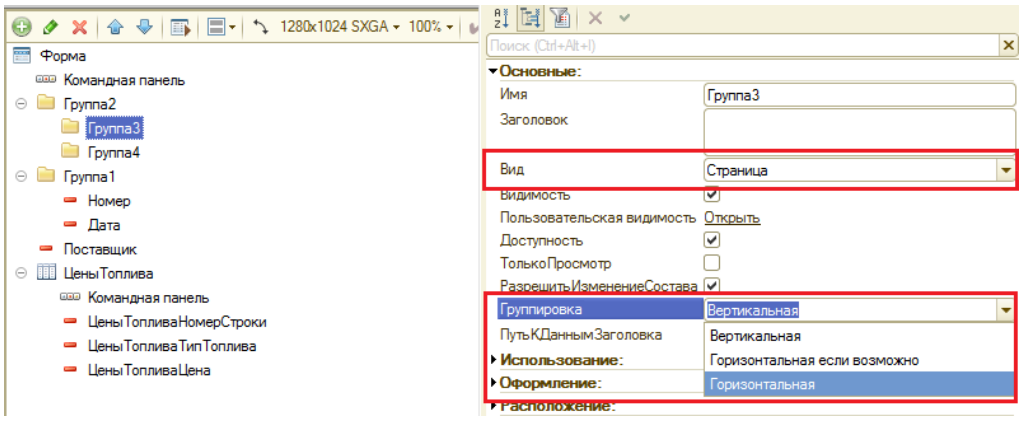


Рис. 5.3.34

Нам осталось с помощью мышки «перетащить» нужные элементы по нужным страницам, и наша форма примет вид почти как на рис. 5.3.28 и 5.3.29

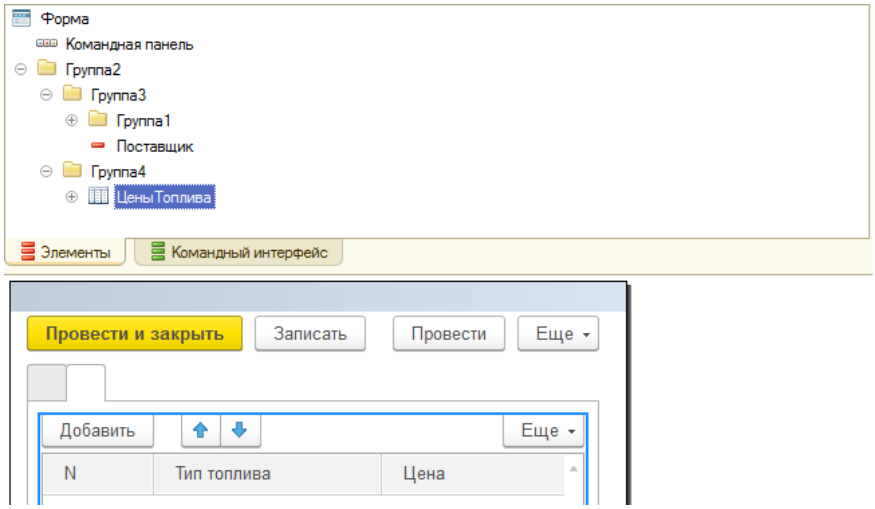


Рис. 5.3.35

Вы заметили, что на рис. 5.3.35 наша группа без страниц. Это потому, что мы не задали заголовки элементам *Группа3* и *Группа4*. Дадим имя заголовку элемента *Группа3* – «Шапка» (см. рис. 5.3.36), а *Группа4* – «Таблица». После этого наша таблица примет более красивый вид (см. рис. 5.3.37).

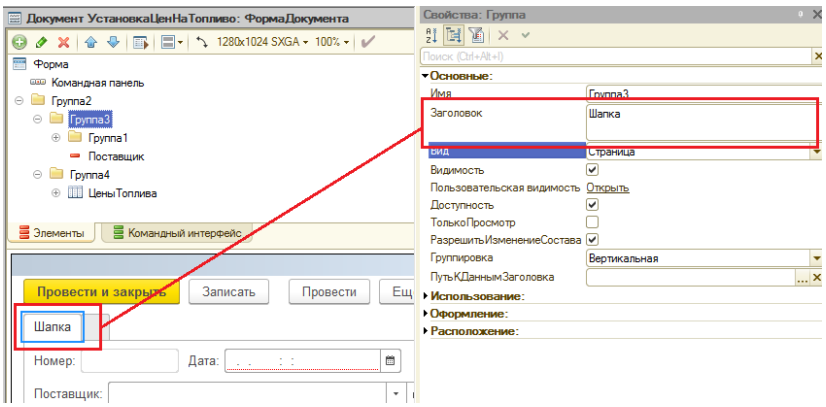


Рис. 5.3.36

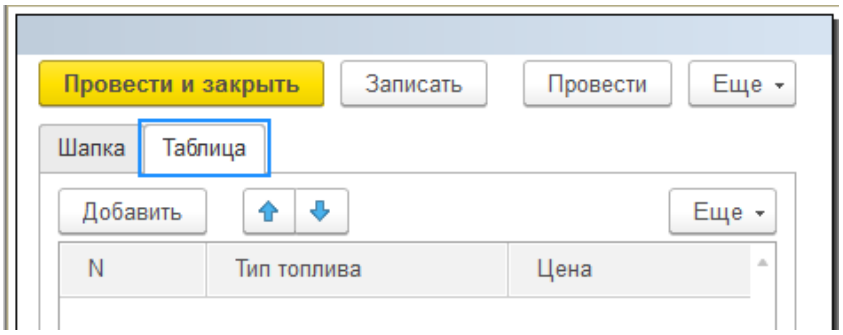


Рис. 5.3.37

Командная панель

Группа с видом *Командная панель* предназначена для размещений различных кнопок и групп. Причем для групп с таким видом можно указывать *Источник*, в этом случае группа будет автоматически заполнена командами источника по умолчанию. Изменим представление формы документа *УстановкаЦенНаТопливо*: «перетащим» командную панель формы в закладку «Шапка».

Начнем. Первым делом нам необходимо отключить командную панель формы. Делается это посредством установки свойства формы *ПоложениеКоманднойПанели* в значение *Нет* (см. рис. 5.3.38).

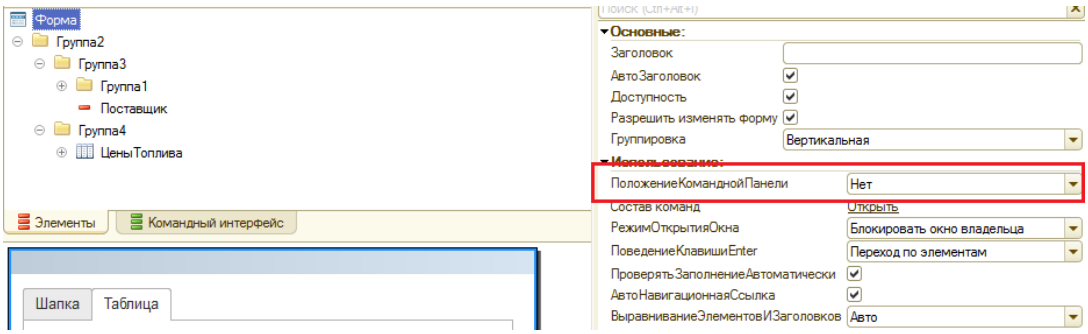


Рис. 5.3.38

Добавим новую группу с видом *Командная панель* в страницу *Шапка*. Для этого необходимо установить курсор на элемент *Группа3* (поскольку именно он является закладкой *Шапка*) и нажать на кнопку «Добавить» (см. рис. 5.3.39).

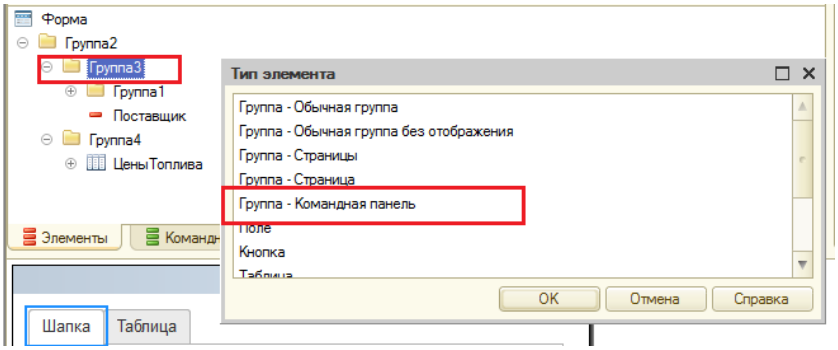


Рис. 5.3.39

После добавления новой группы, переместите её вверх над элементом *Группа1*.

В открывшейся палитре свойств добавленной группы нам нужно указать источник команд. В нашем случае три источника, это команды формы, глобальные команды командной панели формы и команды таблицы «Цены топлива» (см. рис. 5.3.40). Мы выберем источник – «Форма». И после этих действий наша группа сразу же заполнится стандартными командами формы (см. рис. 5.3.41).

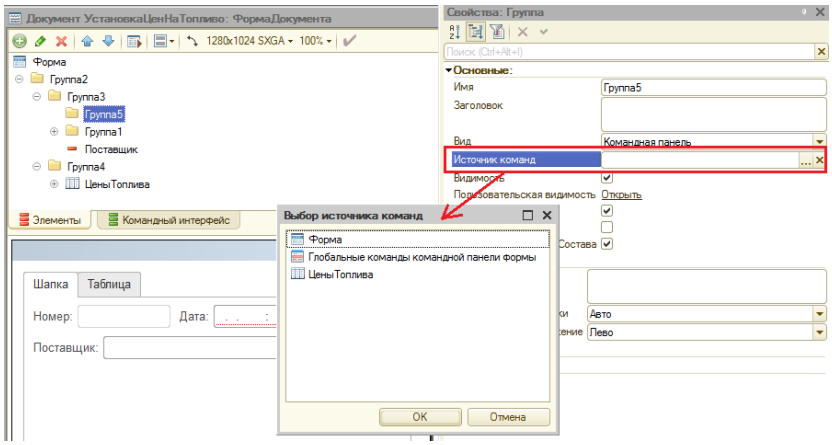


Рис. 5.3.40

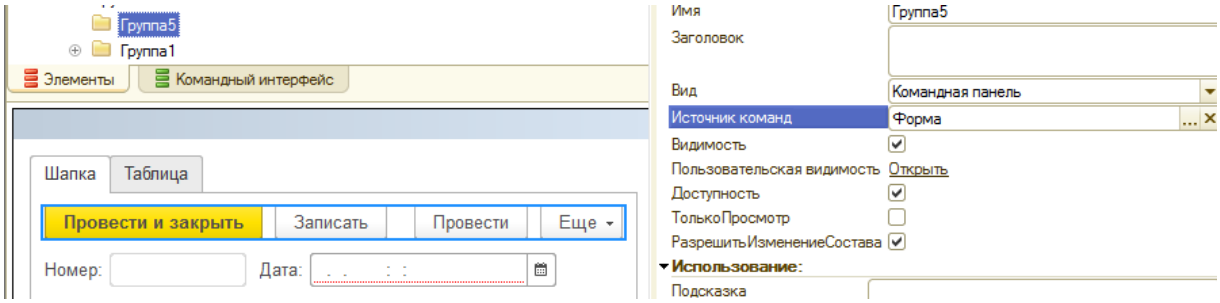


Рис. 5.3.41

На этом мы закончим изучать элемент формы групп, информации, предоставленной в этой главе, Вам хватит, чтобы объединять элементы в простые группы.

Для дальнейшей работы самостоятельно приведите форму в первоначальный вид.

Поля

Следующая очень обширная группа элементов, которые мы рассмотрим, это поля ввода. С полями ввода мы практически не работали. В этом разделе мы научимся работать с полями ввода, и в дальнейшем для ввода какой-то информации в программу будем использовать этот элемент формы.

Посредством элемента формы *Поле ввода* пользователь имеет возможность просматривать и редактировать реквизит формы. Создать его на форме можно двумя способами. Первый - с помощью кнопки «Добавить» командной панели закладки элементы, после нажатия на которую, откроется форма выбора типа элемента, где и выбираем нужный нам элемент с типом «Поле» (см. рис. 5.3.42).

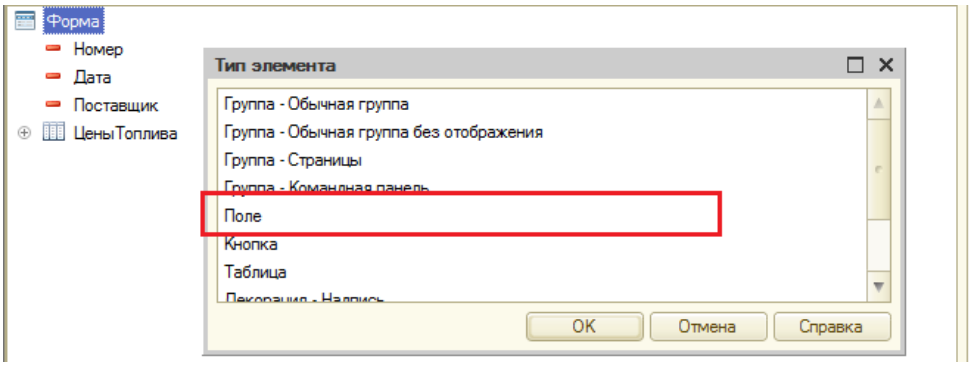


Рис. 5.3.42

Второй способ заключается в перетаскивании нужного реквизита из закладки «Реквизиты» на форму или в группу формы (см. рис. 5.3.43).

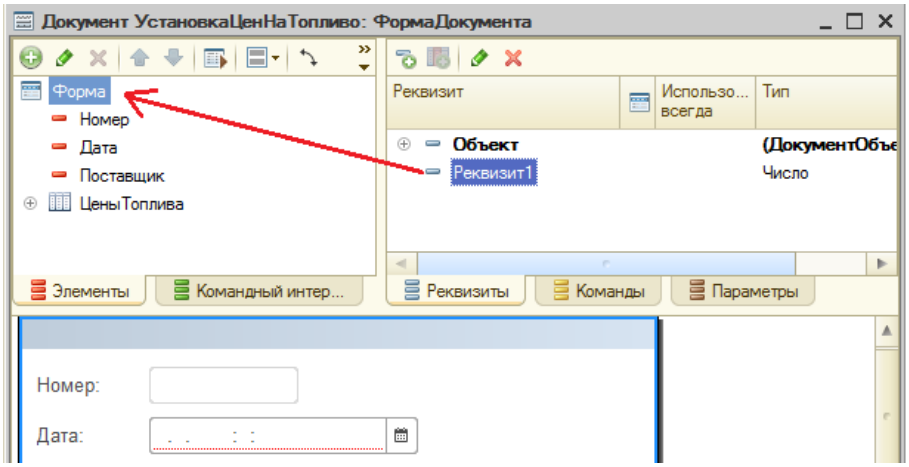


Рис. 5.3.43

На форму можно перетаскивать не только реквизиты формы, но и реквизиты реквизитов. Особенно это применимо в том случае, когда основной реквизит формы объект справочника или документа.

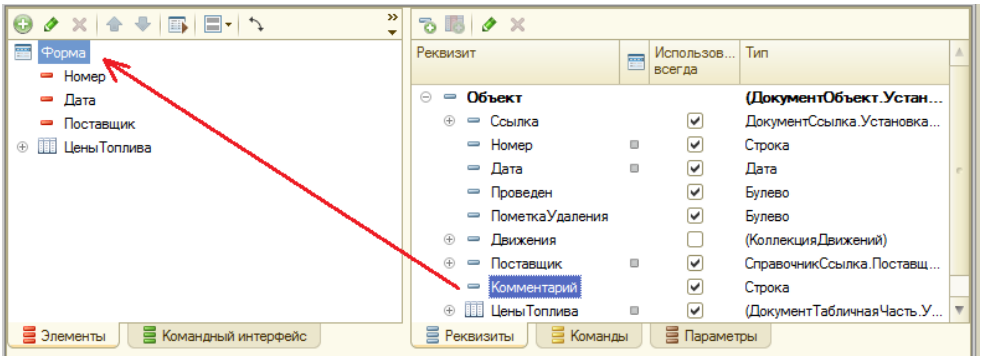


Рис. 5.3.44

При простом добавлении поля ввода (первый способ) на отображении формы никакое поле не появится. Так произойдет потому, что свойство *ПутьКДанным* элемента формы *Поле* не заполнено (см. рис. 5.3.45). Если же мы будем перетаскивать реквизит формы, то это свойство заполнится автоматически (см. рис. 5.3.46), и как следствие он сразу отобразится на экране.

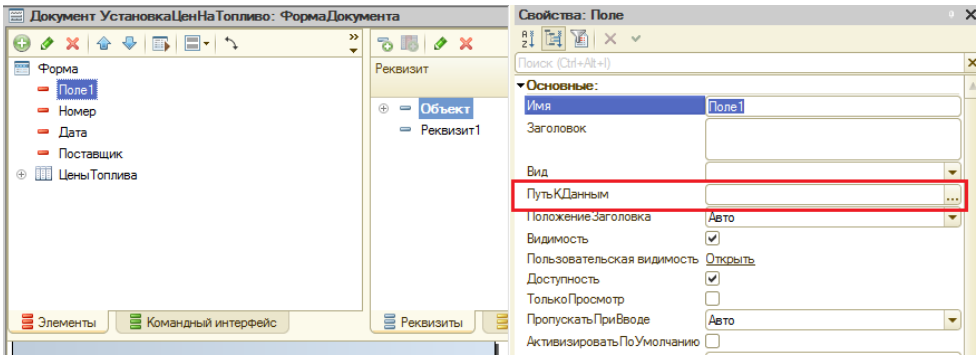


Рис. 5.3.45

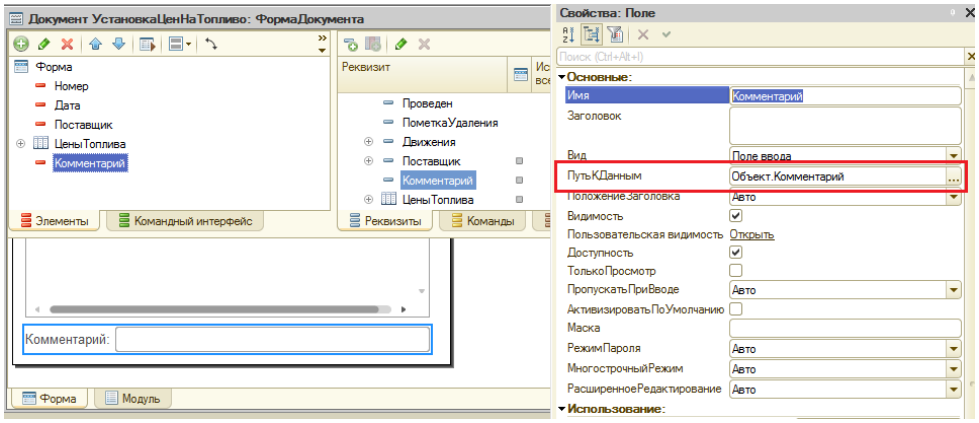


Рис. 5.3.46

Разработчик имеет возможность самостоятельно отредактировать свойство *ПутьКДанным* поля: можно выбрать или реквизит формы или, если у реквизита формы ссылочный (объектный) тип, реквизит реквизита формы (см. рис. 5.3.47).

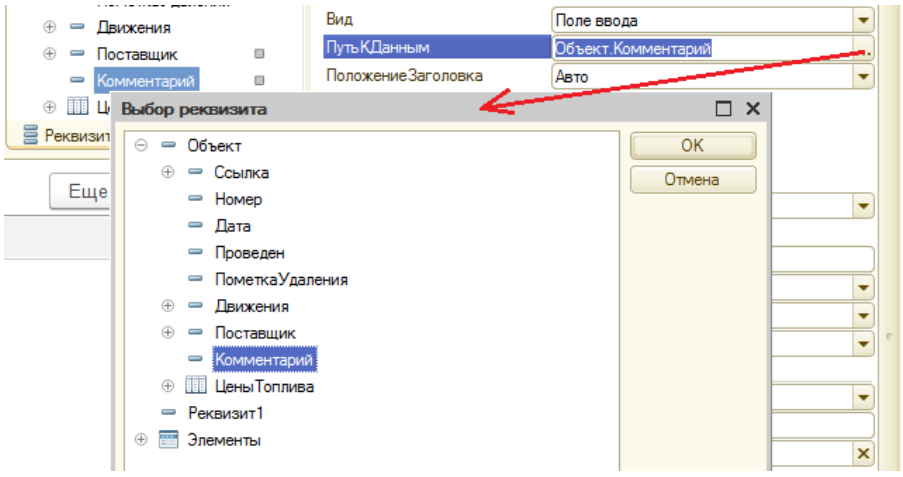


Рис. 5.3.47

Элементов *Поле* может быть множество видов. Мы рассмотрим самые нужные для дальнейшего изучения.

Вид «Поле ввода»

И самый нужный вид элемента *Поле* это – *Поле ввода* (см. рис. 5.3.48). С помощью поля с этим видом мы можем вводить практически любые значения: примитивные (строка, число, дата, булево) и ссылочные (ссылка на справочники, документы и т.д.).

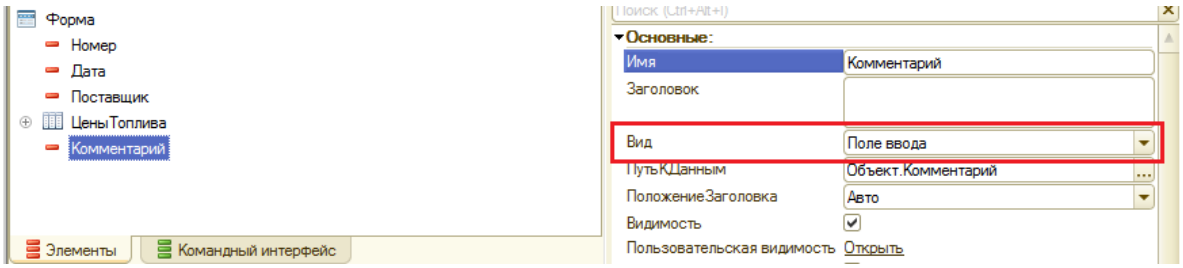


Рис. 5.3.48

Внешний вид *поля* на форме будет зависеть от того, какой тип имеет реквизит, указанный в свойстве поля *ПутьКДанным*. Для реквизита с типом строка внешний вид поля показан на рис. 5.3.49. Для типов число, булево и дата – на рис. 5.3.50, 5.3.51, 5.3.52. А для ссылочного типа («СправочникСсылка.ТипыТоплива») – на рис. 5.3.53.

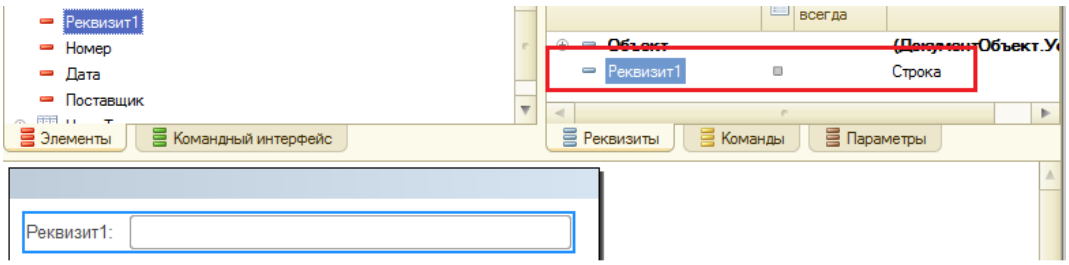


Рис. 5.3.49

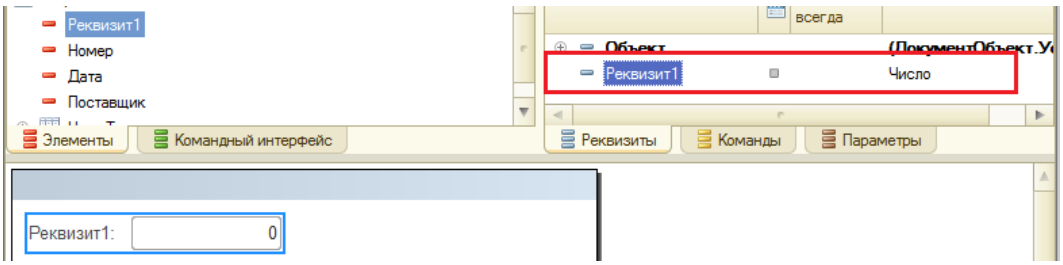


Рис. 5.3.50

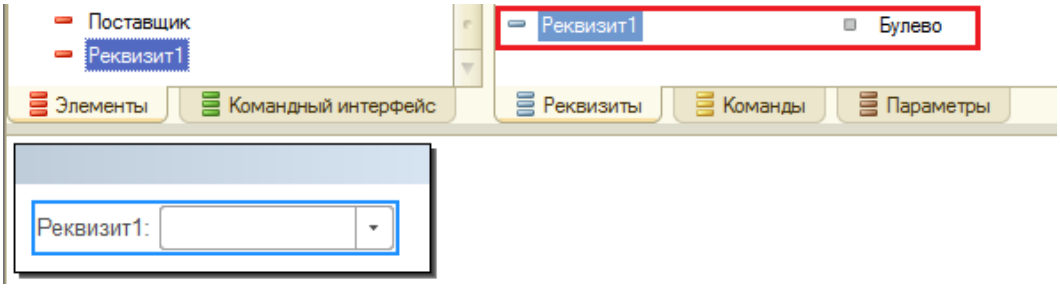


Рис. 5.3.51

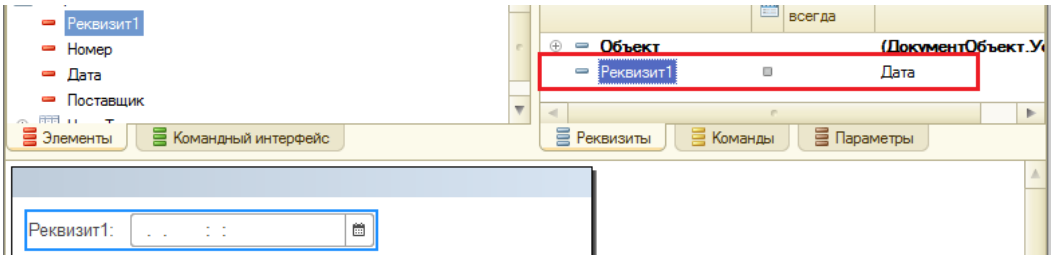


Рис. 5.3.52

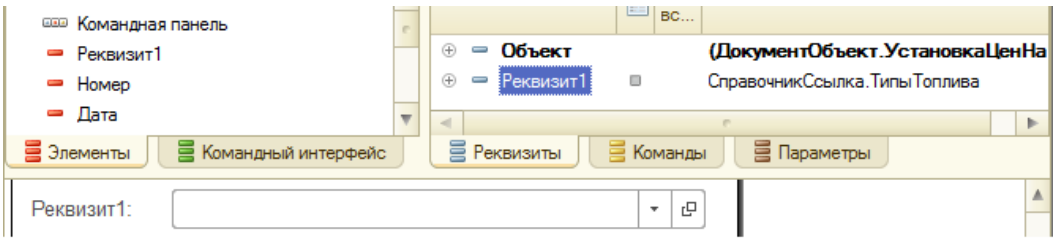


Рис. 5.3.53

Если тип реквизита *Строка*, то в некоторых случаях у элемента формы *Поле* есть смысл устанавливать свойство *МногострочныйРежим* в значение *Да* (см. рис. 5.3.54). Например, это нужно сделать тогда, когда предполагается, что пользователь будет вводить информацию, которая может не войти по длине поля ввода.

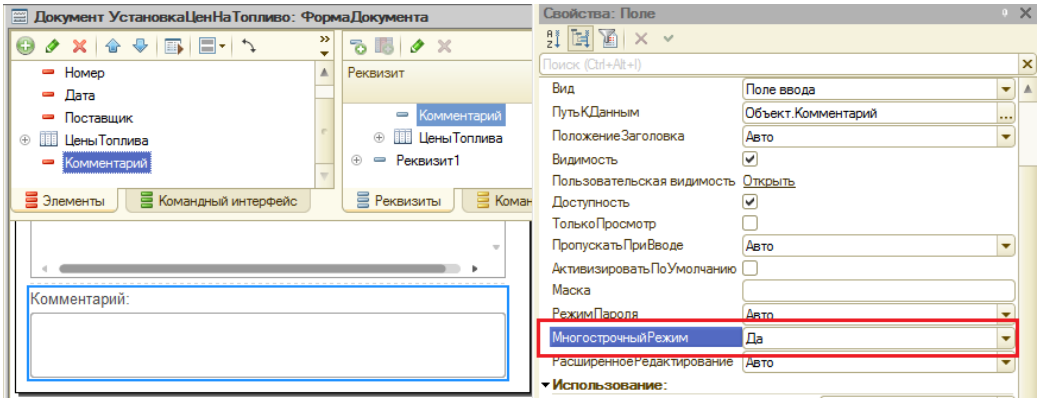


Рис. 5.3.54

Еще одна интересная возможность *Поля* с типом *Поле ввода* - это возможность выбирать какие-либо значения из заранее созданного списка. Для реализации такой возможности нам понадобятся два свойства, это – *РежимВыбораИзСписка*, его нужно включить, и свойство *СписокВыбора*. После нажатия на кнопку «…» свойства *СписокВыбора* откроется окно, в которое можно внести все варианты выбора (см. рис. 5.3.55).

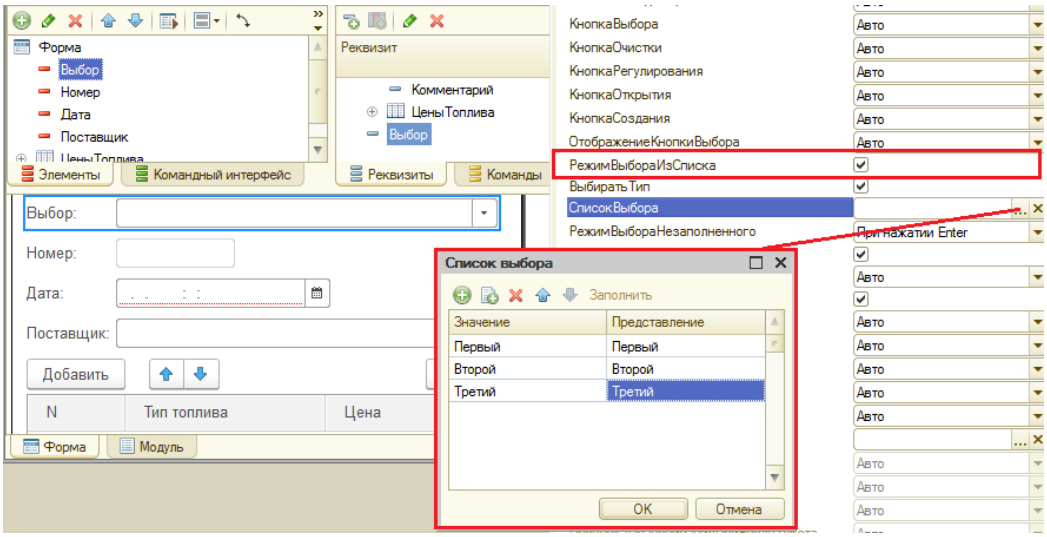


Рис. 5.3.55

Причем не важно, какого типа будет реквизит, просто разными значениями будет заполняться поле *Значение* формы списка выбора.

Для ссылочных типов в конфигураторе можно будет выбрать только предопределенные значения.

После таких настроек у поля ввода на форме появится кнопка списка, при нажатии на которую выпадет список выбора (см. рис. 5.3.56).

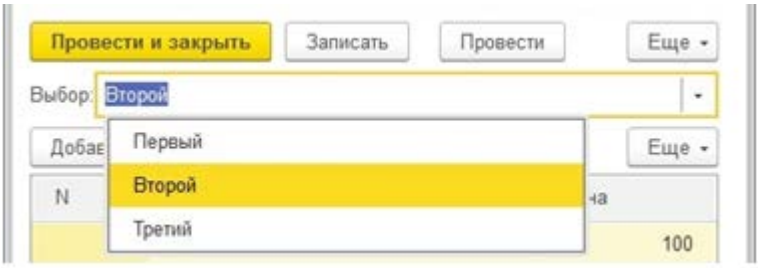


Рис. 5.3.56

Вид «Поле надписи»

Этот вид элемента *Поле* можно применять тогда, когда нужно просто для информации вывести какое-либо значение и у пользователя даже мысли не должно возникнуть о попытке редактирования. Например, добавим на форму документа *УстановкаЦенНаТопливо* поле, в котором будем показывать текущий тип топлива (на который установлен курсор) таблицы формы «Цены».

Сначала просто добавим элемент *Поле* при помощи кнопки «Добавить» командной панели окна *Элементы*, а потом в свойстве ПутьКДанным укажем путь полю *ТипТоплива* текущих данных элемента формы *ЦеныТоплива* (см. рис. 5.3.57).

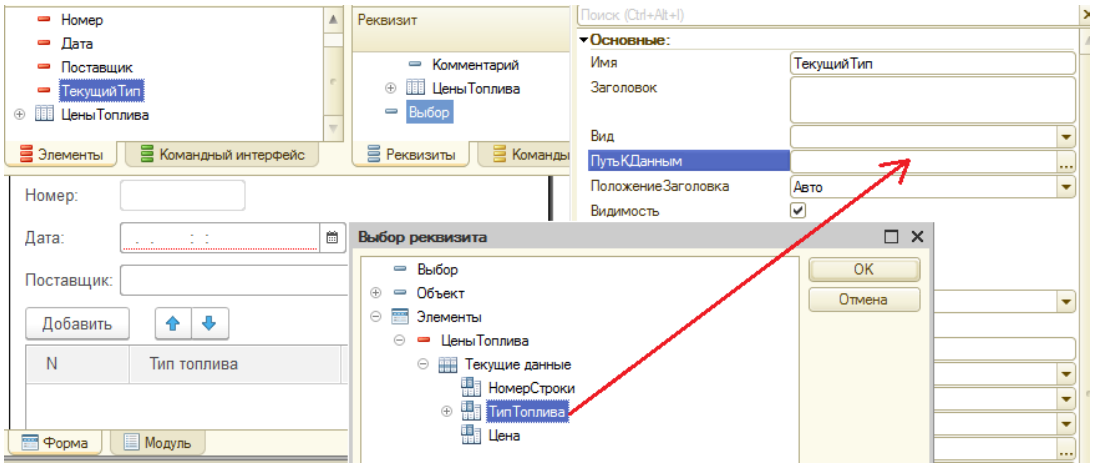


Рис. 5.3.57

Реквизит «ЦеныТоплива» с текущими данными в окне выбора реквизитов будет только тогда, когда на форме размещена таблица «ЦеныТоплива».

Установим свойства *Заголовок* и *Вид* (см. рис. 5.3.58).

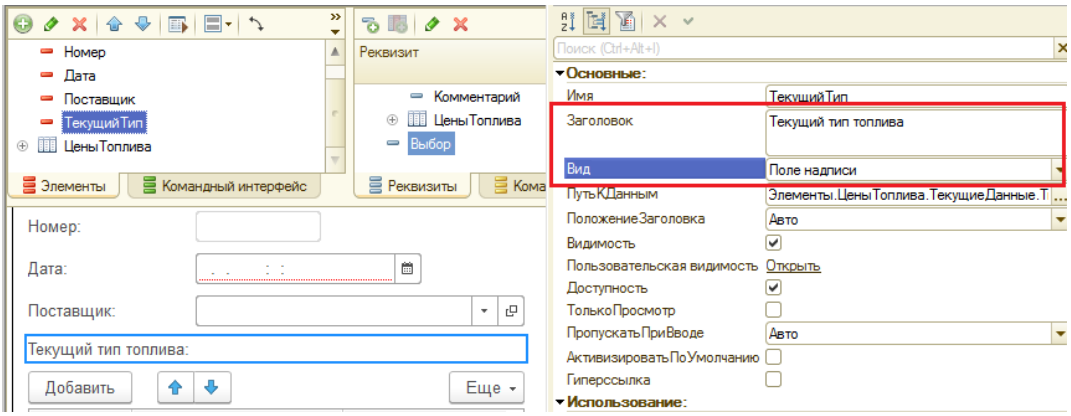


Рис. 5.3.58

Вы уже заметили, что после установки значения *Поле надписи* в свойство *Вид* отображение элемента *Поле* на форме поменялось: непосредственно самого поля ввода нет. Посмотрим, как будет работать наше поле надписи (см. рис. 5.3.59).

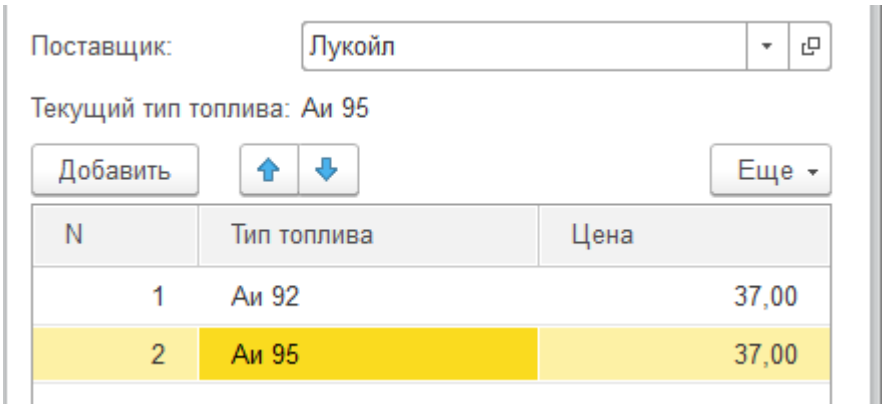


Рис. 5.3.59

Вид «Поле флажка»

Для реквизитов с типом *Булево* можно задать два вида поля – *Поле ввода* и *Поле флажка*. Как выглядит поле ввода для реквизита с типом *Булево* можно посмотреть на рис. 5.3.60.

Рис. 5.3.60

Такой вариант интерфейса применим, но не всегда удобен, гораздо практичнее использовать вид поля – *Поле флажка* (см. рис. 5.3.61).

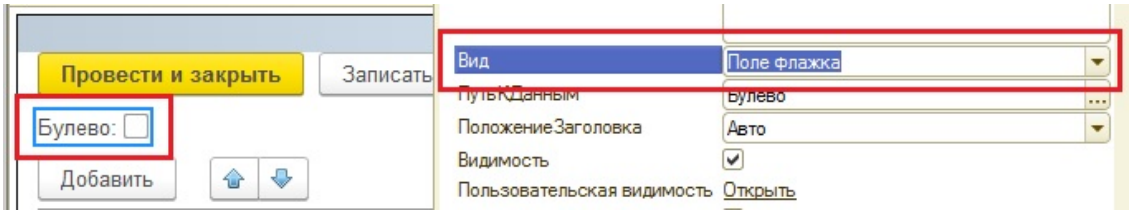


Рис. 5.3.61

В этом случае поле приобретает более компактный вид.

Но самое интересное, можно установить два вида флажка, это вид – *Флажок*, который Вы видите на рис. 5.3.61 и вид «Тумблер» (см. рис. 5.3.62).

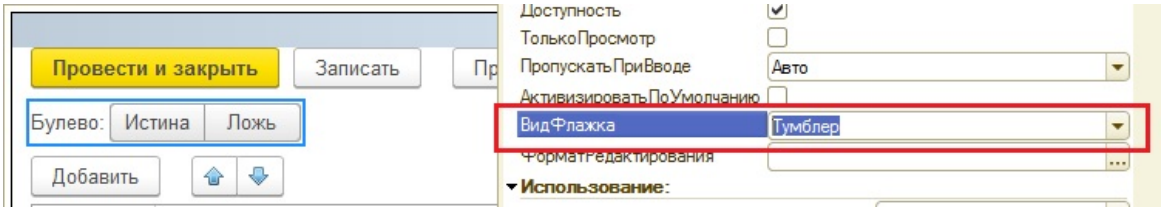


Рис. 5.3.62

Такой вид флажка более удобен для восприятия пользователя. Надписи *Истина* и *Ложь* можно заменить более понятными с помощью свойства поля флажка *ФорматРедактирования.* После нажатия на кнопку «…» этого свойства откроется конструктор редактирования форматной строки, где на закладке «Булево» можно задать представление значений *Истина* и *Ложь* (см. рис. 5.3.63).

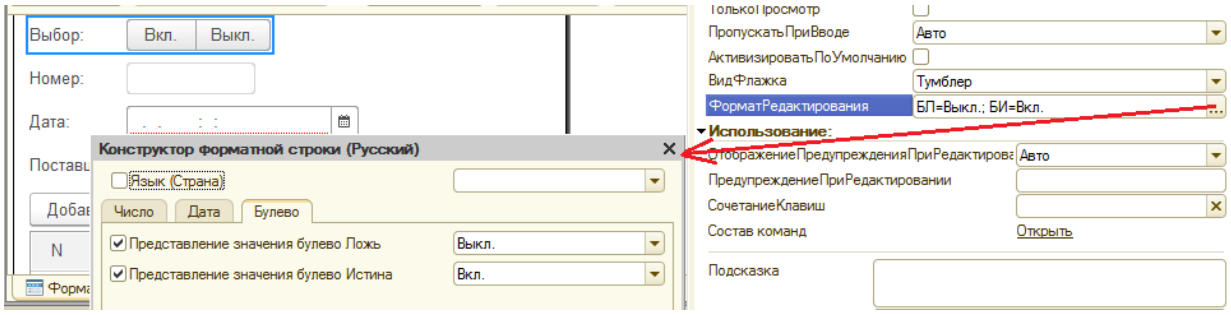


Рис. 5.3.63

На рисунке 5.3.64 показано, как будет выглядеть тумблер в пользовательском режиме.

Рис. 5.3.64

Вид «Поле переключателя»

С помощью этого поля можно устанавливать на форме различные переключатели. При этом виде поля нужно работать с уже знакомым нам свойством *СписокВыбора*, в котором необходимо указывать все возможные переключатели.

Для того чтобы понять, как работать с этим полем, создадим реквизит с типом число, «перетащим» его в элементы и установим для его поля вид *Поле переключателя* (см. рис. 5.3.65).

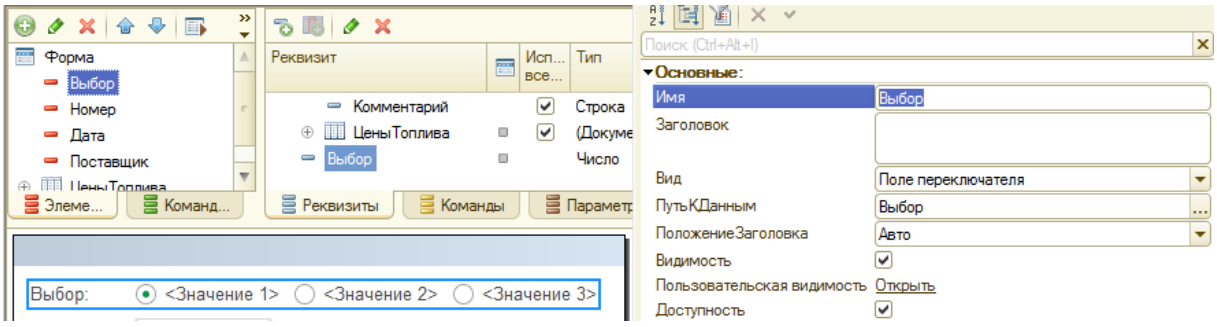


Рис. 5.3.65

Следующим шагом необходимо установить значения переключателя, делаем это с помощью уже знакомого свойства *СписокВыбора* (см. рис. 5.3.66), где в левой части таблицы необходимо указать значение, которое примет наш реквизит при установке того или иного переключателя, а в правой части отображение этого значения на форме.

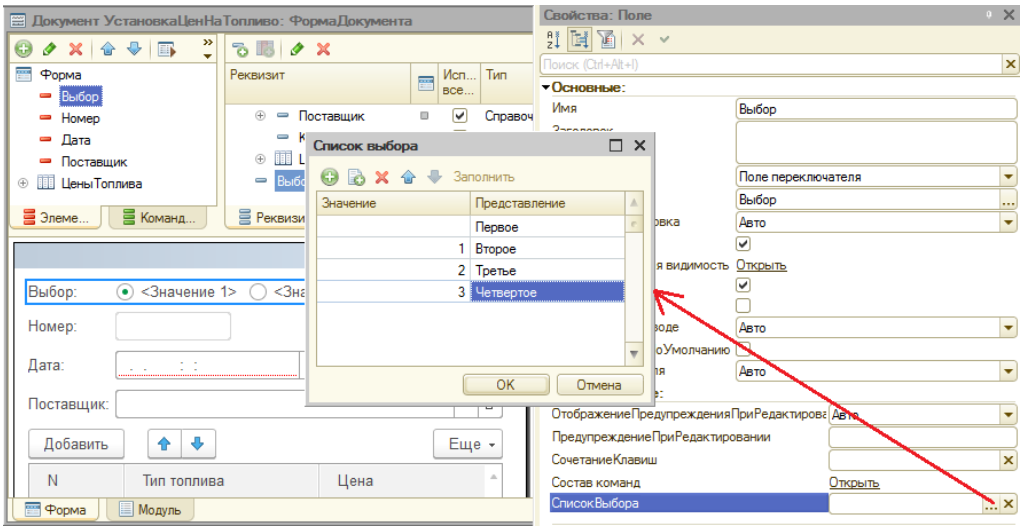


Рис. 5.3.66

Чтобы узнать программно, какое значение выбрано, достаточно прочитать реквизит формы. *Позже (см. стр. 387) мы узнаем, как это делать.*

С помощью свойства *КоличествоКолонок* можно задать различные расположение переключателя на форме (см. рис. 5.3.67 и 5.3.68). Если свойство *КоличествоКолонок* равно 0, то все переключатели идут в один ряд в зависимости от места на форме.

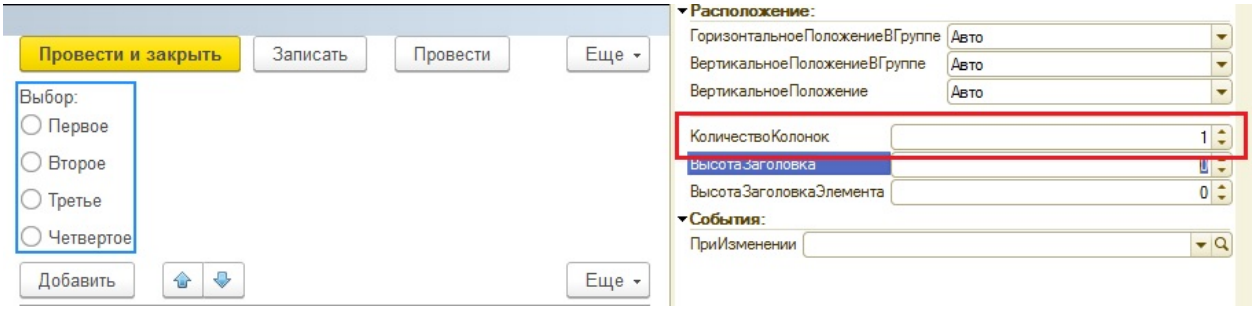


Рис. 5.3.67

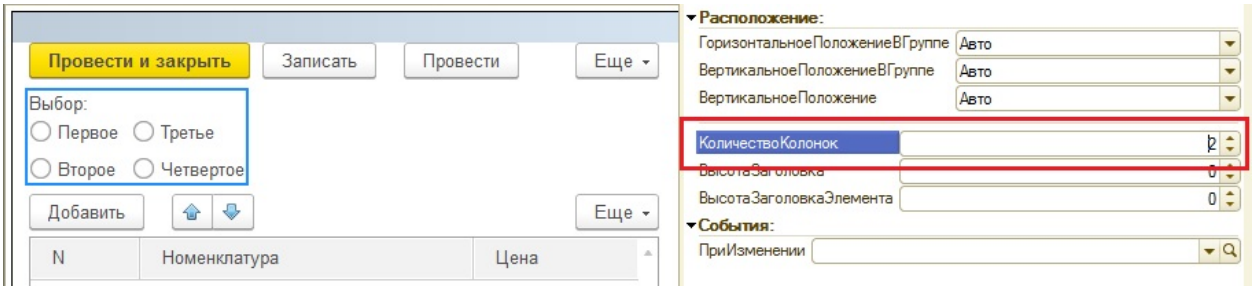


Рис. 5.3.68

И точно так же, как и в случае с флажком, у переключателя есть свойство *ВидПереключателя*, которое может принимать три значения – *Авто*, *Переключатель* и *Тумблер*.

Отображение переключателя с видом *Тумблер* на форме в конфигураторе (см. рис. 5.3.69) и в «1С:Предприятии» такое же, как и для флажка.

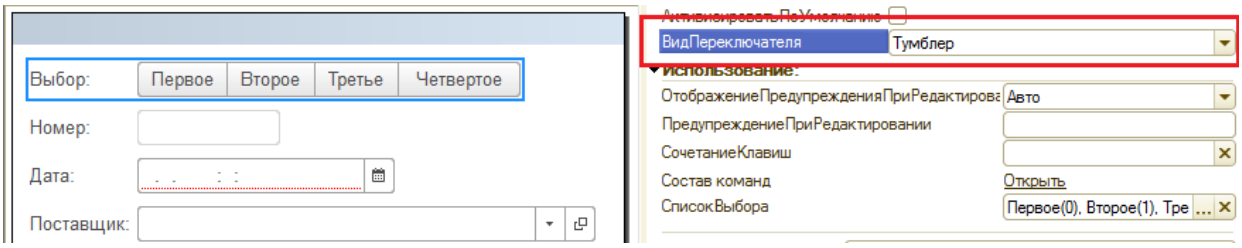


Рис. 5.3.69

На этом закончим изучать поля ввода, этой информации Вам хватит с избытком для дальнейшего изучения материала этой книги.

Кнопки

Следующий элемент формы, с которым мы познакомимся, это *кнопки*. Частично мы с ними уже сталкивались по мере изучения материала этой книги. В этом разделе познакомимся с кнопками более подробно.

Кнопку на форму можно добавить двумя способами. Первый: используя кнопку «Добавить» командной панели закладки «Элементы» (в открывшемся окне выбрать тип элемента *Кнопка*, см. рис. 5.3.70).

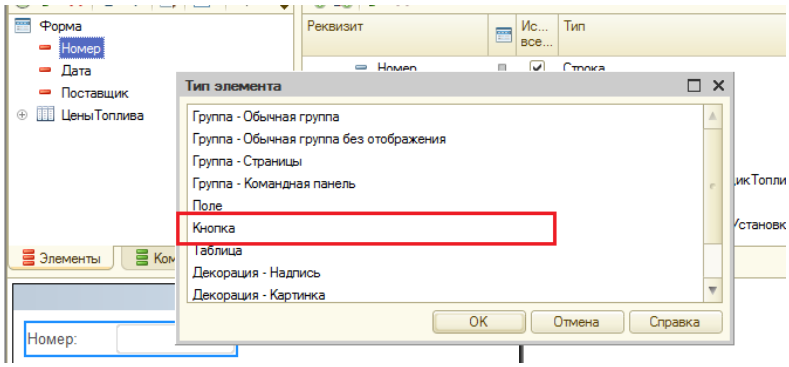


Рис. 5.3.70

Второй способ: перетащить команду на форму (см. рис. 5.3.71).

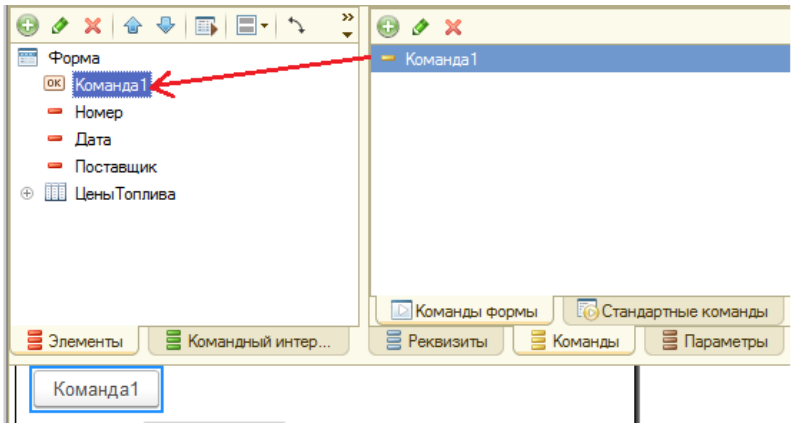


Рис. 5.3.71

Если мы добавим кнопку на форму первым способом, то она не отобразится на форме, потому что свойство *ИмяКоманды* у нашей новой кнопки пустое (см. рис. 5.3.72). Необходимо выбрать команду в этом поле, чтобы кнопка была видна на форме. После нажатия кнопки «…» свойства *ИмяКоманды* откроется окно выбора команд, посредством которого можно выбрать любую команду: от локальных команд формы до глобальных стандартных команд (см. рис. 5.3.73).

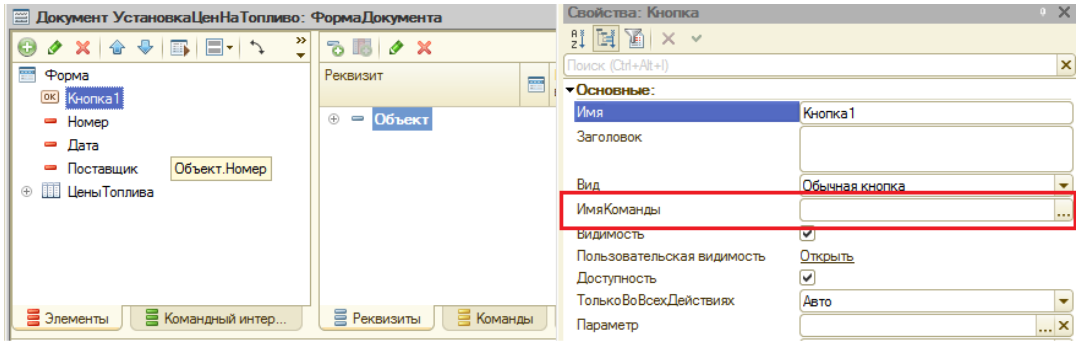


Рис. 5.3.72

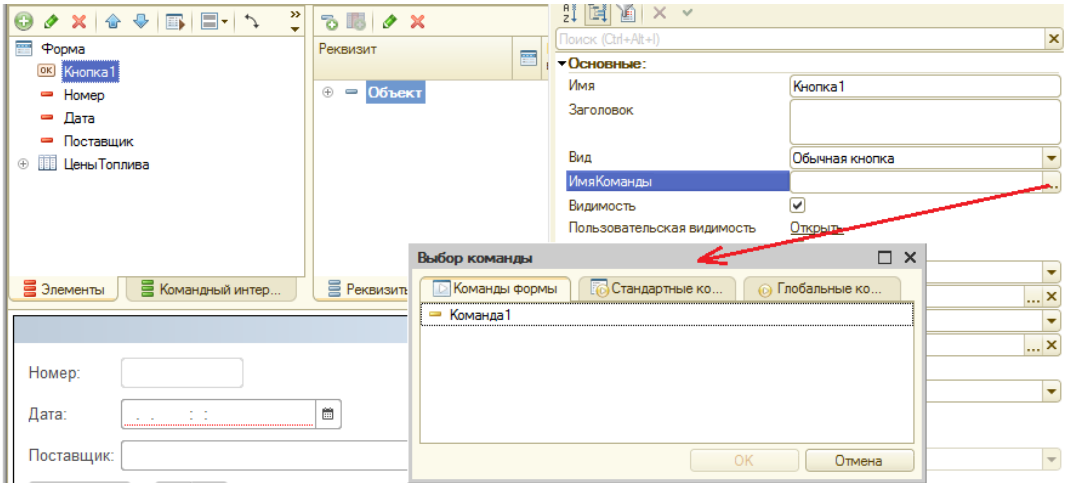


Рис. 5.3.73

Кнопка будет создана в любом случае, когда мы свяжем свойство *ИмяКоманды* с нужной командой. Причем заметьте, кнопка будет создана даже в том случае, если локальная команда формы не связана ни с каким обработчиком. Например, я создал локальную команду формы *Команда1*, у которой свойство «Действие» пустое (см. рис. 5.3.74).

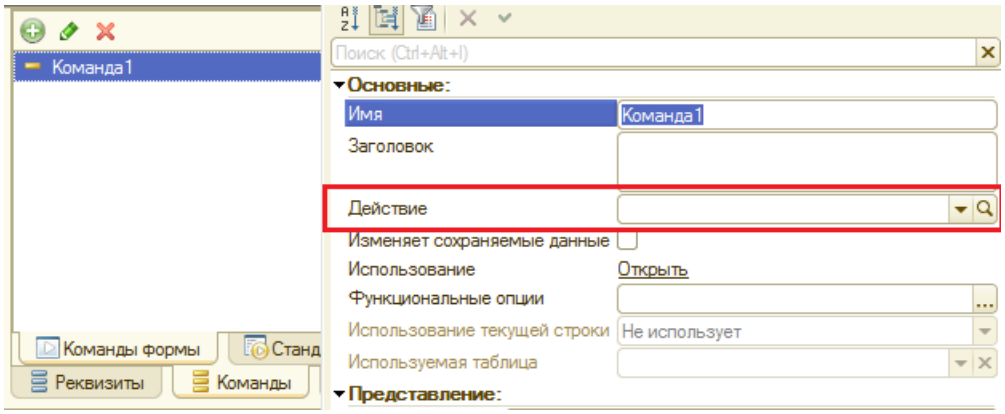


Рис. 5.3.74

Привяжем вновь созданную кнопку к локальной команде формы *Команда1*. Из рис. 5.3.75 Вы видите, что кнопка на форме отобразилась. Она будет и в пользовательском режиме. Просто при нажатии на неё не произойдет никаких действий.

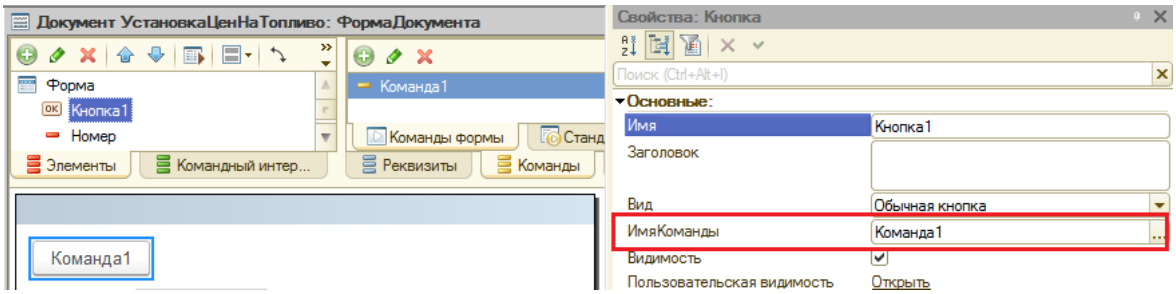


Рис. 5.3.75

Разберем основные и самые интересные свойства элемента *Кнопка*. И самое первое свойство это *Вид*. У кнопки есть два вида, это «Обычная кнопка» (см. рис. 5.3.75) и «Гиперссылка» (см. рис. 5.3.76).

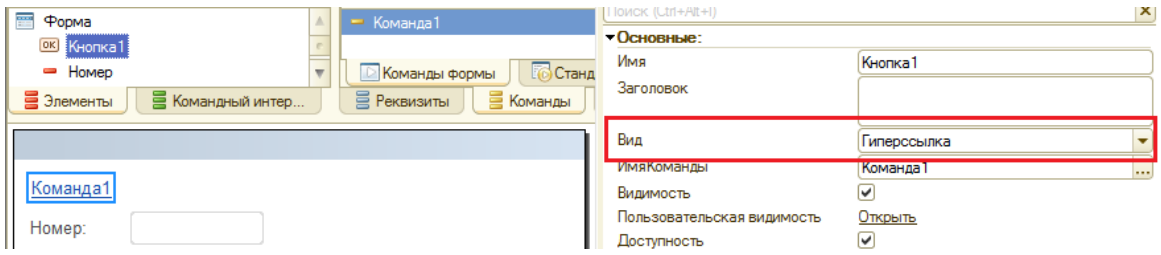


Рис. 5.3.76

Обычно на кнопке какой-нибудь текст. Этот текст берется или из имени команды, которая привязана к кнопке, или из свойства *Заголовок* команды (если оно заполнено), или из свойства *Заголовок* кнопки (если оно заполнено).

В свойстве *Заголовок* локальной команды формы *Команда1* напишем текст «Печать», надпись на кнопке поменяется (в свойство кнопки *Вид* установим обратно значение *Обычная кнопка*).

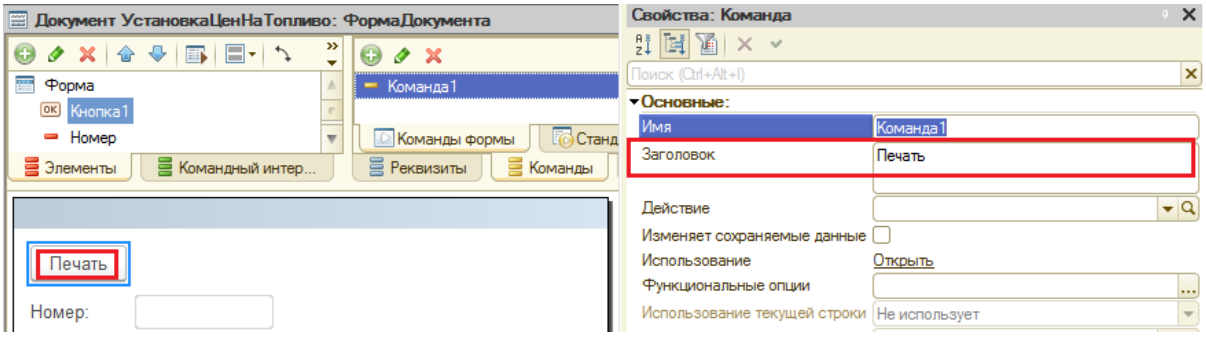


Рис. 5.3.77

Если же мы хотим, чтобы рядом с текстом была пиктограмма, то необходимо в свойство *Отображение* установить значение *Картинка и текст*, а в свойство *Картинка* выбрать нужную картинку с помощью окна выбора картинок (см. рис. 5.3.78).

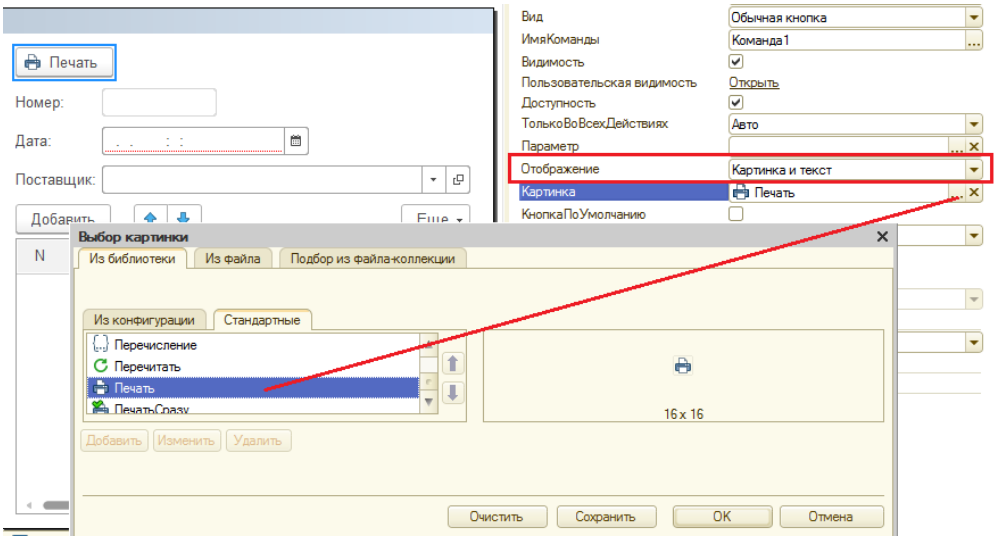


Рис. 5.3.78

И нам осталось сделать так, чтобы наша команда работала. Для этого в палитре свойств команды нажмем на пиктограмму «Лупа» свойства *Действие* и в открывшемся окне выбора вариантов оставим переключатель на варианте «Создать на клиенте» (см. рис. 5.3.79).

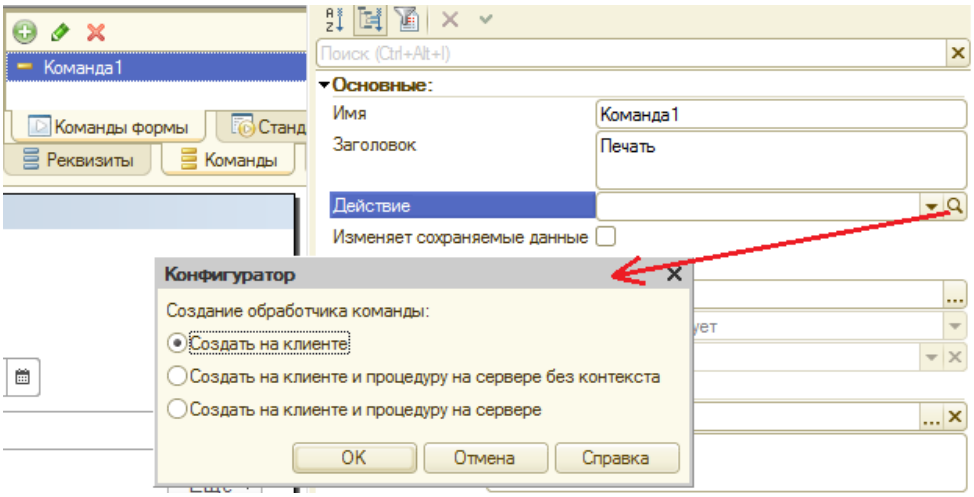


Рис. 5.3.79

В созданной автоматически процедуре на клиенте напишем код, который просто выведет сообщение.

```
&НаКлиенте
Процедура Команда1(Команда)

    Сообщить("Эмуляция печати");

КонецПроцедуры
```

Листинг 5.3.1

Сохраним конфигурацию, перезапустим «1С:Предприятие» и посмотрим, как отрабатывает наша кнопка (см. рис. 5.3.80).

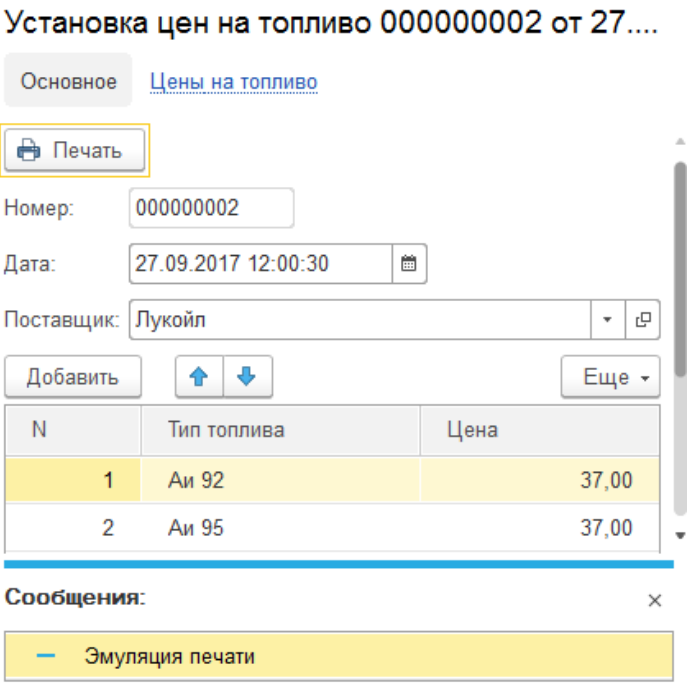


Рис. 5.3.80

Декорации

Этот элемент формы применяется при необходимости различного оформления. Например, нужно разместить какой-то не связанный с данными текст или какую-нибудь картинку. Поместить этот элемент на форму можно при помощи кнопки «Добавить» закладки *Элементы*, при нажатии на которую выйдет форма выбора элементов. В этой форме нужно выбрать или элемент «Декорация - надпись» или «Декорация - картинка» (см. рис. 5.3.81).

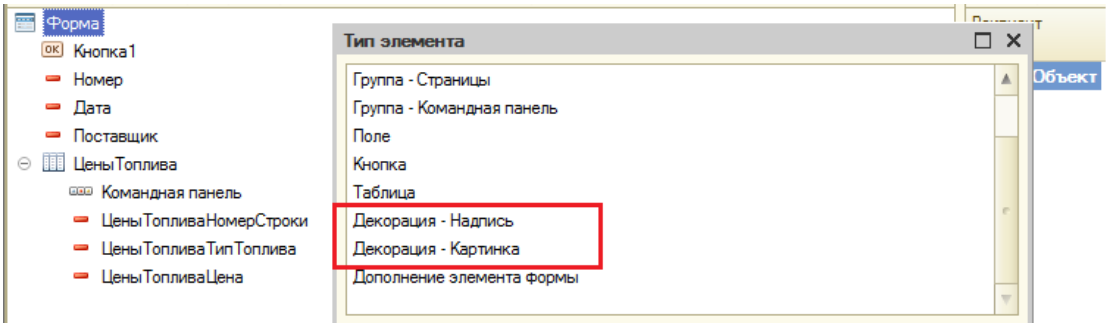


Рис. 5.3.81

Элемент *Декорация* имеет два вида, это *Надпись* и *Картинка*. Когда выбираете тип «Декорация - надпись» или «Декорация - картинка», то свойство *Вид* уже заполнено, но его можно при необходимости поменять (см. рис. 5.3.82)

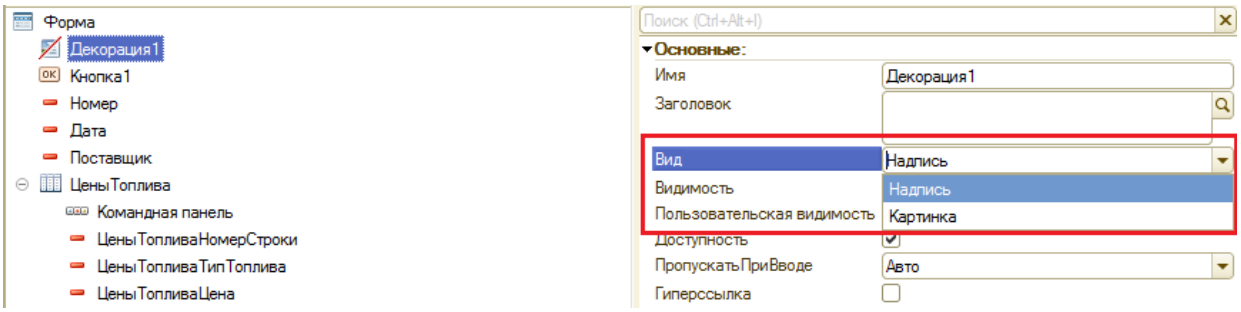


Рис. 5.3.82

Если выбран вид «Надпись», то в свойстве «Заголовок» можно написать любое сообщение. Причем у свойства есть кнопка «Лупа», нажав на которую, откроется окно ввода строки, где можно ввести как обычную строку (см. рис. 5.3.83), так и форматированную (см. рис. 5.3.84 и 5.3.85).

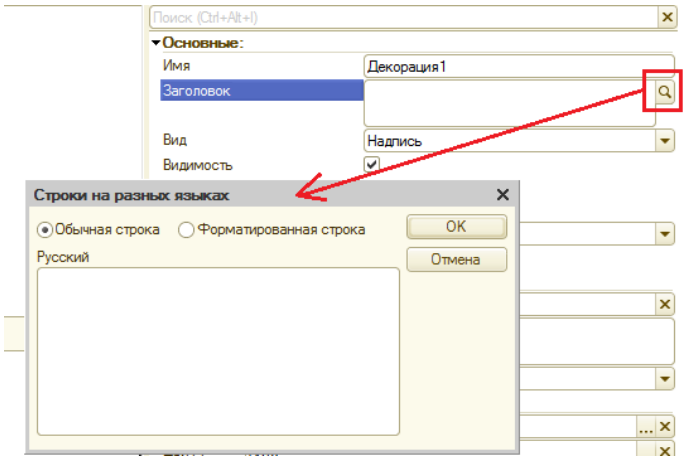


Рис. 5.3.83

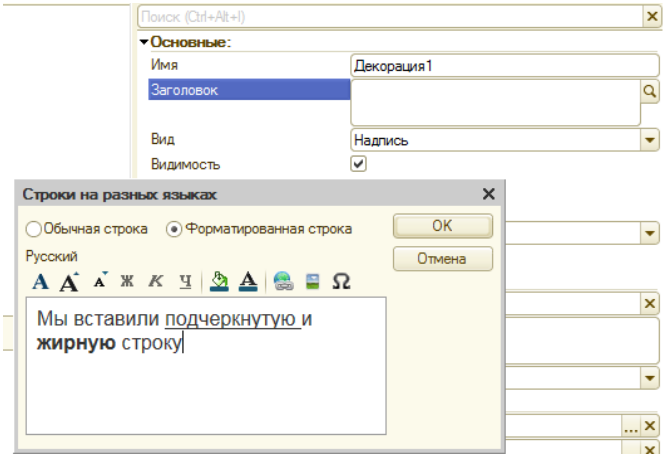


Рис. 5.3.84

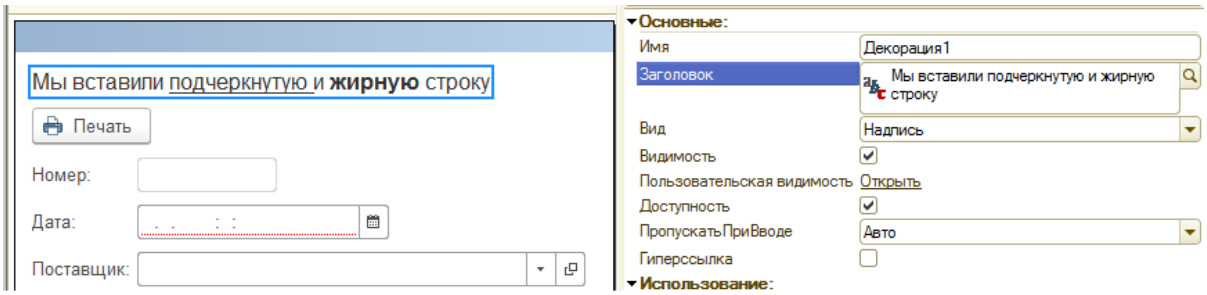


Рис. 5.3.85

И хотя мы этому еще не научились, запомните: не нужно программным способом в заголовок элемента «Декорация» передавать какие-либо значения. Используйте для этого элемент Поле с видом Поле надписи!

Разберем декорацию с видом *Картинка*. С помощью этой декорации можно размещать различные статические картинки, которые не будут меняться при открытии разных элементов справочников, документов и т.д.

В 4-й главе мы добавили общую картинку в конфигурацию (см. стр. 239) эту картинку мы и будем использовать в текущем примере.

Добавим элемент «Декорация - картинка» на форму, для того, чтобы картинка отобразилась на форме, нужно заполнить свойство *Картинка* (см. рис. 5.3.86).

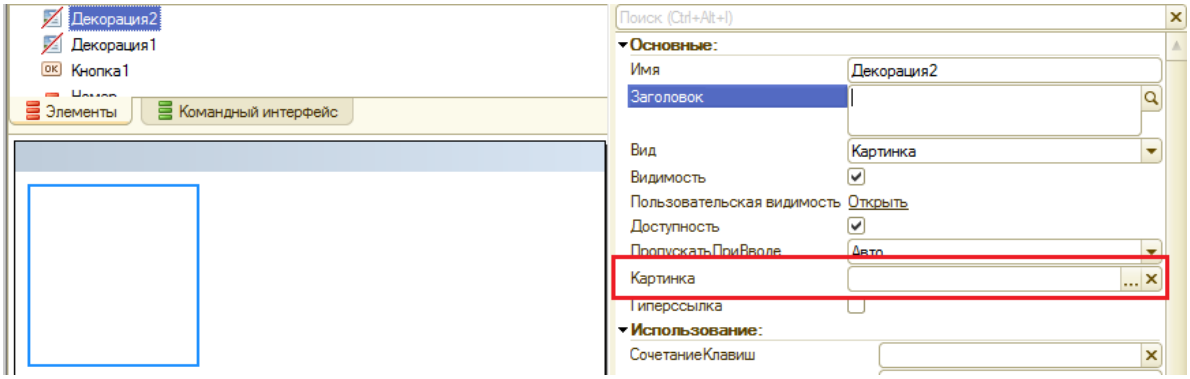


Рис. 5.3.86

После нажатия на кнопку «…» откроется форма выбора картинки (см. рис. 5.3.87), в которой пользователь может или выбрать картинку из конфигурации, или стандартную картинку, или загрузить свою собственную картину.

Не стоит с помощью элемента «Декорация» размещать картинки, которые будут храниться в базе. Для этого необходимо использовать поле ввода с видом «Поле» картинки. Как работать с картинками, хранящимися в базе, рассказывается в третей части книги «Основы разработки в 1С: Такси».

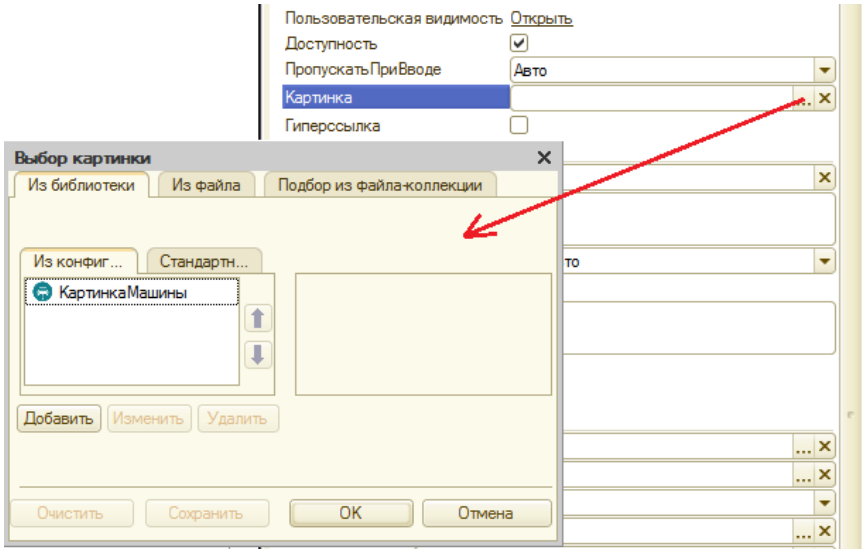


Рис. 5.3.87

Теперь Вам просто нужно выбрать или общую картинку из конфигурации (закладки «Из библиотеки» - «Из конфигурации»), или стандартную картинку, или загрузить картинку из файловой системы. Выберем картинку из конфигурации (см. рис. 5.3.88).

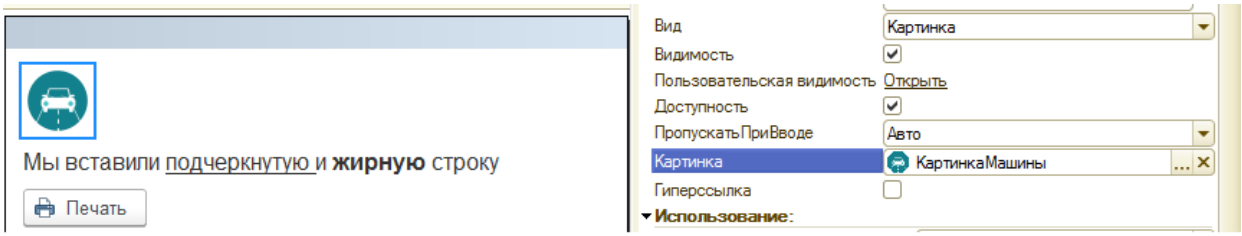


Рис. 5.3.88

Самостоятельно при помощи группы (или групп) разместите декорации следующим образом (см. рис. 5.3.89).

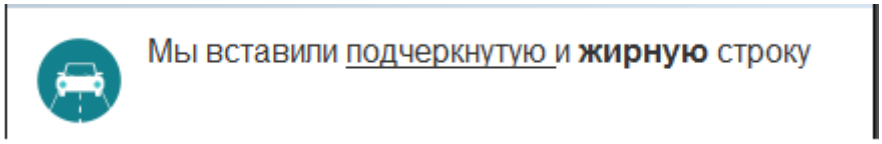


Рис. 5.3.89

На этом мы закончим изучать элементы формы. В дальнейшем при углублении материала мы изучим еще один нужный элемент – «Таблица».

Часть 4. Программирование управляемых форм

Вот мы с Вами и дошли до программирования под управляемым приложением. Имейте в виду, что приступать к изучению материалов этой главы лучше имея определенный набор знаний по программированию в среде 1С. Вы должны свободно понимать, что представляют собой переменные в среде 1С, как работают циклы и условия и чем процедура отличается от функции. Если у Вас еще нет уверенности в этих знаниях, то лучше первым делом ознакомится с предыдущими главами. Если же Вы все это знаете, то можете смело приступать к изучению материала этой части.

Клиент-серверная архитектура

Принцип работы «1С:Предприятия 8.3» под управляемым приложением имеет клиент-серверный характер. Не будем углубляться в дебри, отмечу только, что это значит, что какой-то код выполняется в контексте сервера, а какой-то код в контексте клиента. Причем, при разработке управляемого приложения, разделение контекста выполнения программного кода на клиентский и серверный стало очень критичным. Узнаем, что означает клиентский контекст выполнения кода, а что - серверный.

Для этого рассмотрим классическую трехзвенную архитектуру «1С:Предприятия». У Вас есть клиентские компьютеры, где работают все пользователи, есть кластер серверов «1С:Предприятия», где выполняются все вычисления, и есть SQL-база (СУБД), где хранятся все данные (см. рис. 5.4.1).

Рис. 5.4.1. Клиент-серверная архитектура работы «1С:Предприятия»

На клиентской машине может быть установлено какое-то приложение 1С, которое запускается под толстым или тонким клиентом, а может быть и ничего не установлено, вся работа будет осуществляться посредством веб-клиента из Интернет-браузера. Толстый или тонкий клиент, а также работу под веб-клиентом называют *клиентским приложением*. Таким образом, на клиентской машине осуществляется взаимодействие пользователя программы с самой программой 1С. Это взаимодействие осуществляется при помощи механизмов платформы 1С, а также при помощи кода, который написан разработчиками прикладного решения. Выполняясь, этот код задействует ресурсы клиентского компьютера. В этом случае говорят, что он выполняется «на клиенте». А контекст выполнения этого кода называют *клиентским*. Причем есть клиентский контекст под *тонким клиентом*, а есть под *толстым клиентом*. Не все методы, которые могут работать под *толстым* клиентом, будут работать под *тонким* или *веб-клиентом*.

Кластер серверов 1С не является чем-то единым целым, а представляет собой несколько запущенных процессов, каждый из которых является *сервером 1С*. Сервер 1С осуществляет взаимосвязь клиентского приложения с системой управления базы данных (СУБД). Также *север 1С* может исполнять определенный код, тогда говорят, что код выполняется «на сервере», а контекст выполнения этого кода – *серверный*. В случае выполнения кода *на сервере* все вычисления будут происходить на той машине, где в этот момент запущен соответствующий экземпляр сервера 1С. Как правило, это мощный и производительный компьютер.

Рис. 5.4.2. Клиентский и серверный контекст выполнения кода

При работе в обычном приложении на платформе 8.1 можно было разграничить выполнение кода на клиентский и серверный контекст, но сделать это можно было или в контексте общего модуля, или в контексте модуля объектов при помощи специальных инструкций процессору.

Форма же в обычном приложении выполнялась полностью на стороне клиентского компьютера. Причем с формы можно было что угодно делать с базой данных: создавать новые объекты, редактировать их и т.п.

В управляемом приложении все кардинально поменялось. Теперь на стороне клиентского компьютера форма должна только прорисовываться и всё. Связано это с возникновением тонкого и веб-клиента, которые осуществляют взаимодействие с серверной частью посредством сети Internet. Этот канал связи накладывает существенные ограничения на широту передаваемых данных. И если раньше мы могли на форме делать практически всё: обращаться к базе данных, создавать документы, справочники и т.п., то теперь это стало слишком дорогим удовольствием. Поэтому для нормального функционирования тонкого клиента и веб-клиента был существенно переделан механизм работы форм. Форма только прорисовывается на клиенте и всё, а все обработки данных, вычисления и т.п. должны выполняться на сервере.

При разработке управляемых приложений разработчик сам должен определять, какой код будет работать на сервере, а какой на клиенте.

Например, нам нужно на какой-то форме узнать остаток товара на какую-то дату. Для этого мы разместим на форме два реквизита, в одном будет храниться ссылка на справочник товаров, а в другом - дата, на которую нужно получить остаток, а также команду «Получить остаток». Следующим шагом нужно определить, что будет выполняться на сервере, а что на клиенте. Понятно, что обработка нажатия на кнопку (обработчик команды «Получить остаток») должна

выполняться на клиентской машине. А функция подсчета остатков – на серверной, поскольку идет обращение к базе данных. Но это еще не всё. Нам нужно вызвать из процедуры, которая выполняется на клиенте (обработка нажатия на кнопку), функцию, которая выполняется на сервере (подсчет остатков).

В момент, когда метод на клиенте обращается к процедуре или функции, которая должна выполняться на сервере, происходит вызов сервера и передача управления на сервер. Процедура или функция на сервере отрабатывают и происходит передача управления обратно на клиента, в то место, откуда была вызвана серверная процедура.

И заметьте, вызов сервера будет происходить каждый раз, когда идет вызов процедуры или функции, которая выполняется на сервере. Причем этот вызов в иных случаях может происходить быстро и быть практически незаметным (при работе по локальной сети), а может быть очень медленным (при работе по сети Internet). Поэтому разработчику при конструировании прикладной системы необходимо очень внимательно относиться к вызову процедур на сервере, чтобы не создавать «тормознутые приложения».

В то же время вызовы серверов могут отличаться. При обычном вызове сервера на сервер передается информация о данных формы, но можно вызвать сервер без передачи информации о форме. Это так называемый вызов сервера без контекста. Их отличия мы будем проходить позже.

Таким образом, разработчику прикладного решения придется решать несколько проблем. Одна из них - это определить, когда следует вызывать серверную процедуру или функцию, а когда нет. Например, нет смысла вызывать метод на сервере внутри какого-то цикла, поскольку во время итерации цикла каждый раз будет происходить вызов сервера и передача управления на сервер. В этом случае нужно вызвать один раз серверный метод перед циклом, а потом уже в цикле использовать результаты работы этого метода.

Еще одной головной болью хорошего разработчика будет определить, какой серверный метод будет вызываться обычным образом (с передачей реквизитов формы на сервер), а какой метод будет вызываться без контекста. Потому что если мы каждый раз будем отправлять на сервер большие объемы информации с формы, а потом получать их обратно, это существенно скажется на производительности приложения. Но тут стоит знать, что не все данные с формы при обычном вызове передаются на сервер, а только те, которые были изменены.

Но это еще не все, вспомним наш вызов процедуры подсчета остатков, где мы передаем на сервер какие-то объекты. В нашем случае это ссылка на какой-то элемент справочника «Номенклатура» и дата. Иногда при передаче каких-либо значений на сервер могут возникнуть определенные проблемы: не все объекты можно передать с клиента на сервер и наоборот, не всё можно передать с сервера на клиент. Например, с сервера на клиент нельзя передать объект справочника или документ, и наоборот, с клиента на сервер нельзя передать какой-нибудь элемент формы. Для того, чтобы объект можно было передавать на сервер, он должен сериализоваться. Сериализация, простыми словами, это возможность представления объекта в каком-то виде, например в виде текстовой строки, или в виде XML. Для начинающего программиста поначалу сложно будет понять, какой объект подлежит сериализации, а какой нет. Но вся информация об этом содержится в синтаксис-помощнике. Например, объект *Массив* (будем проходить в 7-й главе) подлежит сериализации.

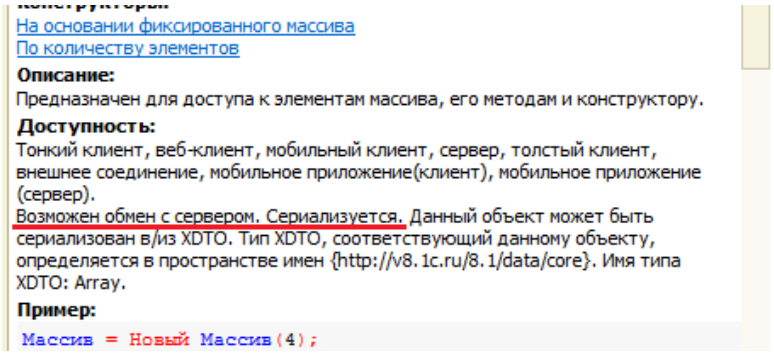


Рис. 5.4.3

И естественно, нужно помнить и знать, что при разработке управляемого приложения очень существенно ограничиваются наши действия в контексте серверного и клиентского кода. С клиентского контекста мы не можем получить прямой доступ к объектам базы 1С, у нас нет возможности ими манипулировать (добавлять, изменять, удалять и т.п.), а также не можем обращаться к свойствам этих объектов.

Например, на клиенте у нас не получится прочитать у ссылки на справочник «Номенклатура», которая хранится в реквизите формы, реквизиты и свойства этой ссылки. При попытке выполнения в клиентском контексте кода вот такого вида: НаименованиеНоменклатуры = Номенклатура.Наименование;, произойдет ошибка. В то же время в серверном контексте нет смысла в выполнении кода, который обрабатывает различные события на форме (нажатие на кнопку и т.п.).

Также стоит учитывать вид клиентского контекста. Какие-то объекты доступны в тонком, толстом и веб-клиенте (как тот же «Массив»). А какие-то только в толстом клиенте, например *ТаблицаЗначений* (изучим в 8-й главе). Определить, что выполняется в каком виде клиента можно в справочной информации.

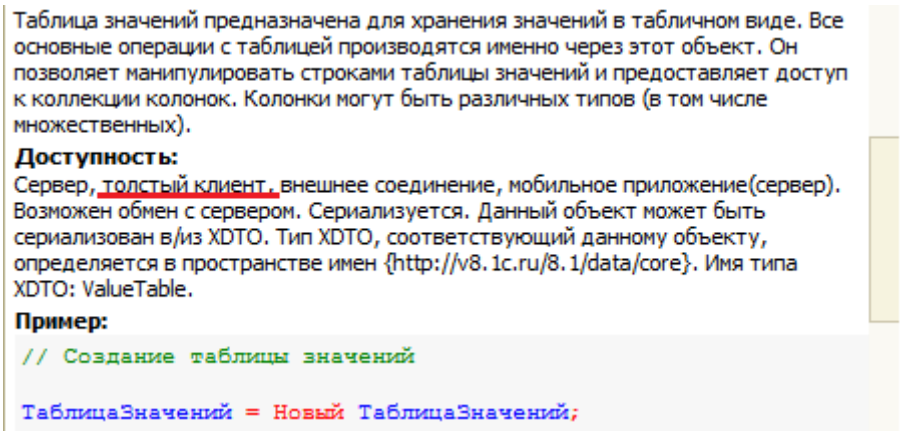


Рис. 5.4.4

Т.е. та же *ТаблицаЗначений* вполне работает в клиентском контексте, но только в режиме *толстого клиента*.

Что же из себя представляет форма в клиент-серверном понимании? Форма - это программный объект, который создается на сервере согласно настройкам, сделанным в конфигураторе (если формы нет в конфигураторе, то она генерируется автоматически платформой), а потом выводится в клиентском приложении. Таким образом, форма

одновременно существует и на сервере, и на клиенте! А как следствие у формы есть и серверный, и клиентский контекст выполнения кода.

При разработке форм под управляемым приложением разработчик должен сам указывать, какой код будет выполняться на сервере, а какой на клиенте. Делается это при помощи директив компиляции.

Директивы компиляции

При разработке формы в управляемом приложении программисту нужно явно указывать, какой код выполняется в клиентском контексте. А какой - в серверном. Осуществляется это при помощи директив компиляции, которые предопределенны в платформе 1С 8. Директиву компиляции следует указывать перед каждой функцией или процедурой (а также переменными) в модуле формы, в общем модуле или в модуле команды. Все директивы компиляции начинаются с символа амперсанд - **&**. И от них зависит, в каком контексте будет выполняться код в процедуре или функции. Всего их пять:

- &НаКлиенте
- &НаСервере
- &НаСервереБезКонтекста
- &НаКлиентеНаСервереБезКонтекста
- &НаКлиентеНаСервере

В этой книге мы изучим только первые три директивы.

&НаКлиенте – когда процедура или функция предварена этой директивой, то данный метод будет выполняться в клиентском контексте. Это значит, что он будет выполняться на той машине, где в данный момент функционирует *клиентское приложение*. Из такой процедуры или функции будут доступны все данные формы, а также из неё можно вызывать любую процедуру или функцию текущего модуля. Под этой директивой будет доступен весь клиентский контекст формы, т.е. мы можем вызывать любые функции под директивой *&НаКлиенте*.

Все обработчики команд всегда выполняются на клиенте. Для того, чтобы проверить это, разместите на любой форме (я для этой цели создал новую обработку) произвольную команду (я её назвал «Нажатие кнопки») и нажмите на кнопку «Лупа» свойства «Действие» этой команды (см. рис. 5.4.5).

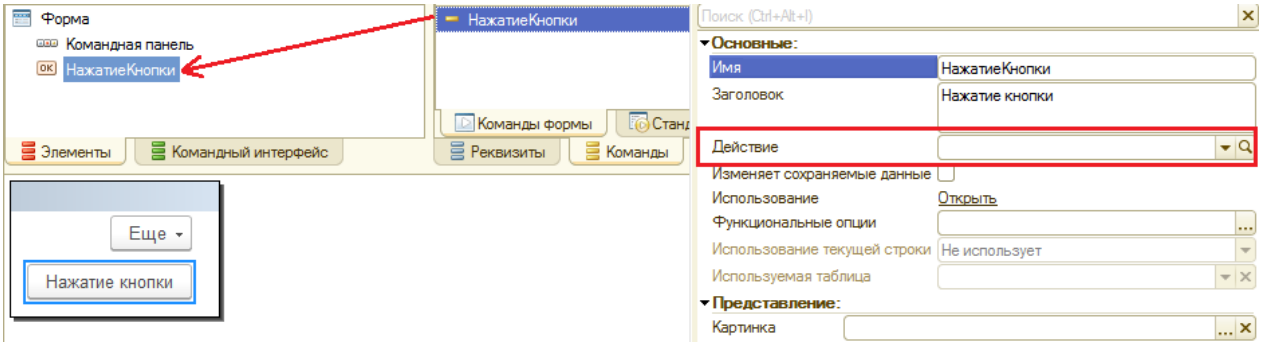


Рис. 5.4.5

После нажатия на кнопку «Лупа» свойства «Действие» команды, платформа предложит Вам выбрать, где создать обработчик события (см. рис. 5.4.6). Заметьте, во всех трех вариантах создания процедуры обработчика присутствует процедура, которая будет выполняться *на клиенте*. Просто в случаях выбора «Создать процедуру на клиенте и на сервере» и «Создать процедуру на клиенте и на сервере без контекста» будет создана процедура *на клиенте*, которая будет вызывать процедуру или на сервере, или на сервере без контекста. Ниже мы разберем этот момент.

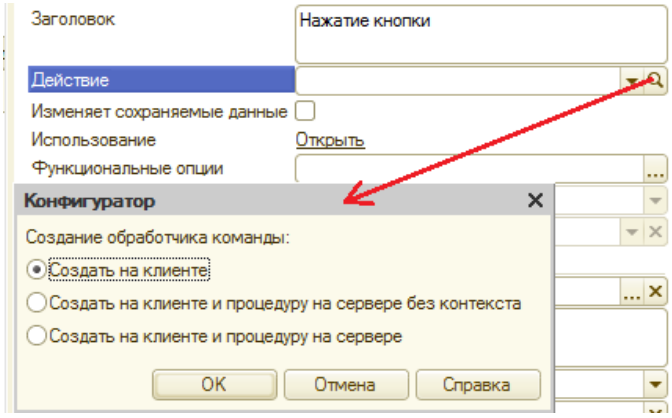


Рис. 5.4.6

Мы выберем пункт «Создать на клиенте», и после нажатия кнопки «ОК» в модуле формы автоматически появится код, как на листинге 5.4.1.

```
&НаКлиенте
Процедура НажатиеКнопки(Команда)
   // Вставить содержимое обработчика.
КонецПроцедуры
```

Листинг 5.4.1

Имейте в виду, все обработчики элементов формы должны всегда выполняться под этой директивой (см. листинг 5.4.1). Другими словами обработка событий всех элементов формы имеет *клиентский контекст*.

Любая процедура или функция, которая написана под директивой *&НаКлиенте*, может использовать все методы этой формы под директивами *&НаКлиенте*, *&НаСервере*, *&НаСервереБезКонтекста* и *&НаКлиентеНаСервереБезКонтекста*. Поэтому, код, показанный на листинге 5.4.2, будет успешно выполняться и не вызовет ошибки.

```
&НаКлиенте
Процедура НажатиеКнопки(Команда)

   ПроцедураВыполняемаяНаКлиенте();

КонецПроцедуры

&НаКлиенте
Процедура ПроцедураВыполняемаяНаКлиенте()
   //....
КонецПроцедуры
```

Листинг 5.4.2

*Директива **&НаКлиенте** - единственная директива, из-под которой доступен клиентский контекст формы.*

&НаСервере - код процедуры или функции под этой директивой будет выполняться в серверном контексте, т.е. на машине, где запущен *сервер 1С*. Все данные формы будут передаваться на сервер, а потом обратно на форму, после завершения выполнения метода. Из процедуры под этой директивой будет доступен весь *серверный контекст* формы, т.е. все процедуры или функции, которые выполняются на сервере, можно вызвать из метода предваренного этой директивой. Но в то же время *клиентский контекст формы* будет недоступен.

Проверим работу данной директивы, для этого удалите все обработчики команды «Нажатие кнопки», очистите свойство «Действие», заново на него нажмите и выберите пункт «Создать на клиенте и процедуру на сервере» (см. рис. 5.4.7).

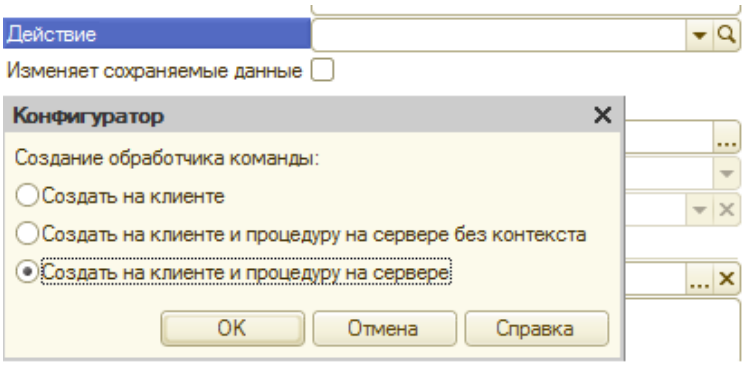


Рис. 5.4.7

После этих действий автоматически будут созданы две процедуры. Одна *на сервере*, а другая *на клиенте* (см. листинг 5.4.3).

```
&НаСервере
Процедура НажатиеКнопкиНаСервере()
   // Вставить содержимое обработчика.
КонецПроцедуры

&НаКлиенте
Процедура НажатиеКнопки(Команда)
   НажатиеКнопкиНаСервере();
КонецПроцедуры
```

Листинг 5.4.3

В этом листинге процедура клиентского контекста *НажатиеКнопки* вызывает процедуру серверного контекста *НажатиеКнопкиНаСервере*. В методе, который выполняется под директивой *&НаСервере*, нам доступны все данные формы: реквизиты, параметры и т.д. Но нам уже не доступен *клиентский контекст* формы. Если мы напишем еще одну процедуру *на клиенте* (см. листинг 5.4.4) и попытаемся её вызвать из новой серверной процедуры *НажатиеКнопкиНаСервере*, то при проверке синтаксиса будет сгенерирована ошибка (см. рис. 5.4.8).

```
&НаСервере
Процедура НажатиеКнопкиНаСервере()
   ВыполнениеНаКлиенте()
КонецПроцедуры

&НаКлиенте
Процедура НажатиеКнопки(Команда)
   НажатиеКнопкиНаСервере();
КонецПроцедуры

&НаКлиенте
Процедура ВыполнениеНаКлиенте()

КонецПроцедуры
```

Листинг 5.4.4

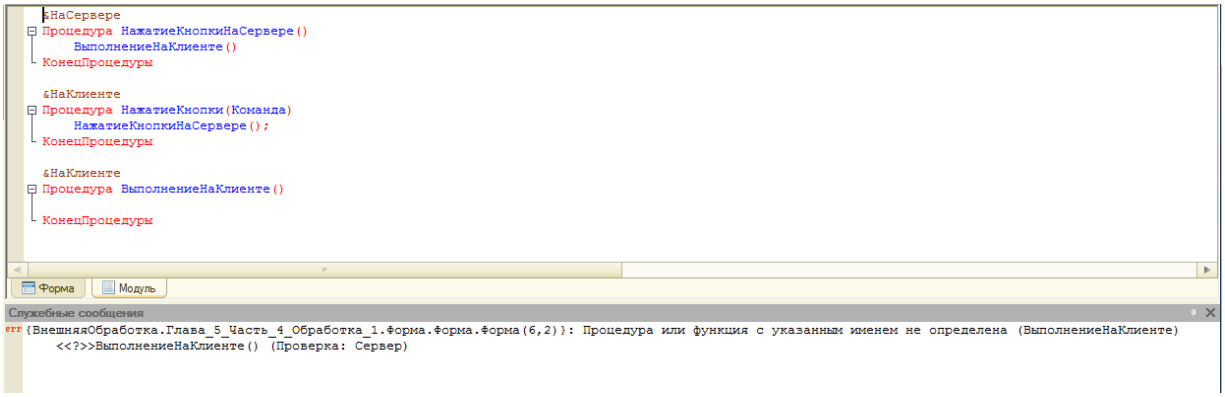


Рис. 5.4.8

Платформа просто не видит эту процедуру, потому что из серверного контекста клиентский контекст не доступен, в отличие от обратного.

Теперь проверим утверждение про разное чтение объектов при клиентском вызове методов (тонкого клиента) и при серверном. Для этого создадим новый реквизит «Автомобиль» с соответствующим типом (ссылка на справочник *Автомобили*) и разместим его на форме (см. рис. 5.4.9). Поставим точки останова в процедурах *НажатиеКнопки* и *НажатиеКнопкиНаСервере* (см. рис. 5.4.10) и начнем отладку тонкого клиента (см. рис. 5.4.11).

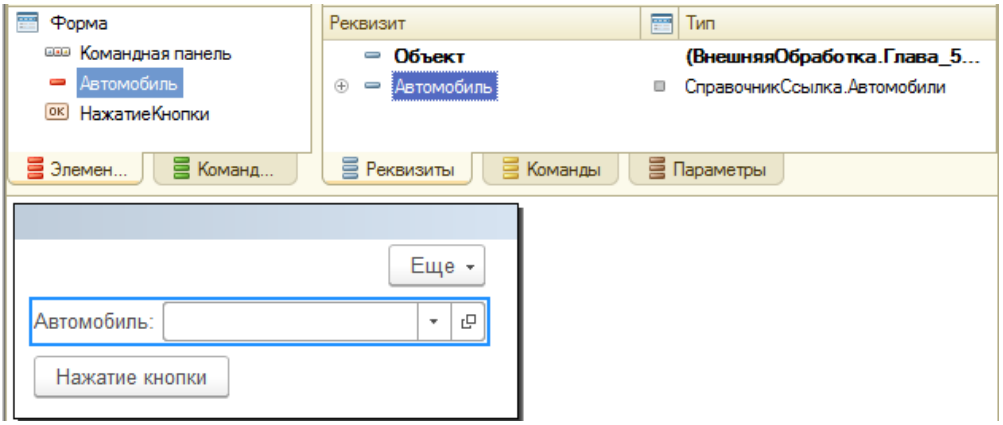


Рис. 5.4.9

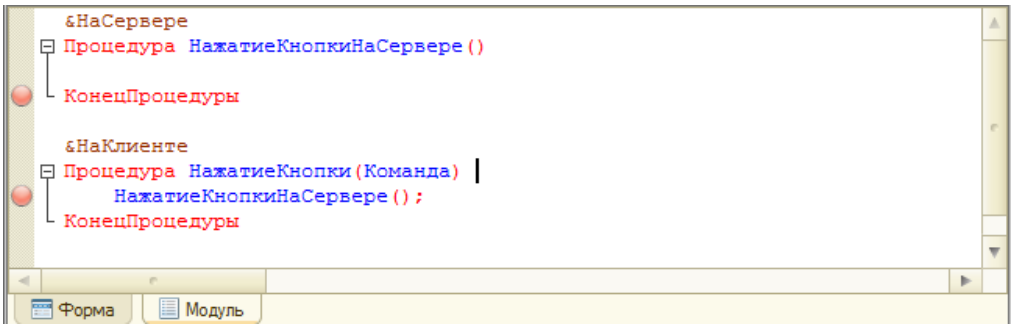


Рис. 5.4.10

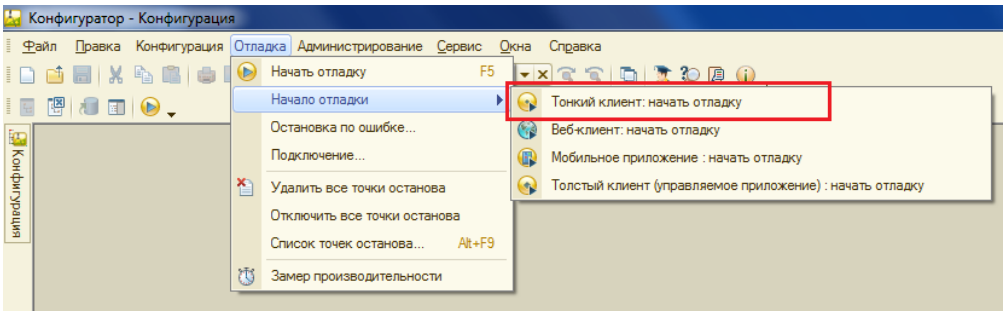


Рис. 5.4.11

В открывшейся обработке выберем какой-нибудь автомобиль (см. рис. 5.4.12) и нажмем на кнопку.

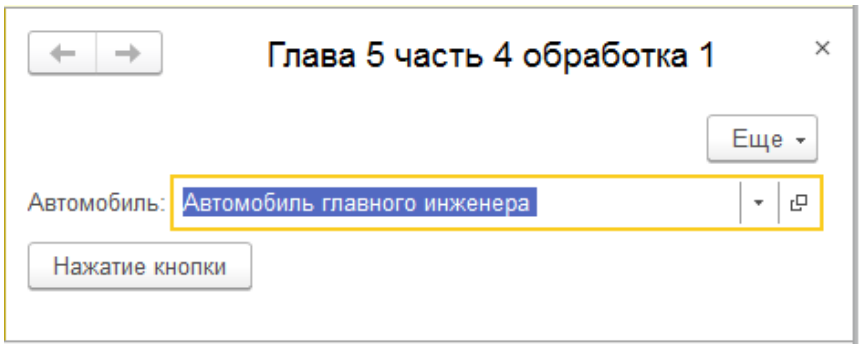


Рис. 5.4.12

Первой сработает точка останова в процедуре *НажатиеКнопки.* Попробуем в табло прочитать значение реквизита «Автомобиль» (см. рис. 5.4.13).

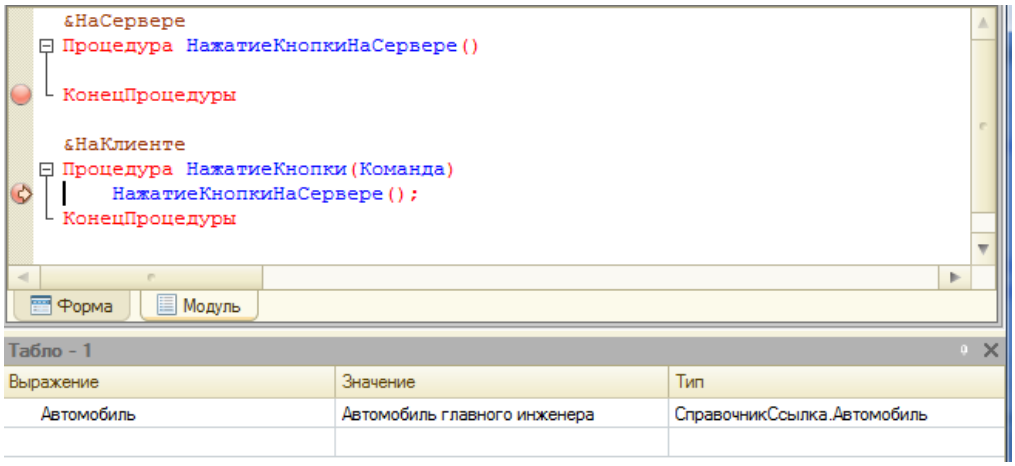


Рис. 5.4.13

Как понятно из рисунка 5.4.13, мы видим только само значение реквизита, но не можем прочитать сам объект (значения его реквизитов).

А если мы перейдем в процедуру *НажатиеКнопкиНаСервере*, то картинка в табло будет другой (см. рис. 5.4.14).

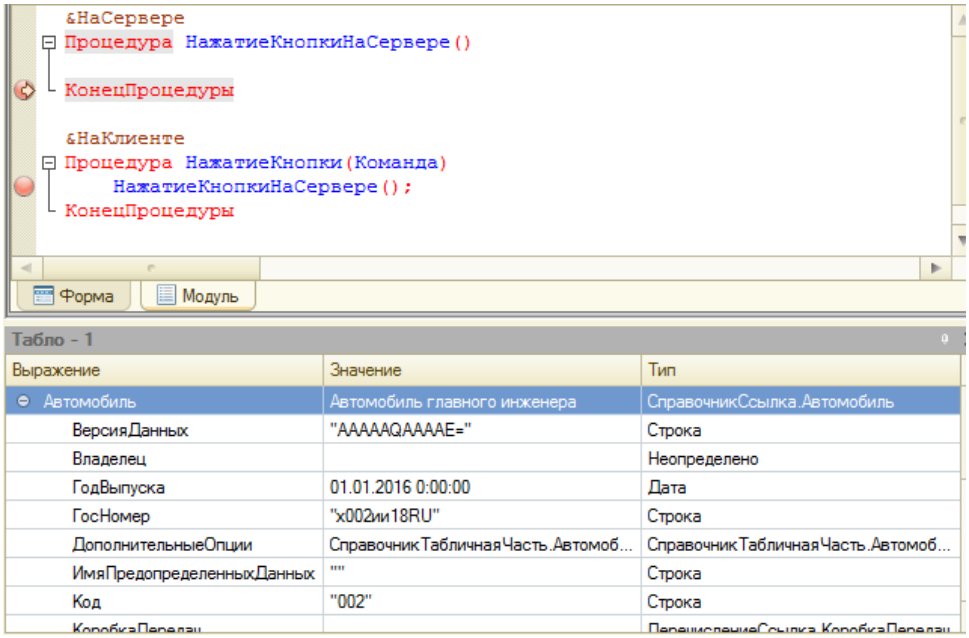


Рис. 5.4.14

Т.е. в серверном контексте можем получить подробнейшую информацию об объекте ссылочного типа, в отличие от клиентского контекста тонкого клиента.

В то же время к свойствам элемента формы *Автомобиль* (об элементах формы ниже) можно обращаться как из процедуры *НажатиеКнопки*, так и из процедуры *НажатиеКнопкиНаСервере*, т.е. из клиентского, и из серверного контекста формы (см. рис. 5.4.15 и 5.4.16).

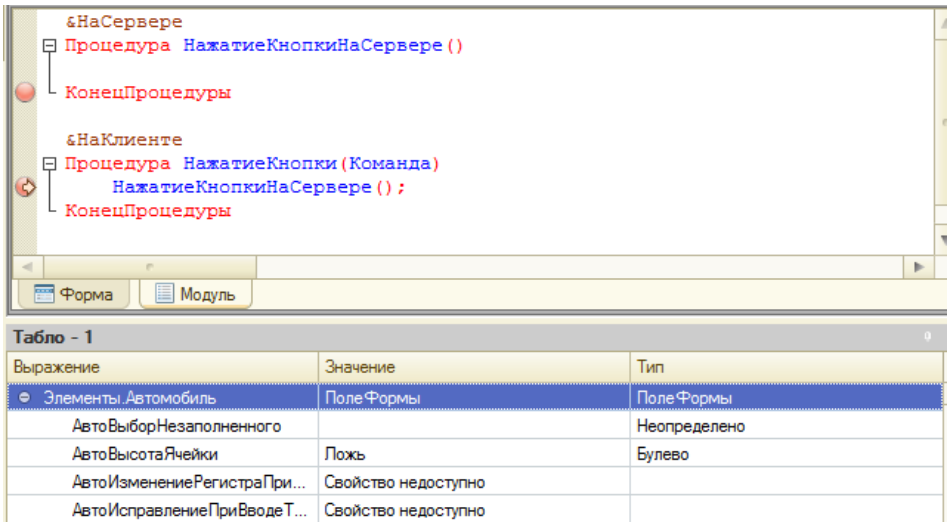


Рис. 5.4.15

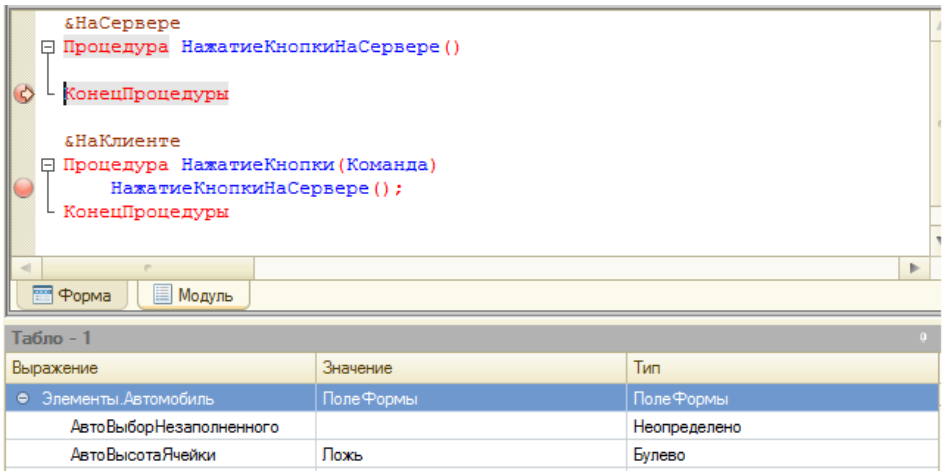


Рис. 5.4.16

Но не рекомендую Вам работать с элементами формы на серверном контексте, а использовать для этого клиентские методы, поскольку такое неоптимальное использование вызова серверных методов может сказаться на производительности приложения в целом.

Ещё раз подытожу, при вызове метода под директивой *&НаСервере* доступны все данные формы, доступен серверный контекст формы, а также можно вызывать внеконтекстные серверные процедуры и функции (с директивой *&НаСервереБезКонтекста*). В то же время нужно иметь в виду, что при вызове серверного метода происходит передача всех данных формы на сервер, а потом обратно, что может сказаться на производительности приложения, особенно запускаемого под тонким клиентом.

&НаСервереБезКонтекста - код процедуры или функции под этой директивой, как и в случае с директивой *&НаСервере,* будет выполняться в серверном контексте, т.е. на машине, где запущен *сервер 1С*. Но в отличие от серверного вызова методов, при *внеконтекстном* серверном вызове, данные формы **не будут** передаваться на сервер. Из процедуры или функции под этой директивой будет отсутствовать доступ к *серверному контексту* формы, т.е. из процедуры или функции под директивой *&НаСервереБезКонтекста* нельзя вызвать процедуры или функции под директивой *&НаСервере*.

Сейчас мы проверим, как работает безконтекстный вызов процедуры или функции. Для этого очистим на форме все что есть, а также очистим обработчик нажатия кнопки «Нажатие кнопки», и создадим новый обработчик (см. рис. 5.4.17).

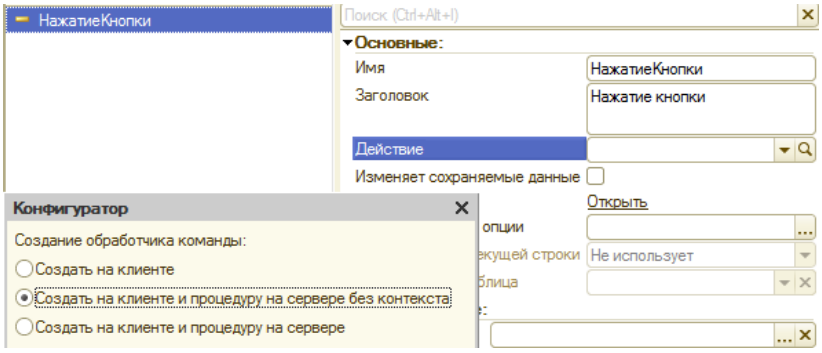


Рис. 5.4.17

После нажатия кнопки «ОК», в модуле формы будут автоматически созданы две процедуры, одна будет под директивой *&НаКлиенте*, а другая - *&НаСервереБезКонтекста* (см. листинг 5.4.5)

```
&НаСервереБезКонтекста
Процедура НажатиеКнопкиНаСервере()
   // Вставить содержимое обработчика.
КонецПроцедуры

&НаКлиенте
Процедура НажатиеКнопки(Команда)
   НажатиеКнопкиНаСервере();
КонецПроцедуры
```

Листинг 5.4.5

Как уже было сказано, из внеконтекстной процедуры не доступен *серверный контекст* формы (как и клиентский, кстати). Проверим это, попытаемся использовать в нашей новой процедуре *НажатиеКнопкиСервер* метод под директивой *&НаСервере*, т.е. процедуру, которая выполняется в серверном контексте формы (см. листинг 5.4.6).

```
&НаСервереБезКонтекста
Процедура НажатиеКнопкиНаСервере()
   ВыполенениеНаСервере()
КонецПроцедуры

&НаКлиенте
Процедура НажатиеКнопки(Команда)
   НажатиеКнопкиНаСервере();
КонецПроцедуры

&НаСервере
Процедура ВыполнениеНаСервере()
КонецПроцедуры
```

Листинг 5.4.6

Если мы выполним синтаксис-проверку модуля, то платформа выдаст ошибки (см. рис. 5.4.18). Из методов под директивой *&НаСервереБезКонтекста* можно вызвать только методы такой же директивой, или с директивой *&НаКлиентеНаСервереБезКонтекста.*

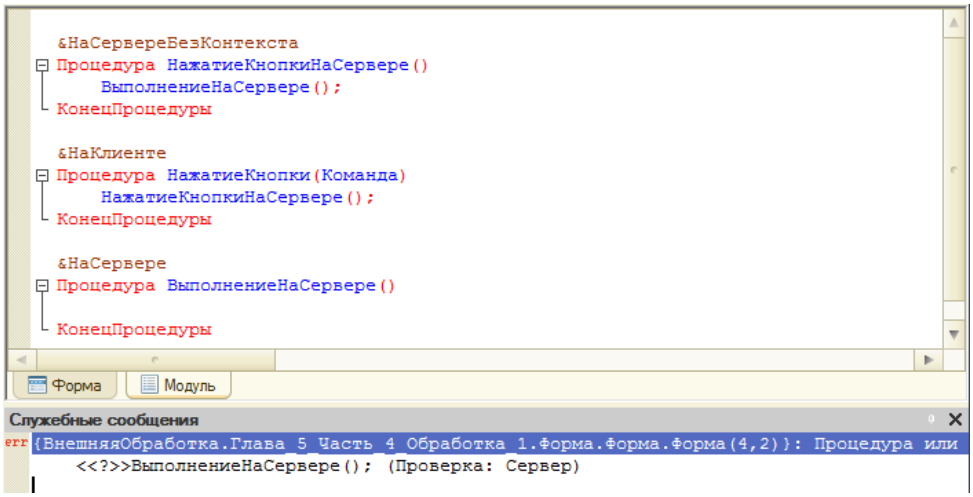


Рис. 5.4.18

Также в момент вызова внеконтекстного метода не происходит передача данных формы на сервер и обратно. И как следствие, из внеконтексной процедуры нет доступа ни к реквизитам формы, ни к элементам и т.п. (см. рис. 5.4.19). В то же время, поскольку передача данных формы при вызове внеконтекстного серверного метода не происходит, то вызов такого метода осуществляется гораздо быстрее. Поэтому рекомендуется использовать вызов безконтекстных серверных методов для оптимизации работы приложения.

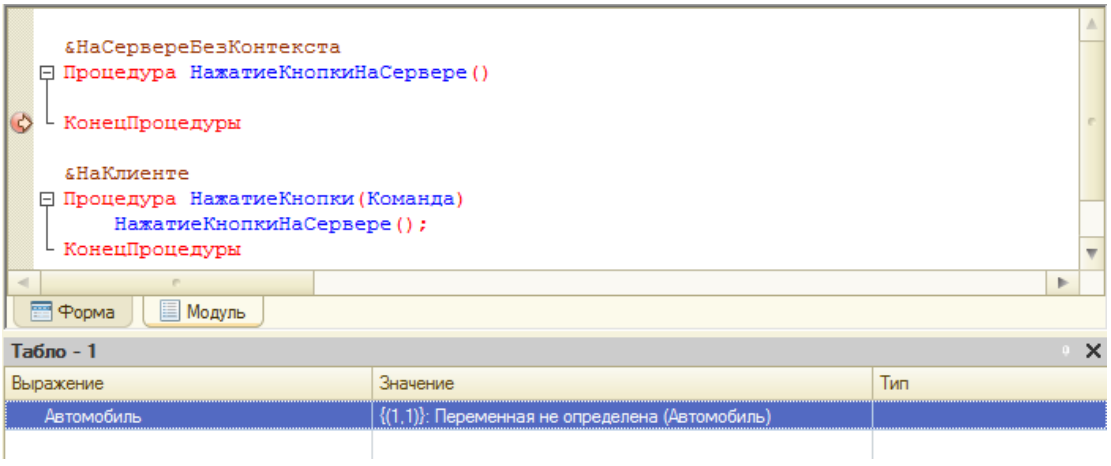


Рис. 5.4.19

Причем внеконтекстный серверный метод можно спокойно вызывать, как и из клиентского контекста (см. листинг 5.4.7), так и из серверного контекста (см. листинг 5.4.7).

```
&НаСервереБезКонтекста
Процедура НажатиеКнопкиНаСервере()

КонецПроцедуры
```

```
&НаСервере
Процедура ВызовНаСервере()

   НажатиеКнопкиНаСервере();

КонецПроцедуры

&НаКлиенте
Процедура НажатиеКнопки(Команда)
   НажатиеКнопкиНаСервере();
КонецПроцедуры
```

Листинг 5.4.7

На этом мы закончим изучать директивы компиляции, при помощи этих трех директив можно выполнить большинство работ с управляемыми формами.

Реквизиты и параметры формы

Разберем основные моменты работы с реквизитами и параметрами формы. Реквизиты мы уже проходили в предыдущей части, в этой углубим наши знания. Напомню, что реквизиты формы задаются в закладке «Реквизиты» редактора формы (см. рис. 5.4.20). А параметры формы - в закладке «Параметры» (см. рис. 5.4.21).

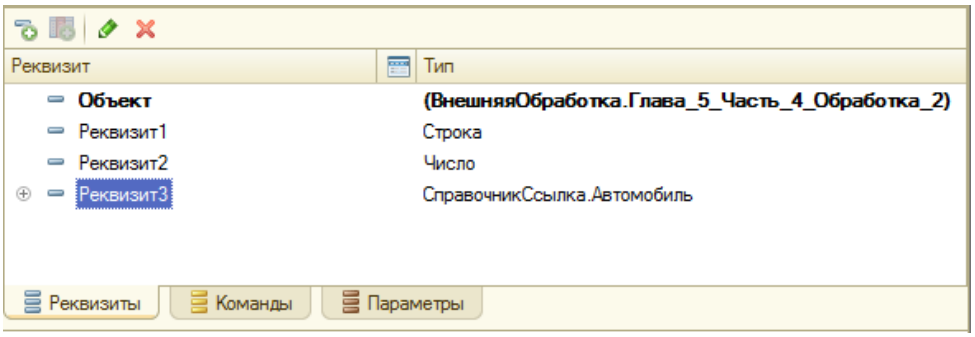


Рис. 5.4.20

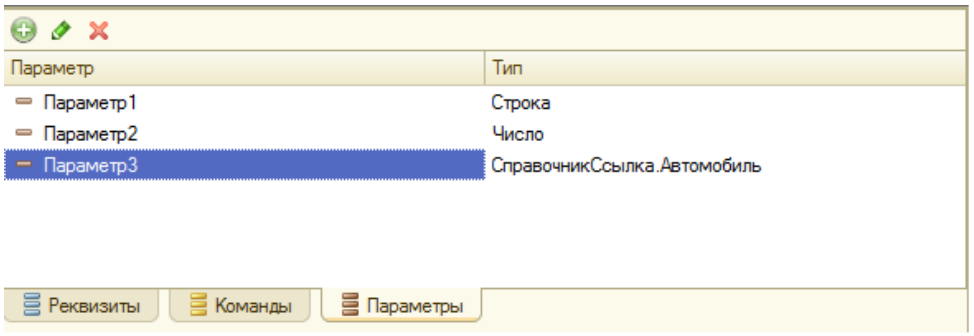


Рис. 5.4.21

Реквизиты формы предназначены для хранения данных, с которыми работает форма. Они образуются в момент создания формы и уничтожаются после её закрытия. К реквизитам можно обратиться и в *серверном* контексте формы, и в *клиентском*.

На форме можно создать элемент, который будет связан с реквизитом посредством свойства «ПутьКДанным» (см. рис. 5.4.22). Посредством этого элемента можно менять значения реквизита формы. Но совсем не обязательно размещать реквизит на форме. Его можно использовать как какую-нибудь внутреннюю переменную формы, которая будет доступна и в серверном, и в клиентском контексте.

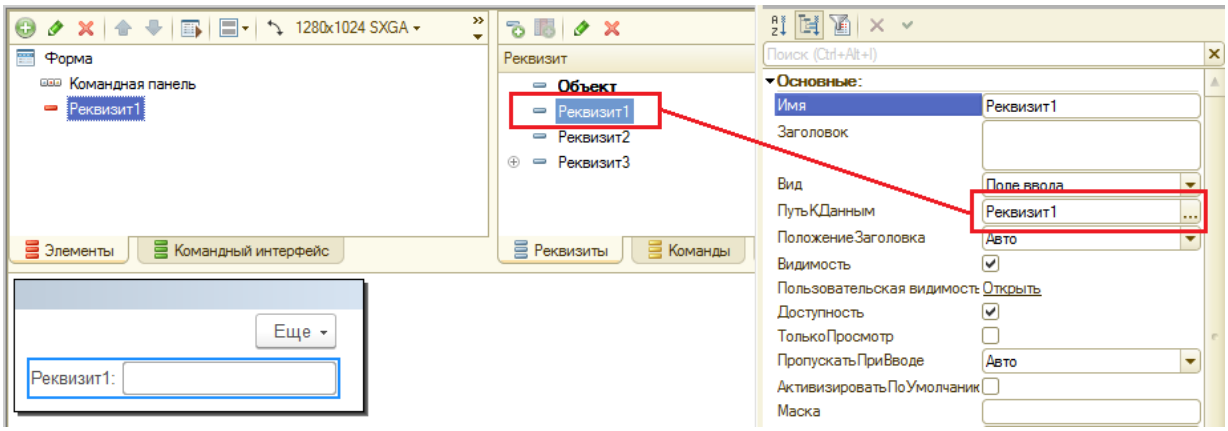


Рис. 5.4.22

Параметры необходимы для передачи информации в форму при её открытии, к ним можно получить доступ только в моменты создания формы и её открытия. После её создания параметры будут уничтожены. Если необходимо, чтобы параметр существовал после создания формы, то ему необходимо установить свойство *Ключевой параметр* (см. рис. 5.4.23).

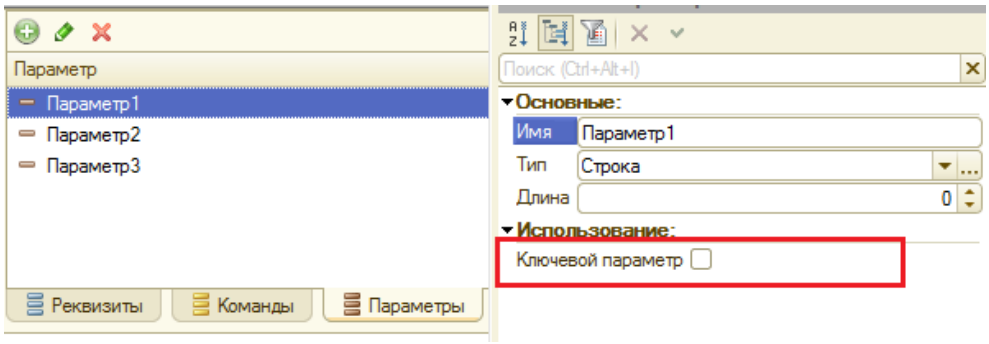


Рис. 5.4.23

Работу с параметрами мы разберем в разделе, когда будем изучать вопросы открытия форм. А пока разберем некоторые моменты использования реквизитов.

Для начала, посмотрим, как используются реквизиты в серверном и клиентском контексте формы. Тут нет ничего особенного. Мы просто пишем в коде название реквизита и присваиваем ему нужное значение. Для примера, я создам две команды, одна будет работать на клиенте, а вторая будет вызывать процедуру на сервере, и размещу их на форме (см. рис. 5.4.24).

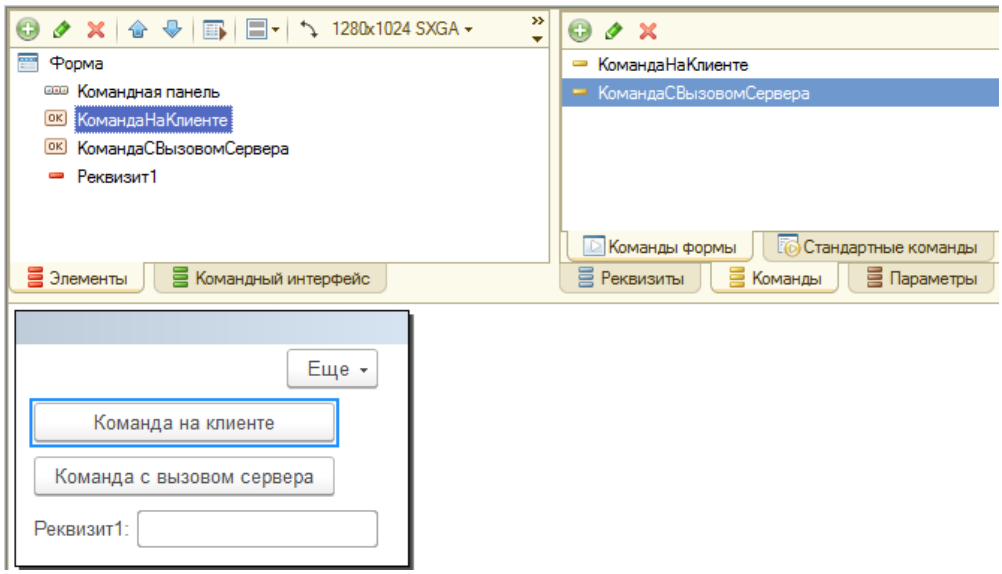


Рис. 5.4.24

Создадим обработчики событий этих команд. В первом случае будет только обработчик на клиенте, а во втором случае обработчик на клиенте будет делать серверный вызов. В каждом обработчике присвоим реквизиту *Реквизит1* какое-нибудь значение (см. листинг 5.4.8).

```
&НаКлиенте
Процедура КомандаНаКлиенте(Команда)
    Реквизит1 = "Заполнили на клиенте";
КонецПроцедуры

&НаСервере
Процедура КомандаСВызовомСервераНаСервере()
    Реквизит1 = "Заполнили на сервере";
КонецПроцедуры

&НаКлиенте
Процедура КомандаСВызовомСервера(Команда)
    КомандаСВызовомСервераНаСервере();
КонецПроцедуры
```

Листинг 5.4.8

Теперь посмотрим, как будет работать код при выполнении обеих команд (см. рис. 5.4.25 и 5.4.26).

Рис. 5.4.25

Рис. 5.4.26

Заметьте, в обоих случаях мы не писали какой-то код, чтобы новое значение автоматически отобразилось в поле на форме. Платформа сама обновила элемент формы при изменении реквизита. Во втором же случае изменение реквизита произошло в серверном контексте формы и тоже элемент на форме отобразился. Когда мы нажали на кнопку, произошел вызов формы, все данные формы были переданы в серверный контекст, потом после окончания вызова они были переданы обратно в клиентский контекст формы. В этом же контексте и произошло обновление элемента на форме.

Но в то же время мы не можем использовать реквизит при внеконтекстном серверном вызове. Например, код, который показан на листинге 5.4.9, ни к чему не приведет. Объясните самостоятельно, почему.

```
&НаСервереБезКонтекста
Процедура КомандаСВызовомСервераНаСервере()

   Реквизит1 = "Заполнили на сервере";

КонецПроцедуры


&НаКлиенте
Процедура КомандаСВызовомСервера(Команда)
   КомандаСВызовомСервераНаСервере();
КонецПроцедуры
```

Листинг 5.4.9

Резюме

На этом мы закончим изучение форм. В этой главе мы научились создавать формы объектов в конфигураторе, узнали, что такое командный интерфейс, познакомились с реквизитами формы и с элементами формы, а также узнали, что такое клиент-серверная архитектура и директивы компиляции. Этой информации Вам хватит для дальнейшего изучения материала книги. В процессе изучения мы углубим наши знания: научимся открывать форму программно и научимся программно работать с элементами формы.

Глава 6. Объектные типы

Часть 1. Объекты метаданных

Теоретическая часть

Перед тем, как начать работать с объектными типами, выясним - что такое *Объекты*? Рассмотрим пример, на основе которого Вам будет легче осмыслить понятие объектов. Пусть мы имеем объект "автомобиль" (например, "Лада-Калина", будем патриотами). У данного автомобиля есть свойства, такие как: объем двигателя, мощность в лошадиных силах, коробка передач (автомат или ручная), количество пассажиров, максимальное количество топлива в бензобаке, текущая скорость автомобиля, его текущий вес и т.п.

И над данным автомобилем мы можем совершать следующие операции: изменять скорость, переключать передачи, заливать бензин, сажать пассажиров, и также очень многое другое. Мы рассмотрели нашу "Ладу-Калину" как объект в целом, но есть еще *Экземпляры объекта*.

Это значит, что "Калину" может купить Вася, Петя или Марья Ивановна и у каждого из них будет свой *Экземпляр объекта* "Лада-Калина".

Некоторые свойства экземпляров объекта могут различаться, например, Вася купил "Ладу-Калину" с автоматической коробкой передач, а Петя с ручной. Васина скорость в данный момент 100 км/ч, он едет по автостраде, а Петина – 10 км/ч, он плетется в пробке.

Какие-то свойства автомобиля может менять пользователь (например, скорость), а какие-то нет (например, коробку передач). Это *открытые* и *закрытые* свойства.

Тем самым приходим к выводу, что *Объект* в языке программирования - это некоторая сущность, обладающая *Свойствами*, над которой можно свершать различные манипуляции посредством *Методов*.

Пользователь может работать не с самими объектами, а с *Экземплярами объектов*, которые он создает либо самостоятельно, либо с помощью кода, написанного разработчиком.

Еще один момент. Обычно водитель совершает действия с автомобилем в определенном порядке: сначала заливает бензин, потом садится в салон, включает зажигание, после этого выжимает сцепление, ставит передачу и трогается. Что произойдет, если водитель сразу попытается тронуться, не совершив эти действия? Правильно, автомобиль не поедет. Т.е. действия объекта обладают некоторой историей.

Так же, как и в случае с автомобилем, поведение объекта определяется его историей: к примеру, мы не сможем отсортировать таблицу значений, если она не заполнена.

Хочу предупредить тех, кто изучал объектно-ориентированное программирование в других языках. Язык 1С не является объектно-ориентированным, как говорят сами разработчики, язык 1С это объектно-прикладной язык. В платформе 1С существуют различные прототипы объектов, с которыми Вы уже частично знакомы. Это справочники, документ и прочие метаданные, которые мы называем *прототипами*. В языке программирования 1С разработчик не может самостоятельно создавать собственные прототипы объектов. Также объекты не обладают такими понятиями, как наследственность, изменчивость и инкапсуляция. Т.е. Вы сможете работать только с теми прототипами объектов, которые создали разработчики платформы 1С. В этом есть свои плюсы и свои минусы. Плюс в том, что любое решение, с которым Вы будете работать, стандартизировано, Вам будут знакомы все прототипы объектов, в нем используемые. Минус в том, что при разработке самостоятельных решений Вы не сможете создавать собственные прототипы объектов, которые покажутся Вам более уместными в данном случае. Вам придется довольствоваться тем, что предоставили разработчики платформы.

Каким образом можно прочитать свойства объекта или исполнить тот или иной метод? Во многих языках программирования, и в 1С в том числе, это делается через точку. Сначала пишется экземпляр объекта, потом точка, потом свойство данного объекта или метод.

В общей схеме, касательно языка 1С, это будет выглядеть так:

Объект.Свойство

Объект.Метод(<Параметры>)

Свойство пишем без скобок, мы можем присвоить ему какое-нибудь значение, а можем и просто прочитать. Если свойство закрыто от редактирования, то можем его только прочитать.

Это выглядит так. Присваиваем свойству значение:

Объект.Свойство = «Товар»;

Читаем значение из свойств:

НашаПеременная = Объект.Свойство;

В первом случае мы присваиваем какому-нибудь свойству строковое значение, причем свойство должно быть тоже строкового типа, иначе оно просто не присвоится.

Вот на это я хочу обратить Ваше внимание! Очень много ошибок связано с тем, что какому-нибудь реквизиту присваивается значение не того типа. Компилятор в этом случае никаких ошибок не выдает. Он просто ничего не присвоит. Будьте внимательны, когда записываете данные в реквизиты.

Теперь посмотрим, как работают методы:

Объект.Метод();

Объект.Метод(Параметр1, Параметр2);

НашаПеременная = Объект.Метод();

НашаПеременная = Объект.Метод(Параметр1, Параметр2);

Метод пишется также через точку, только обязательно должен содержать открывающую и закрывающую скобку. Скобки могут содержать в себе параметры, а могут и не содержать. Также метод может возвращать какое-либо значение, а может и не возвращать.

Обращаю Ваше внимание, что к свойству и методу объекта в модуле объекта можно обращаться напрямую, минуя точку.

Помимо свойств и методов, объекты 1С могут иметь *События*. *Событие* - это то, как объект реагирует на те или иные манипуляции с ним. Например, событием может быть запись справочника или открытие формы или проведение документа по регистрам.

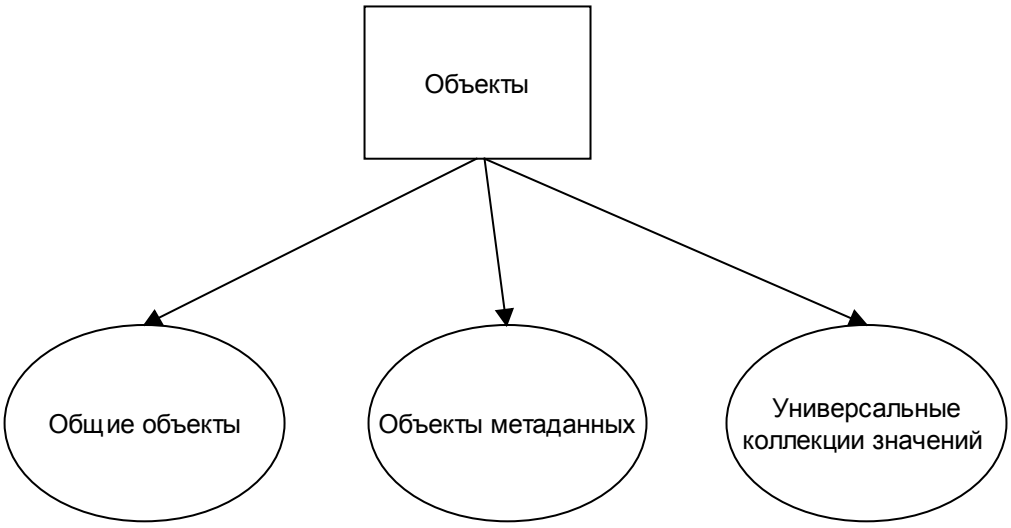
Виды событий являются предопределенными, т.е. разработчик не может самостоятельно придумывать новые события. Он может только создавать имеющиеся события. Например, у объекта *Справочник* есть события *ПриЗаписи* и *ПередЗаписью*, разработчик самостоятельно не может создать новое событие, скажем, *ПослеЗаписи*, он может использовать только имеющиеся события *ПриЗаписи* и *ПередЗаписью,* которые определены создателями платформы 1С. Программист может создать событие *ПередЗаписью*, где будет описан какой-нибудь алгоритм проверки, а может и не создавать, в этом случае объект просто запишется.

Не все объекты имеют события. События имеют только прикладные объекты или объекты метаданных. Подробнее о событиях мы будем говорить позже.

Сейчас перейдем непосредственно к основным объектам.

Объекты 1С

Все объекты в языке 1С можно разбить на три основные группы: *Общие объекты*, *Объекты метаданных (или прикладные объекты)* и *Универсальные коллекции значений*.



В этой главе мы разберем объекты метаданных (прикладные объекты), а в следующей универсальные коллекции значений.

Запомните ещё: условно все типы в языке 1С можно разделить на две большие группы. Это *Примитивные типы*, которые мы прошли во второй главе. И *Объектные типы*, которые мы начали изучать только сейчас. И есть еще тип *Тип*, который не входит в обе эти группы, но о нем позже.

Менеджер Объекта

Объекты метаданных - это элементы конфигурации, такие как: константы, справочники, документы, регистры сведений и регистры накоплений - это все, что нас пока интересует. Они создаются на этапе конфигурирования, а *Экземпляр объекта* - это элемент справочника или документ, который создал пользователь при работе с прикладным решением.

Обращаю Ваше внимание, что все реквизиты документов и справочников, которые создал разработчик в конфигураторе, по своей сути являются свойствами объекта. А все процедуры и функции, написанные в модуле объекта, с ключевым словом *Экспорт* являются методами объекта.

Тем самым для объектов метаданных разработчик может самостоятельно конфигурировать нужные свойства и методы. В отличие от общих объектов. Но у всех объектов также есть и закрытые свойства и методы, которые определили разработчики платформы 1С. Например, у всех справочников есть свойство *Наименование*. Разработчик конфигурации не может его удалить, в отличие от реквизита, который создал на этапе разработки. Хотя и реквизит не стоит удалять, особенно если началась эксплуатация программы.

Основное ключевое понятие метаданных - это *Менеджер*. Для каждого прикладного объекта конфигурации существует свой менеджер, посредством него идет работа с самим объектом. Используя менеджер, можно программно создать объект, получить выборку ссылок объектов (о них мы поговорим позже), найти ссылку на какой-нибудь объект и получить форму объекта.

Программно создать менеджер объекта просто.

```
&НаСервереБезКонтекста
Процедура СоздаемОбъектНаСервере()

    ПрибытиеВГаражМенеджер = Документы.ПрибытиеВГараж;

КонецПроцедуры

&НаКлиенте
Процедура СоздаемОбъект(Команда)
    СоздаемОбъектНаСервере();
КонецПроцедуры
```

Листинг 6.1.1

Из предыдущей главы Вы должны помнить, что работать с объектными типами мы можем или в серверном контексте формы или вообще на сервере без контекста. Обращение к менеджеру документа в листинге 6.1.1 я выполнил в бесконтекстном вызове сервера. Самостоятельно ответьте - почему.

В коде листинга 6.1.1 мы создали переменную, которая является менеджером объекта *Документ «Прибытие в гараж»* (тут и далее мы используем конфигурацию, созданную в четвертой главе). Рассмотрим данный код.

Слово *«Документы»* - это *коллекция* объектов метаданных документов, она включает в себя все объекты - документы, которые мы создали в нашей конфигурации. *«Прибытие ВГараж»* - название конкретного документа (как оно задано в конфигурации), объект которого мы хотим получить.

Менеджер объекта *Справочник* создается по аналогии с документом:

```
&НаСервереБезКонтекста
Процедура СоздаемОбъектНаСервере()

    ПрибытиеВГаражМенеджер = Документы.ПрибытиеВГараж;
    АвтомобилиМенеджер = Справочники.Автомобили;

КонецПроцедуры
```

Листинг 6.1.2

В этом коде все точно так же, только *«Справочники»* - это коллекция объектов метаданных справочников.

Создадим новую обработку, основную форму обработки, и на форме сделаем команду «СоздатьМаркуАвтомобиля». В обработчике команды напишем код, который создает с помощью менеджера справочника *МаркиАвтомобилей* новый экземпляр объекта (см. листинг 6.1.3). Причем переменная *МаркаАвтомобилей* имеет тип *СправочникОбъект.МаркиАвтомобилей*.

```
&НаСервереБезКонтекста
Процедура СоздатьМаркуАвтомобиляНаСервере()
    МаркаАвтомобилейМенеджер = Справочники.МаркиАвтомобилей;
    НоваяМарка = МаркаАвтомобилейМенеджер.СоздатьЭлемент();
    НоваяМарка.Наименование = "Renault";
    НоваяМарка.Записать();
КонецПроцедуры

&НаКлиенте
Процедура СоздатьМаркуАвтомобиля(Команда)
    СоздатьМаркуАвтомобиляНаСервере();
КонецПроцедуры
```

Листинг 6.1.3

В этом случае я тоже делаю бесконтекстный вызов сервера. Почему?

Сохраните Вашу обработку, запустите в «1С:Предприятие» обработку и нажмите кнопку выполнения команды. После этого перейдите в справочник *МаркиАвтомобилей*,

посмотрите на список, и Вы увидите, что появилась новая марка. Вы ее создали программным способом.

Рис. 6.1.1

Если Вы нажмете кнопку несколько раз, то появится много элементов справочника с одинаковым названием. Что, согласитесь, неправильно. Ниже мы научимся избегать таких ситуаций.

Теперь разберем код из листинга 6.1.3.

Первое, мы создаем менеджер объекта. После этого, используя стандартный метод менеджера объекта *«Создать Элемент»,* мы создаем новый экземпляр объекта *«Справочник МаркиАвтомобилей»*. Переменная *НоваяМарка* - это экземпляр объекта. Через эту переменную мы получаем доступ ко всем свойствам объекта и его методам. В данном конкретном случае нас интересует одно свойство: это *Наименование*, и один метод - *Записать*. Это закрытые (типовые) свойства и методы, которые автоматически появляются у экземпляра метаданных при его создании разработчиком.

Остановимся подробнее на функции *СоздатьЭлемент*, она создает новый экземпляр объекта *«Справочник МаркиАвтомобилей»*, но данный экземпляр существует только в оперативной памяти, непосредственно в базе данных он не появился. Но, несмотря на это, для него доступны все свойства данного объекта, и все методы, которые он может применять.

Для того чтобы данный экземпляр появился в базе, мы используем метод *Записать* объекта. После того, как этот метод успешно отработал, данный экземпляр объекта сразу же появляется в базе данных.

Все экземпляры объектов 1С имеют типы *ДокументОбъект.<НазваниеДокумент>*, *СправочникОбъект. <НазваниеСправочника>* и т.д.

Метод *Записать* является предопределенным методом объекта. А как использовать те методы объекта, которые Вы сами написали? Очень просто.

В имеющейся конфигурации в модуле объекта *«Справочник Марки автомобилей»* напишем метод, который выводит оповещение о наименовании в окно сообщений.

Откройте модуль объекта справочника *Марки автомобилей*. И напишите в нем следующий код:

```
Процедура ВывестиНаименование() Экспорт

    Сообщить(Наименование);

КонецПроцедуры // ВывестиНаименование()
```

Листинг 6.1.4

Разберем данный код: Вы создали свою собственную процедуру и сделали ее экспортной. Это простая процедура, призванная Вам показать, как работают методы объектов. Ключевое слово *Экспорт* делает ее внешней процедурой (см. главу 4, часть 2, стр. 265), это значит, что объект может ее вызывать в любом месте программы. Если не указать слово *Экспорт*, то функцию можно будет использовать только внутри модуля объекта. В принципе, ее тоже можно будет считать методом объекта. Только он будет закрыт от работы извне.

Теперь допишите Вашу недавно созданную обработку, где будете вызывать данную процедуру.

Перед этим я расскажу Вам о небольшой возможности, как сделать свое программирование более удобным. Если Вы напишите Ваш объект и поставите точку после данного слова, то платформа Вам выведет список свойств и методов, с которыми работает данный объект. Используйте эту возможность, и она позволит Вам быстрее писать и избежать многих ошибок.

```
&НаСервереБезКонтекста
Процедура СоздатьМаркуАвтомобиляНаСервере()
    МаркаАвтомобилейМенеджер = Справочники.МаркиАвтомобилей;
    НоваяМарка = МаркаАвтомобилейМенеджер.СоздатьЭлемент();
    НоваяМарка.Наименование = "Renault";
    НоваяМарка.Записать();
    НоваяМарка.ВывестиНаименование();
КонецПроцедуры
```

Листинг 6.1.5

Также изменим немного нашу обработку: добавим на форму реквизит *МаркаАвтомобиля* (тип Строка), разместим его на форме (см. рис. 6.1.2), а в свойство наименование нового объекта *Марки автомобилей* будем присваивать значение этого реквизита.

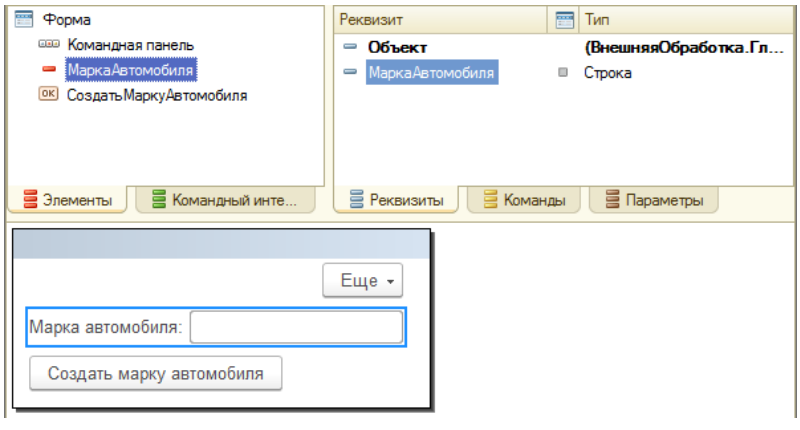


Рис. 6.1.2

Но если мы попытаемся использовать реквизит *МаркаАвтомобиля* в том месте, где создаем экземпляр справочника, то у нас ошибка возникнет уже на этапе синтаксис-контроля.

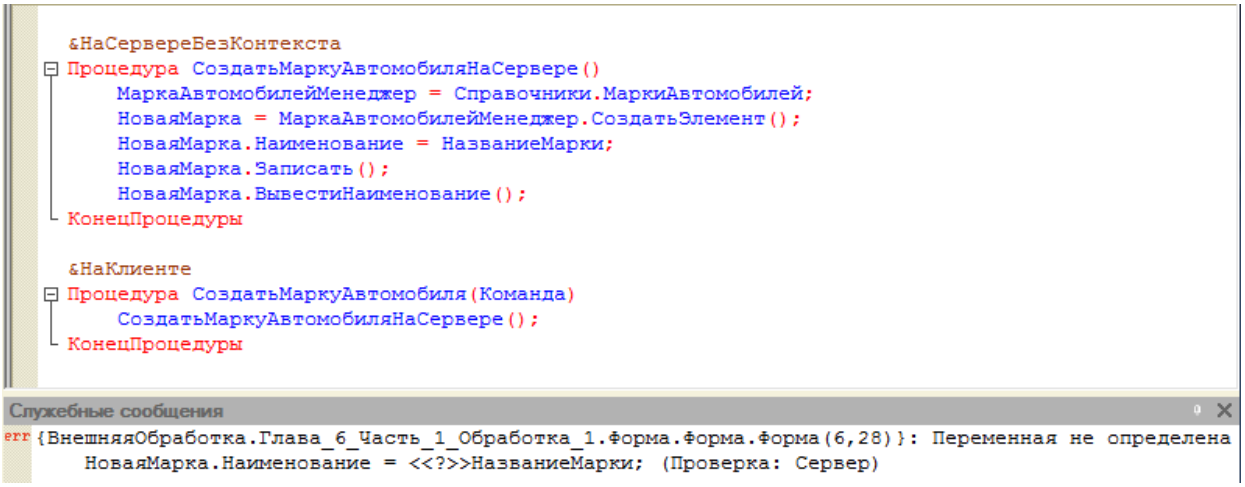


Рис. 6.1.3

В нашем коде мы используем бесконтекстный вызов, как следствие мы не можем в процедуре `СоздатьМаркуАвтомобиляНаСервере` обращаться к реквизитам формы. Из сложившейся ситуации есть два выхода. Первый - сделать контекстный вызов процедуры, второй - передать в процедуру значение реквизита. Я выберу второй способ. В этом случае он будет вполне применим.

```
&НаСервереБезКонтекста
Процедура СоздатьМаркуАвтомобиляНаСервере(НазваниеМарки)
    МаркаАвтомобилейМенеджер = Справочники.МаркиАвтомобилей;
    НоваяМарка = МаркаАвтомобилейМенеджер.СоздатьЭлемент();
    НоваяМарка.Наименование = НазваниеМарки;
    НоваяМарка.Записать();
    НоваяМарка.ВывестиНаименование();
КонецПроцедуры

&НаКлиенте
Процедура СоздатьМаркуАвтомобиля(Команда)
    СоздатьМаркуАвтомобиляНаСервере(МаркаАвтомобиля);
КонецПроцедуры
```

Листинг 6.1.6

Сохраните конфигурацию, обработку и всё перезапустите. При нажатии кнопки *«Выполнить»* должен быть создан элемент справочника *МаркмАвтомобилей,* а в окно сообщений выйдет наименование вновь созданного элемента (см. рис. 6.1.4).

А сейчас Вам небольшое задание: для объекта *«Справочник МаркиАвтомобилей»* создайте метод, который будет возвращать длину наименования. Доделайте обработку: с помощью созданного метода выводите длину названия вновь созданной марки, а также исключите ситуацию некорректной записи наименования, когда пользователь ввел пробелы перед или после основного названия (например, так : «KIA »).

А как создать элемент справочника *Автомобили* программно? Ведь в нем нужно указать в качестве реквизита конкретную марку автомобиля. Можно ли это сделать?

Да, можно, но перед тем, как сделать это, изучим понятие *Ссылка*.

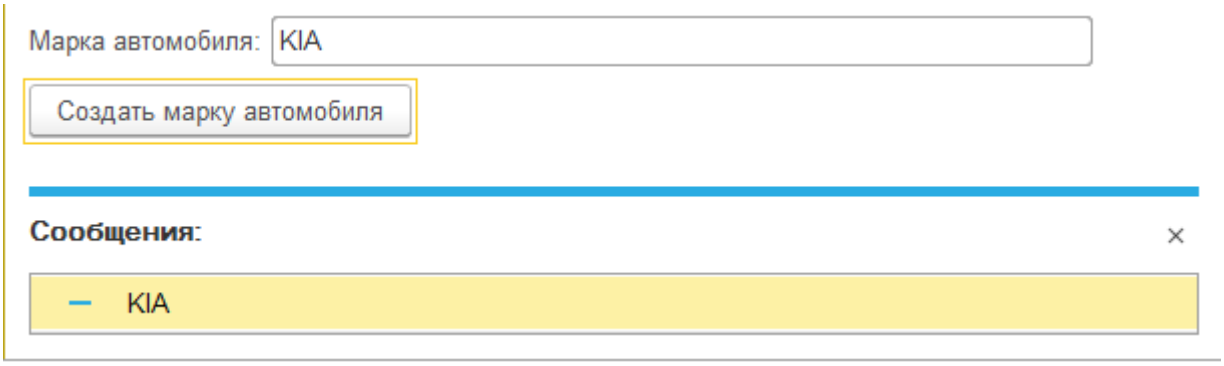


Рис. 6.1.4

Ссылка объекта

Ссылка объекта указывает на данные объекта, но не хранит их, т.е. получая ссылку объекта, мы сможем узнать всю информацию о данном объекте, но не сможем внести какие-нибудь коррективы в эту информацию.

Все ссылки объектов имеют тип *СправочникСсылка.<НазваниеОбъекта>*, *ДокументСсылка.<НазваниеОбъекта>* и т.д.

Ссылку можно использовать в качестве переменной, и также как реквизит других объектов. Если мы используем ссылку как реквизит объекта, то при редактировании экземпляра, на который указывает ссылка, сам реквизит не изменится.

Поясню на примере. В нашем справочнике *Автомобили* есть ссылка на марку автомобиля. Создайте элемент справочника *Автомобили*, выберите марку автомобиля (к примеру, «ВАЗ») и сохраните его.

После этого переименуйте элемент справочника *Марки автомобилей* «ВАЗ» в «Волжский Автозавод».

Зайдите обратно в недавно созданный элемент справочника *Автомобили*, и Вы увидите, что реквизит *Марка* изменился автоматически.

Что это значит? Это не значит, что реквизит *Марки автомобилей* элемента и сам элемент справочника *Автомобили* поменялся автоматически. Нет, это значит, что данный реквизит хранит указатель на экземпляр объекта, который отображается посредством наименования. Поменяли наименование, поменялось только отображение, если бы поменяли какой-нибудь другой реквизит справочника *Марки автомобилей*, то пользователь бы ничего не заметил.

Идем дальше. Каким образом программно получить ссылку на конкретный объект, который есть в базе?

Сделать это можно несколькими способами: найти по коду, найти по наименованию, найти по реквизиту и найти по нескольким реквизитам через запрос. Запросы мы будем разбирать позже, а вот первые три метода рассмотрим.

Перед тем, как разобрать эти методы, напишем код, который будет создавать элемент справочника *Автомобили*.

Для этого создайте обработку и разместите на форме реквизиты: наименование (тип строка (длина 50)), год выпуска (тип дата), гос.номер (тип строка (длина 10)), и коробка передач – тип ссылка на перечисление *Коробка передач* (см. рис. 6.1.5).

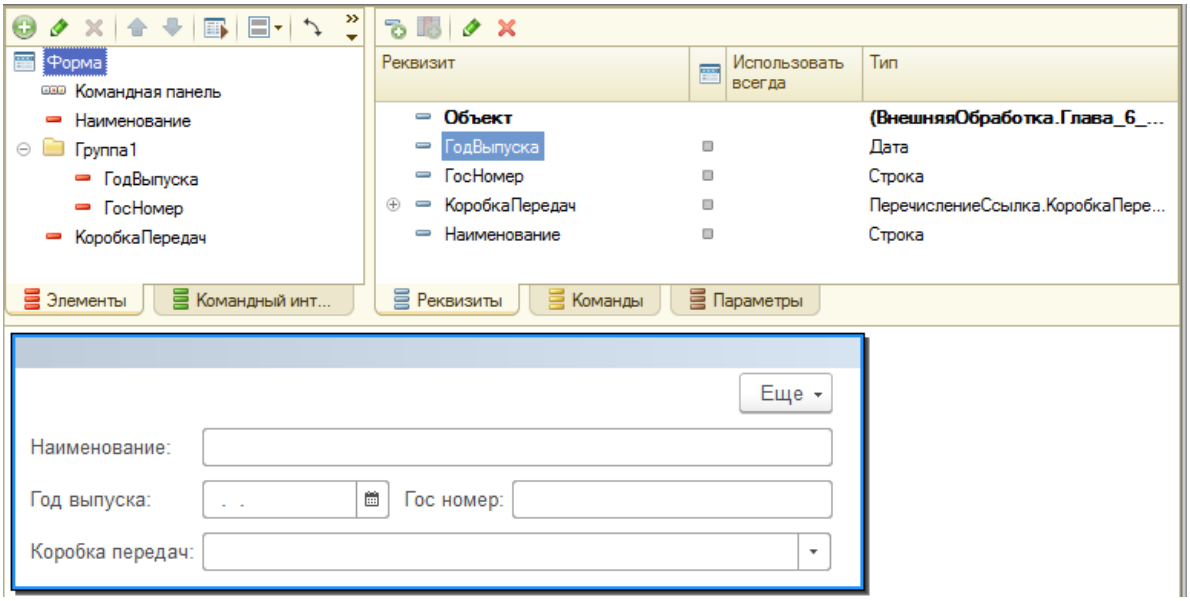


Рис. 6.1.5

Сделаем команду «Создать автомобиль» и в обработчиках команды напишем следующий код (см. листинг 6.1.7).

```
&НаСервере
Процедура СоздатьАвтомобильНаСервере()
    СправочникАвтомобильМенеджер = Справочники.Автомобили;
    НовыйОбъект = СправочникАвтомобильМенеджер.СоздатьЭлемент();
    НовыйОбъект.Наименование = СокрЛП(Наименование);
    НовыйОбъект.ГосНомер    = СокрЛП(ГосНомер);
    НовыйОбъект.ГодВыпуска = ГодВыпуска;
    НовыйОбъект.КоробкаПередач = КоробкаПередач;
КонецПроцедуры

&НаКлиенте
Процедура СоздатьАвтомобиль(Команда)
    СоздатьАвтомобильНаСервере();
КонецПроцедуры
```

Листинг 6.1.7

В этом коде все, как мы уже проходили: сначала создаем менеджер объекта, потом создаем экземпляр объекта *«Справочник Автомобили»*. И заполняем его свойства значениями реквизитов формы. Заметьте, в этот раз используем контекстный вызов сервера.

Пусть Вас не смущает, что реквизиты формы и объекта называются одинаково. У этих свойств разные объекты, поэтому когда названия совпадают, нет ничего страшного. Наоборот, я Вам рекомендую, когда будете заполнять программно реквизиты справочников или документов, называть переменные так же, как реквизиты, чтобы код был более читаемым.

У нас осталось еще два реквизита - это *Марка автомобилей и Модели автомобилей*. Самое простое это разместить их на форме. Но мы сейчас хотим научиться работать со ссылками, поэтому пойдем другим путем.

Разберемся первым делом с реквизитом *Марки автомобилей*. Сначала найдем его по коду. Для этого используем метод менеджера справочника *Марки автомобилей - «Найти по коду»*.

```
&НаСервере
Процедура СоздатьАвтомобильНаСервере()
    СправочникАвтомобильМенеджер = Справочники.Автомобили;
    НовыйОбъект = СправочникАвтомобильМенеджер.СоздатьЭлемент();
    НовыйОбъект.Наименование = СокрЛП(Наименование);
    НовыйОбъект.ГосНомер    = СокрЛП(ГосНомер);
    НовыйОбъект.ГодВыпуска = ГодВыпуска;
    НовыйОбъект.КоробкаПередач = КоробкаПередач;
    НовыйОбъект.Марка = Справочники.МаркиАвтомобилей.НайтиПоКоду("003");
    НовыйОбъект.Записать();
КонецПроцедуры
```

Листинг 6.1.8

В данном коде мы создаем, как Вы должны уже понять, менеджер объекта справочник *Автомобили*, а потом создаем сам экземпляр объекта и заполняем его.

Функция *НайтиПоКоду* возвращает ссылку на экземпляр объекта *«Справочник Марки автомобилей»,* которому соответствует код *001*. Почему мы задали код в виде строки, а не в виде цифры, например? Чтобы понять это, откройте непосредственно справочник *Марки автомобилей* в конфигурации, зайдите на закладку *«Данные»* и посмотрите, какой тип кода у данного справочника.

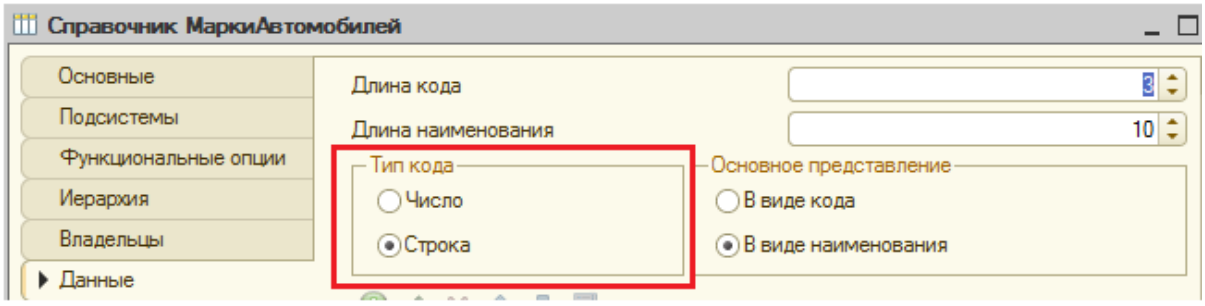


Рис. 6.1.6

Тип кода у данного справочника - строка, поэтому код мы ищем по строковому значению.

Таким образом данный код:

Справочники.МаркиАвтомобилей.НайтиПоКоду("001")

возвращает ссылку на справочник *Марки автомобилей*, код у которого *001.* Если такой нет, возвращается пустая ссылка (о них позже), если с таким кодом несколько элементов, то возвращается один из них, какой конкретно - предугадать невозможно, платформа выберет элемент по своим внутренним идентификаторам.

Рассмотрим синтаксис данной функции:

НайтиПоКоду(<Код>,<ПоискПоПолномуКоду>,<Родитель>,<Владелец>)

Где:

«Код» – непосредственно тот код, по которому мы ищем наш элемент;

«Поиск по полному коду» - если стоит параметр *Истина*, то поиск будет вестись по кодам всей иерархии, т.е. Вам необходимо будет указывать код каждой группы, куда входит элемент через флеш. Например, так: **001/005/110**, где слева будет располагаться самая верхняя группа, а справа непосредственно код элемента.

Данное поле необязательно, по умолчанию – *Ложь;*

«Родитель» – для иерархических справочников можно указать группу, тогда поиск будет вестись внутри группы. Данное поле необязательно;

«Владелец» – для подчиненных справочников можно указать владельца, тогда поиск будет вестись только среди элементов, подчиненных данному владельцу. Данное поле необязательно.

Теперь, когда мы знаем полный синтаксис метода *НайтиПоКоду*, то мы можем найти ссылку на элемент справочника *Модели автомобилей*. В моей базе элемент справочника *Марки автомобилей* с кодом «001» («Ford») является владельцем элемента справочника *Модели автомобилей* с названием Focus и кодом «001» (см. рис. 6.1.7).

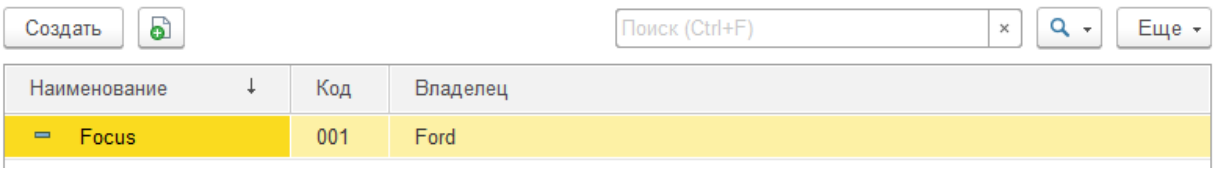


Рис. 6.1.7

Изменим наш код.

```
&НаСервере
Процедура СоздатьАвтомобильНаСервере()
    СправочникАвтомобильМенеджер = Справочники.Автомобили;
    НовыйОбъект = СправочникАвтомобильМенеджер.СоздатьЭлемент();
    НовыйОбъект.Наименование = СокрЛП(Наименование);
    НовыйОбъект.ГосНомер    = СокрЛП(ГосНомер);
```

```
    НовыйОбъект.ГодВыпуска = ГодВыпуска;
    НовыйОбъект.КоробкаПередач = КоробкаПередач;
    Марка = Справочники.МаркиАвтомобилей.НайтиПоКоду("003");
    Модель = Справочники.МоделиАвтомобилей.НайтиПоКоду("001",,,Модель);
    НовыйОбъект.Марка = Марка;
    НовыйОбъект.Модель = Модель;
    НовыйОбъект.Записать();
КонецПроцедуры
```

Листинг 6.1.9

В этом коде мы найденные экземпляры справочников *Марки автомобилей* и *Модели автомобилей* записываем в отдельные переменные, которые потом используем при заполнении реквизитов нового объекта.

Сохраните Вашу обработку, запустите ее в «1С:Предприятии», введите любые данные и выберите коробку передач.

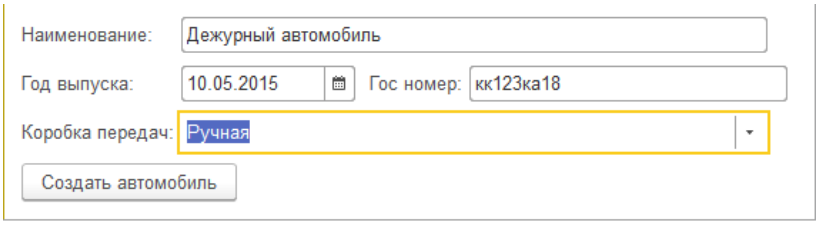


Рис. 6.1.8

Нажмите кнопку *«Создать автомобиль»* и после этого зайдите в справочник *Автомобиль*, где мы видим, что создался новый элемент справочника. Откроем его и посмотрим, какая марка и модель у этого элемента.

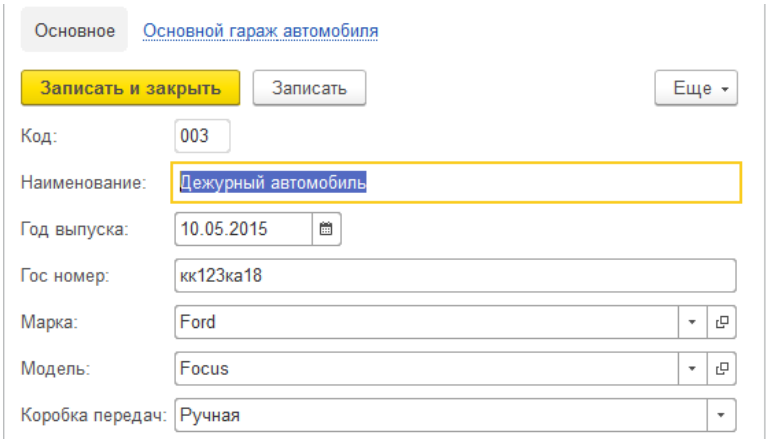


Рис. 6.1.9

Зайдите в справочники *Марки автомобилей* и *Модели автомобилей* и посмотрите, все ли правильно с кодами.

Понятно, что искать элемент справочника просто по коду будет не очень удобно и непрактично. Гораздо удобнее осуществлять поиск по наименованию.

Для такого поиска мы будем использовать метод менеджера справочника *НайтиПоНаименованию*.

Переделаем теперь Вашу обработку, новый код будет выглядеть следующим образом:

```
СправочникАвтомобильМенеджер = Справочники.Автомобили;
НовыйОбъект = СправочникАвтомобильМенеджер.СоздатьЭлемент();
НовыйОбъект.Наименование = СокрЛП(Наименование);
НовыйОбъект.ГосНомер    = СокрЛП(ГосНомер);
НовыйОбъект.ГодВыпуска = ГодВыпуска;
НовыйОбъект.КоробкаПередач = КоробкаПередач;
Марка = Справочники.МаркиАвтомобилей.НайтиПоНаименованию("Ford");
Модель = Справочники.МоделиАвтомобилей.НайтиПоНаименованию("Focus",,,Модель);
НовыйОбъект.Марка = Марка;
НовыйОбъект.Модель = Модель;
НовыйОбъект.Записать();
```

Листинг 6.1.10

Рассмотрим функцию *НайтиПоНаименованию* более подробно. Синтаксис этой функции следующий:

НайтиПоНаименованию(<Наименование>,<ТочноеСоответствие>,<Родитель>,<Владелец>)

Где:

«Наименование» – непосредственно то наименование, по которому мы ищем наш элемент;

«ТочноеСоответствие» - если стоит параметр *Ложь*, то поиск будет вестись не точно, т.е. когда левая часть наименования элемента и строка поиска будут совпадать (например, *Металл* и *Металл 01*), то поиск выдаст результат. Если стоит *Истина*, то будут найдены только те элементы, наименование которых будет точно совпадать со строкой поиска.

Данное поле необязательно, по умолчанию – *Ложь. Помните это, что по умолчанию поиск ведется не точно;*

Параметры «*Родитель*» и «*Владелец*» точно такие же, как для процедуры *НайтиПоКоду*.

Сохраните обработку и посмотрите, как она работает.

Изучим еще один метод поиска ссылки на справочник - это *НайтиПоРеквизиту*.

Для этого мы рассмотрим такой пример: будем искать ссылку на элемент справочника *Автомобили* по гос.номеру.

Создайте обработку, на форме которой будут реквизиты *Гос.номер (тип строка (длинна 10))* и *Автомобиль* (см. рис.6.1.10) . В поле *Автомобиль* будем записывать тот автомобиль, который будет найден по введенному гос.номеру.

Для этого создадим команду «НайтиАвтомобильПоГосНомеру», в обработчике которой напишем следующий код (см. листинг 6.1.11).

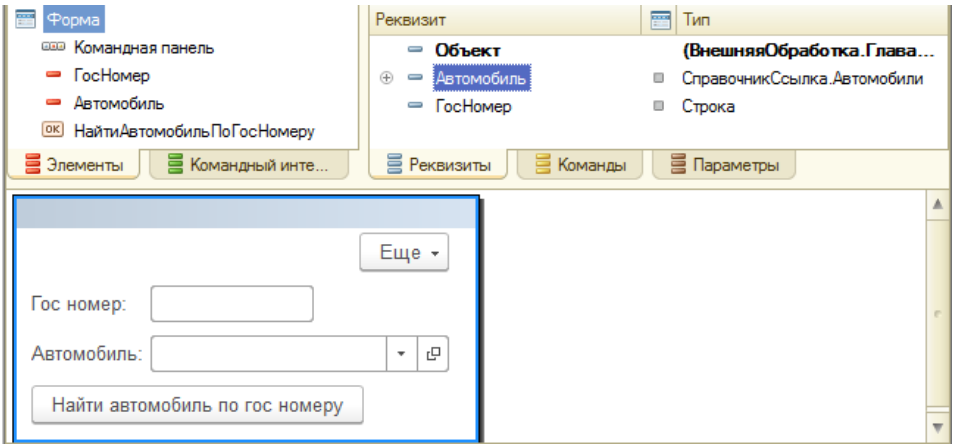


Рис. 6.1.10

```
&НаСервере
Процедура НайтиАвтомобильПоГосНомеруНаСервере()
    СправочникАвтомобильМенеджер = Справочники.Автомобили;
    Автомобиль =
СправочникАвтомобильМенеджер.НайтиПоРеквизиту("ГосНомер",ГосНомер);
КонецПроцедуры

&НаКлиенте
Процедура НайтиАвтомобильПоГосНомеру(Команда)
    НайтиАвтомобильПоГосНомеруНаСервере();
КонецПроцедуры
```

Листинг 6.1.11

Поясним данный код. В нем мы первоначально создали менеджер справочника *Автомобили*. После этого реквизиту формы *Автомобиль* присваиваем значение, которое возвращает функция *НайтиПоРеквизиту* менеджера справочника.

Рассмотрим синтаксис данной функции.

НайтиПоРеквизиту(<ИмяРеквизита>, <ЗначениеРеквизита>, <Родитель>, <Владелец>)

Где:

«ИмяРеквизита» - строковое значение имени реквизита, по которому мы будем искать наш объект. Оно должно быть аналогично тому имени, которое указано в конфигураторе;

«ЗначениеРеквизита» - это значение, по которому должен выполняться поиск. Обращаю Ваше внимание: тип данного значения должен совпадать с типом реквизита. Если объектов с данным реквизитом будет несколько, то будет возвращен один из них. Какой именно, заранее определить невозможно: программа выполнит это по внутреннему идентификатору.

Параметры «*Родитель*» и «*Владелец*» точно такие же, как для процедуры *НайтиПоКоду*.

Пустая ссылка

У Вас уже, наверное, назрел вопрос: что будут возвращать методы *НайтиПоКоду*, *НайтиПоЭлементу* и *НайтиПоРеквизиту*, если такой элемент справочника не был найден. Они будут возвращать *Пустую ссылку*. *Пустая ссылка* - это пустое значение какого-нибудь справочника или документа. У менеджеров справочника или документа есть методы, которые возвращают пустую ссылку соответствующих объектов.

Выглядит он так:

ПустойАвтомобили = Справочники.Автомобили.ПустаяСсылка();

ПустойПрибытиеВГараж = Документы.ПрибытиеВГараж.ПустаяСсылка();

Переменные *ПустойАвтомобили* и *ПустойПрибытиеВГараж* содержат в себе пустые ссылки на каждый объект.

Обращаю Ваше внимание, что для каждого вида справочника и каждого документа будет своя собственная пустая ссылка.

Данный метод очень удобно использовать, когда нам необходимо знать, найден ли элемент справочника по коду (названию) или нет.

Подкорректируем обработку, где мы создавали новый элемент справочника *Автомобили* так, что если ссылка по названию марки найдена, то пишем автомобиль, если - нет, то выходим из процедуры.

```
Марка = Справочники.МаркиАвтомобилей.НайтиПоНаименованию("Ford");
Если Марка = Справочники.МаркиАвтомобилей.ПустаяСсылка() Тогда
    Возврат;
КонецЕсли;
Модель = Справочники.МоделиАвтомобилей.НайтиПоНаименованию("Focus",,,Модель);


Если Модель = Справочники.МоделиАвтомобилей.ПустаяСсылка() Тогда
    Возврат
КонецЕсли;

СправочникАвтомобильМенеджер = Справочники.Автомобили;
НовыйОбъект = СправочникАвтомобильМенеджер.СоздатьЭлемент();
НовыйОбъект.Наименование = СокрЛП(Наименование);
НовыйОбъект.ГосНомер    = СокрЛП(ГосНомер);
НовыйОбъект.ГодВыпуска = ГодВыпуска;
НовыйОбъект.КоробкаПередач = КоробкаПередач;
НовыйОбъект.Марка = Марка;
НовыйОбъект.Модель = Модель;
НовыйОбъект.Записать();
```

Листинг 6.1.12

Поясним этот код: первым делом мы ищем элементы нужных справочников, и если их нет, то ничего не записываем. *Марка* - это ссылка на элемент справочника *Марки автомобилей*, которую мы ищем по наименованию *«Ford»*. Дальше мы проверяем, равна ли переменная *Марка* пустой ссылке, которая возвращается методом *ПустаяСсылка* менеджера соответствующего справочника. Если равна, то выходим из процедуры с помощью операции *Возврат*. То же самое делаем и для элемента справочника *Модели автомобилей*. Самостоятельно добавьте соответствующие сообщения, когда не найдена марка или модель.

Помимо метода менеджера объекта есть и метод ссылки объекта *Пустая*. Данный метод возвращает «истина», если ссылка пустая, и «ложь» в противном случае.

Например, предыдущий код можно переделать следующим образом:

```
Марка = Справочники.МаркиАвтомобиля.НайтиПоНаименованию("Ford");
Если Марка.Пустая() Тогда
    Возврат;
КонецЕсли;
Модель = Справочники.МоделиАвтомобиля.НайтиПоНаименованию("Focus",,,Модель);
Если Модель.Пустая() Тогда
    Возврат
КонецЕсли;

СправочникАвтомобильМенеджер = Справочники.Автомобили;
НовыйОбъект = СправочникАвтомобильМенеджер.СоздатьЭлемент();
НовыйОбъект.Наименование = СокрЛП(Наименование);
НовыйОбъект.ГосНомер    = СокрЛП(ГосНомер);
НовыйОбъект.ГодВыпуска = ГодВыпуска;
НовыйОбъект.КоробкаПередач = КоробкаПередач;
НовыйОбъект.Марка = Марка;
НовыйОбъект.Модель = Модель;
НовыйОбъект.Записать();
```

Листинг 6.1.13

Как видите, данный метод ссылки аналогичен методу менеджера объекта и позволяет в некоторых случаях оптимизировать код программы.

Ссылка как свойство объекта

У любого объекта справочника, документа или константы есть свойство *Ссылка*, которое возвращает ссылку на данный объект.

Зачем она нужна, если мы и так можем получить все данные, имея объект? В основном она применяется в том случае, если мы, создав и записав объект, хотим записать ссылку на этот объект в какой-нибудь реквизит, типом которого будет соответствующая ссылка.

Измените вышеприведенную обработку (а лучше создайте её копию): разместите на форме реквизит *Автомобиль* с типом *Ссылка на справочник Автомобили* (см. рис. 6.1.11), и после создания нового элемента справочника «Автомобиль» будем в этот реквизит записывать ссылку на новый элемент.

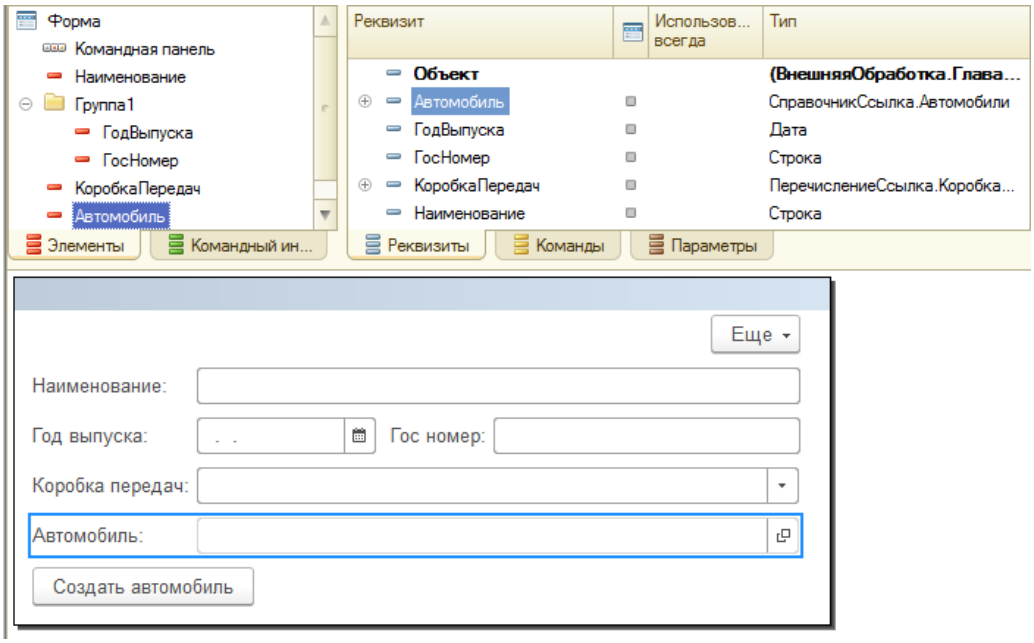


Рис. 6.1.11

У элемента формы *Автомобиль* установим свойство *ТолькоПросмотр* в «Истина», а свойству *КнопкаОткрытия* поставим значение *Да*. Тем самым мы не сможем выбрать другой автомобиль в этом поле, но в то же время сможем его открыть.

Внизу кода обработчика команды допишем:

```
/////........предыдущий код
НовыйОбъект.Марка = Марка;
НовыйОбъект.Модель = Модель;
НовыйОбъект.Записать();

Автомобиль = НовыйОбъект.Ссылка;
```

Листинг 6.1.14

Запустите обработку, заполните все поля и нажмите кнопку *«Создать автомобиль»*. В поле *Автомобиль* появилась ссылка на тот элемент справочника, который Вы только что создали.

Ссылка на перечисление

Такие элементы конфигурации, как *Перечисления,* тоже являются объектами. Но изменять их программно нельзя, только через конфигуратор.

Мы точно так же можем получить менеджер какого-нибудь перечисления и ссылку на значение конкретного перечисления. Измените предыдущую обработку, установите значение реквизита *КоробкаПередач* не интерактивно, а программно, причем это будет одно и то же значение для любого автомобиля, который был создан.

Измените код, где Вы присваиваете реквизиту объекта *КоробкаПередач* значение реквизита формы, на следующий:

```
//.........
НовыйОбъект.ГосНомер    = СокрЛП(ГосНомер);
НовыйОбъект.ГодВыпуска = ГодВыпуска;

НовыйОбъект.КоробкаПередач = Перечисления.КоробкаПередач.Автоматическая;

НовыйОбъект.Марка = Марка;
НовыйОбъект.Модель = Модель;
//.........
```

Листинг 6.1.15

В данном коде: перечисление это объект коллекции метаданных, *Перечисления. КоробкаПередач* это менеджер перечисления *КоробкаПередач*. А в целом эта строка - ссылка на конкретное значение перечисления *Автоматическая*.

Получить объект

Мы с Вами научились создавать объекты, получать ссылки на объекты, а что если стоит обратная задача: есть ссылка, и необходимо отредактировать какой-нибудь реквизит объекта, на который она указывает? Для этого нам понадобится метод объекта *Ссылка – ПолучитьОбъект*. Данный метод возвращает объект ссылки, на который она указывает.

Создадим новую обработку, где по введенному названию будем получать объект *Гараж* и менять его наименование. В этой обработке будет три поля: *Текущее название* (тип строка), *Новое наименование* (тип строка) и *Гараж* (тип *ссылка на справочник Гаражи*). А две команды: *Найти гараж* и *Изменить гараж* (см. рис. 6.1.12). Самостоятельно создайте обработчики к командам на клиенте и на сервере.

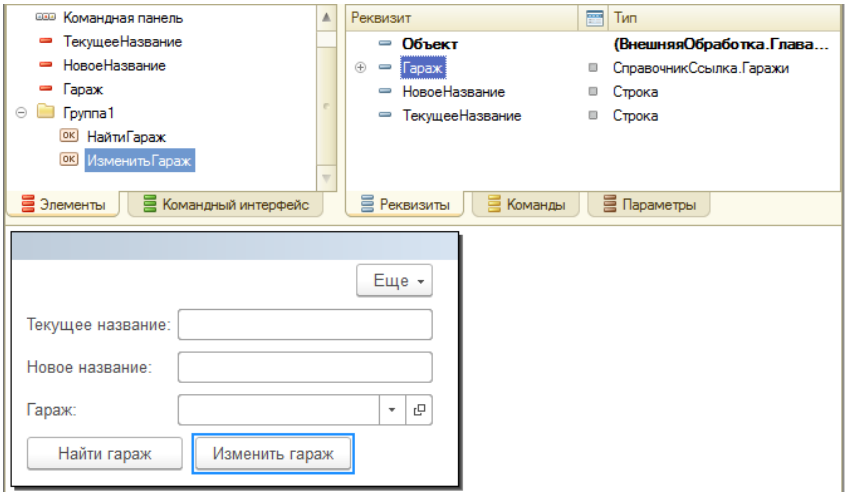


Рис. 6.1.12

В обработчиках команды «Найти гараж» напишите следующий код:

```
&НаСервере
Процедура НайтиГаражНаСервере()

    Гараж =
Справочники.Гаражи.НайтиПоНаименованию(СокрЛП(ТекущееНазвание),Истина);

КонецПроцедуры

&НаКлиенте
Процедура НайтиГараж(Команда)

    Если  ПустаяСтрока(ТекущееНазвание) тогда
        Возврат;
    КонецЕсли;

    НайтиГаражНаСервере();
КонецПроцедуры
```

Листинг 6.1.16

В обработчике команды «Изменить гараж» напишите следующий код:

```
&НаСервере
Процедура ИзменитьГаражНаСервере()
    Если Гараж.Пустая() тогда
        Возврат;
    КонецЕсли;

    ОбъектГараж = Гараж.ПолучитьОбъект();
    ОбъектГараж.Наименование = НовоеНазвание;
    ОбъектГараж.Записать();
    Гараж = ОбъектГараж.Ссылка;
КонецПроцедуры



&НаКлиенте
Процедура ИзменитьГараж(Команда)
    ИзменитьГаражНаСервере();
    ОтобразитьИзменениеДанных(Гараж,ВидИзмененияДанных.Изменение);
КонецПроцедуры
```

Листинг 6.1.17

Рассмотрим данный код. В процедуре *НайтиГараж* мы сначала с помощью функции *ПустаяСтрока* проверяем, заполнен наш реквизит или нет. Функция *ПустаяСтрока*, как Вы помните из предыдущих глав, возвращает *Истина,* если строка не заполнена. Поэтому, когда данная функция возвращает *Истину*, мы выходим из процедуры, и дальнейшего вызова сервера не происходит. Если же она заполнена, мы вызываем серверную процедуру *НайтиГаражНаСервере*, в которой реквизиту формы *Гараж* присваиваем ссылку, найденную по текущему названию.

В процедуре *ИзменитьГаражНаСервере* мы проверяем, является ли реквизит *Гараж* пустой ссылкой, если да, то выходим. Если нет, мы получаем объект с помощью функции *ПолучитьОбъект*, меняем наименование и записываем его. Чтобы наша новая ссылка отобразилась на форме, присваиваем переменной *Гараж* ссылку этого объекта и в процедуре

ИзменитьГараж применяем метод управляемой формы *ОбновитьИзменениеДанных*, который обновит отображение элемента на форме.

Запомните метод *ПолучитьОбъект*, на моей практике не раз приходилось изменять реквизиты документов или справочников, которые по тем или иным причинам были не правильно записаны.

События объектов

У многих объектов (но не у всех) есть *События*. *События* - это, по сути, перехват определенного действия с объектом.

Например, когда мы записываем элемент справочника, мы можем перехватить данное событие и нужным нам способом обработать.

Как получить событие того или иного объекта?

Для этого нам необходимо зайти в модуль этого объекта (установить курсор) и нажать на кнопку *«Процедуры и функции»* в меню *Текст* главного меню.

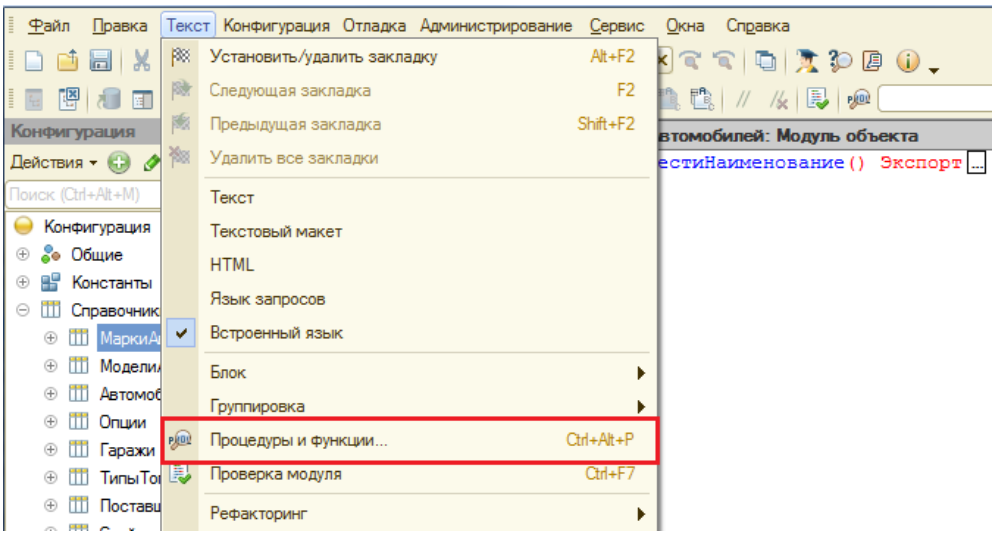


Рис. 6.1.13

В открывшемся окне *«Процедуры и функции»* в скобочках мы видим процедуры и функции, которых еще нет, но они уже предопределены.

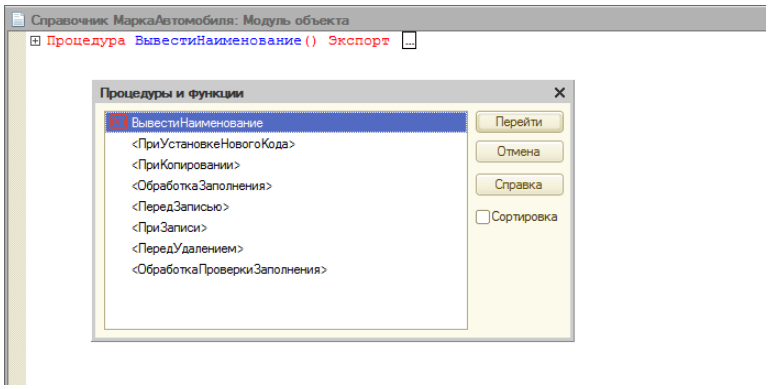


Рис. 6.1.14

Это и есть события на данный объект. Как Вы видите, все события строго предопределены разработчиками платформы, и Вам при написании собственных алгоритмов придется учесть это. Все названия событий интуитивно понятны, и я не буду объяснять их подробно. Кто интересуется, может посмотреть информацию о том или ином событии в синтаксис-помощнике.

Для примера покажу, как работает одно событие - *ПередЗаписью* справочника *Марки автомобилей*. В этом примере, когда создается новый элемент справочника, будем искать, есть ли уже созданный элемент справочника с названием как у нового, и если есть, то не будем записывать элемент.

Откройте модуль объекта *Справочник*, нажмите на *«Процедуры и функции»* в меню *«Текст»* и в открывшемся списке выберите событие *ПередЗаписью*.

Событие создалось, и Вы видите, что у процедуры *ПередЗаписью* есть параметр *Отказ*, если этот параметр принимает значение *Истина*, то запись не будет произведена.

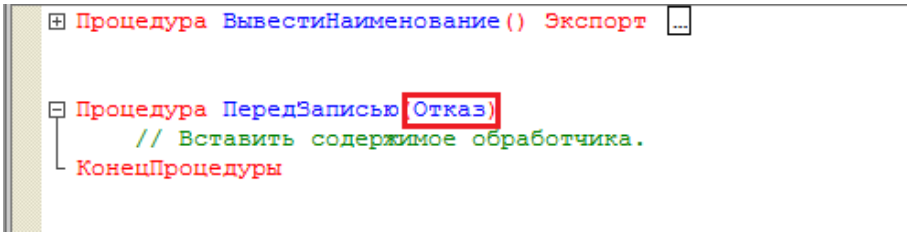


Рис. 6.1.15

В данной процедуре пишите следующий код.

```
Процедура ПередЗаписью(Отказ)

    Если ЭтоНовый() Тогда

        НайденнаяМарка =
Справочники.МаркиАвтомобилей.НайтиПоНаименованию(Наименование,Истина);
        Отказ = ?(НайденнаяМарка.Пустая(), Ложь, Истина);

    КонецЕсли;

КонецПроцедуры
```

Листинг 6.1.18

Код простой, и Вы без труда его разберете. Метод *ЭтоНовый* разъясняется ниже. Теперь попробуйте записать элемент с уже имеющимся названием самостоятельно.

Пытливый читатель спросит: почему тут использовалась процедура *ПередЗаписью?* Дело в том, что процедура *ПередЗаписью* срабатывает тогда, когда элемент еще не записан в базу, в отличие от процедуры *ПриЗаписи*. Потому если бы мы написали такой код в процедуре *ПриЗаписи*, это событие никогда не дало бы нам записать новый элемент.

Метод ЭтоНовый

У любого объекта справочника и документа есть метод *ЭтоНовый*. Данный метод возвращает *Истину*, если объект только что создан интерактивно или с помощью метода *СоздатьЭлемент* (или *СоздатьДокумент*), но при этом не записан в базу. Т.е до того как был выполнен метод *Записать*.

Рассмотрим простой пример, поясняющий этот метод. Изменим предыдущую обработку с гаражами (можете создать новую): если по старому названию гараж не найден, то будем создавать гараж и присваивать ему новое название, а если найден, то менять старое на новое. Пример больше учебный, чтобы показать Вам работу метода *ЭтоНовый*. Изменим код в процедуре *ИзменитьГаражНаСервере*.

```
&НаСервере
Процедура ИзменитьГаражНаСервере()
    Если Гараж.Пустая() тогда
        ОбъектГараж = Справочники.Гаражи.СоздатьЭлемент();
    иначе
        ОбъектГараж = Гараж.ПолучитьОбъект();
    КонецЕсли;

    ОбъектГараж.Наименование = ?(ОбъектГараж.ЭтоНовый(), ТекущееНазвание,
НовоеНазвание);
    ОбъектГараж.Записать();
    Гараж = ОбъектГараж.Ссылка;
КонецПроцедуры
```

Листинг 6.1.19

В данном коде мы проверяем новый это справочник или нет, и если новый, то присваиваем свойству *Наименование* значение реквизита *ТекущееНазвание*.

Пометка на удаление

В программе 1С удалить объекты из базы можно двумя способами: удалить их непосредственно без контроля ссылочной целостности, или удалить через установку пометки и запуском в работу сервиса по удалению помеченных объектов. Данный сервис контролирует

ссылочную целостность и не даст удалить документ, если на него есть какие-нибудь ссылки. В нашей базе документы можно удалять и тем, и другим способом.

Разберем и первый, и второй способ удаления.

Для того чтобы пользователь мог удалять объект без контроля ссылочной целостности, у него должна быть хотя бы одна роль, у которой на этот объект установлено право *Интерактивное Удаление*.

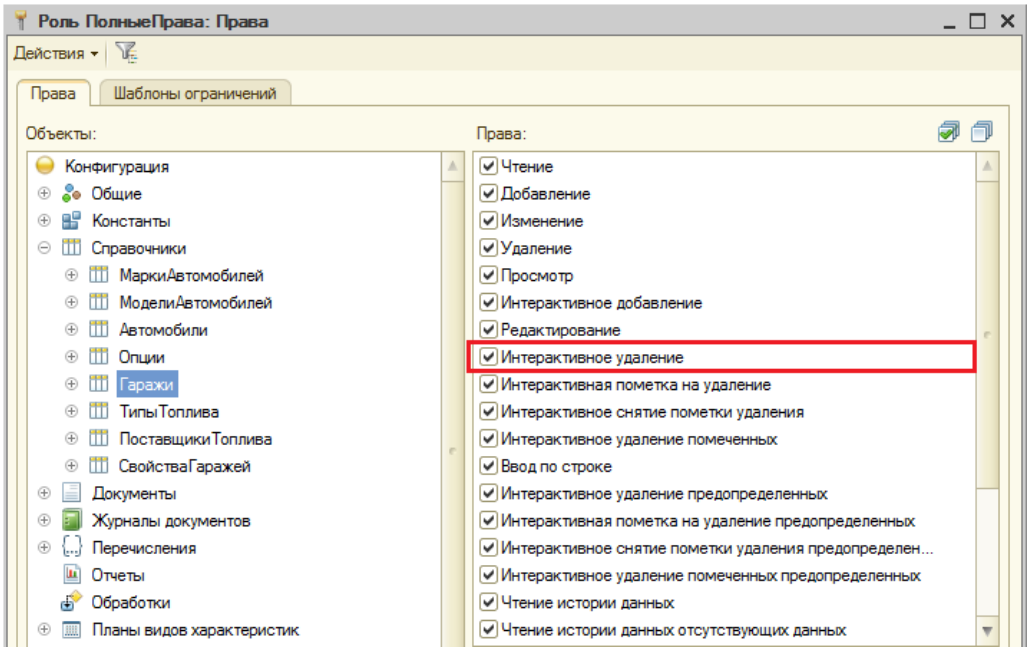


Рис. 6.1.16

Тогда на форме списка и непосредственно у элемента будет доступна команда «Удалить» (см. рис. 6.1.17 и 6.1.18).

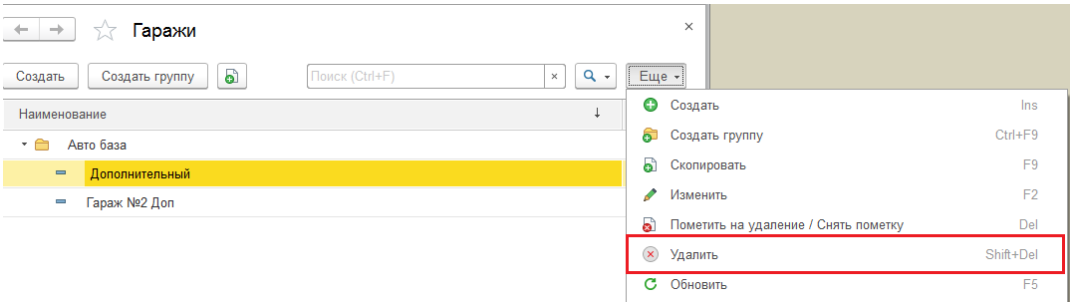


Рис. 6.1.17

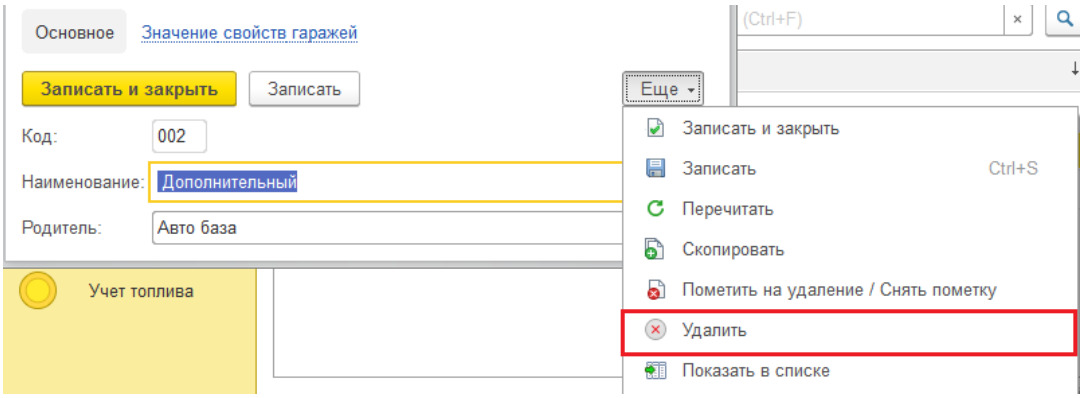


Рис. 6.1.18

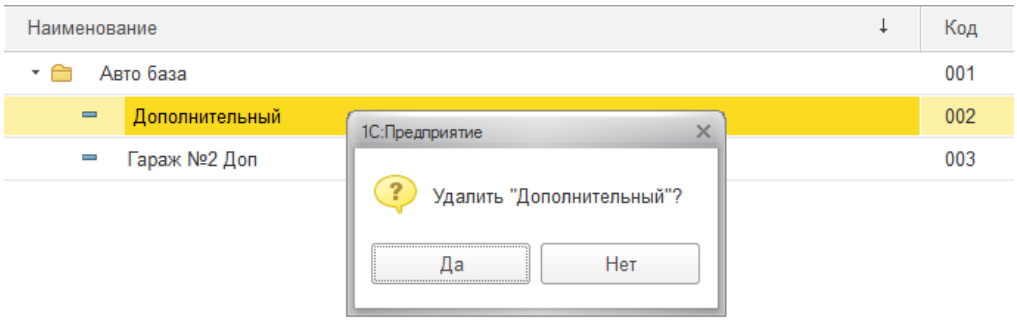


Рис. 6.1.19

Отвечаете *«Да»,* и документ исчез из базы навсегда.

Если же мы уберем право *ИнтерактивноеУдаление* у роли пользователя, то кнопок непосредственного удаления в пользовательском интерфейсе не будет.

Рекомендую при разработке собственных конфигураций (или объектов) всегда снимать право на непосредственное удаление у всех объектов.

Изучим удаление документа через установку пометки на удаление. Для того, чтобы пользователь мог устанавливать пометку на удаление, у него должна быть хотя бы одна роль, в которой для нужного объекта установлено право *Интерактивная пометка на удаление*. А также право *Интерактивное снятие пометки удаления*, чтобы снять уже установленную пометку на удаление (см. рис. 6.1.20).

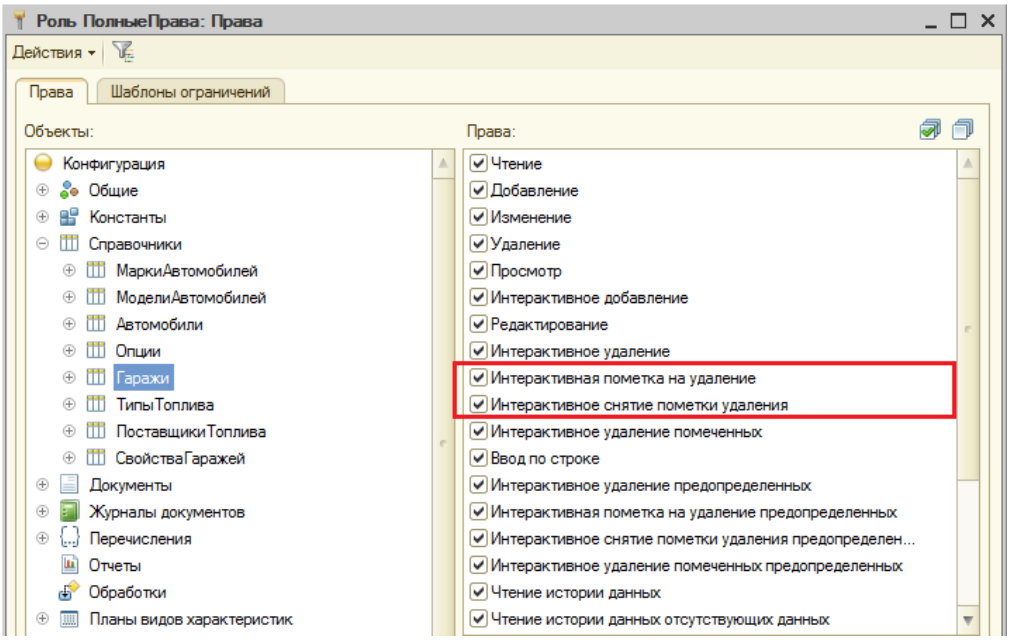


Рис. 6.1.20

Если права на нужный объект установлены, то в командном интерфейсе этого объекта появляются кнопки «Пометить на удаление»/ «Снять пометку».

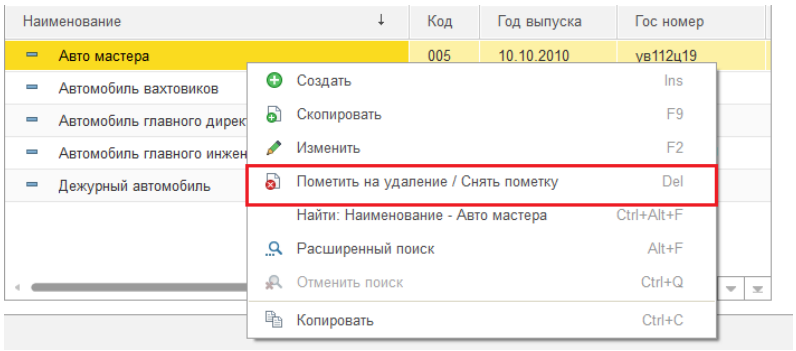
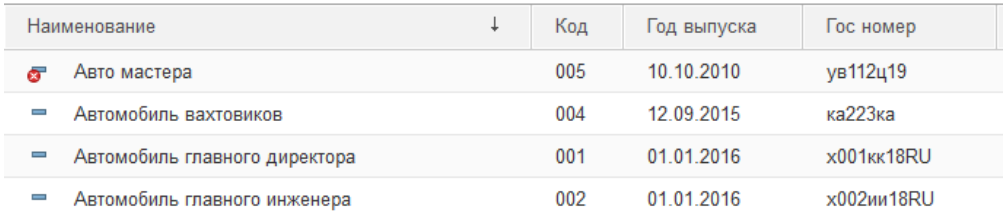


Рис. 6.1.21

Выделите элемент справочника или документ, кликните правой кнопкой мышки и выберите «Пометить на удаление / Снять пометку». Вы увидите, что значок слева обозначился крестиком.



Наименование	Код	Год выпуска	Гос номер
Авто мастера	005	10.10.2010	ув112ц19
Автомобиль вахтовиков	004	12.09.2015	ка223ка
Автомобиль главного директора	001	01.01.2016	х001кк18RU
Автомобиль главного инженера	002	01.01.2016	х002ии18RU

Рис. 6.1.22

Это значит, что документ помечен на удаление и появится в списке сервиса удаления помеченных объектов.

Его нужно удалить с контролем ссылочной целостности. Для этих целей тоже есть право, которое называется *Интерактивное удаление помеченных*.

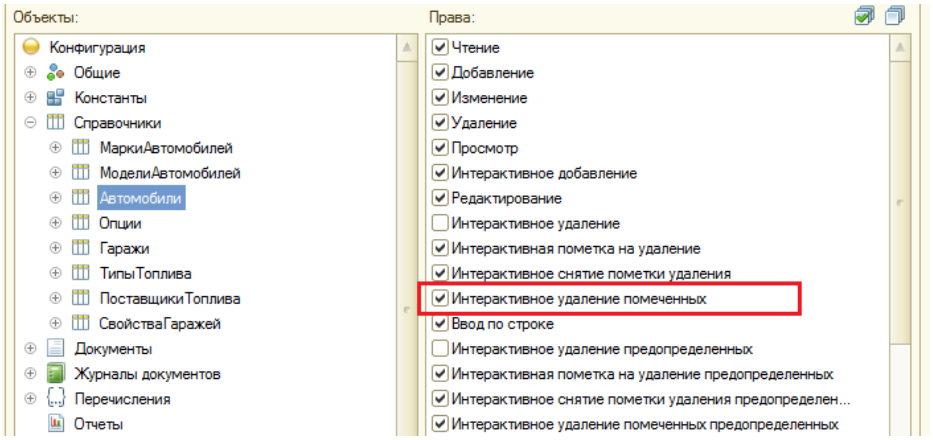


Рис. 6.1.23

Удалим помеченный элемент справочника. Это делается при помощи стандартной обработки «Удаление помеченных объектов». Зайдем в эту обработку через команду «Все функции». Но сначала эту команду нужно включить в нашем интерфейсе. Для этого зайдем в параметры приложения (см. рис. 6.1.24).

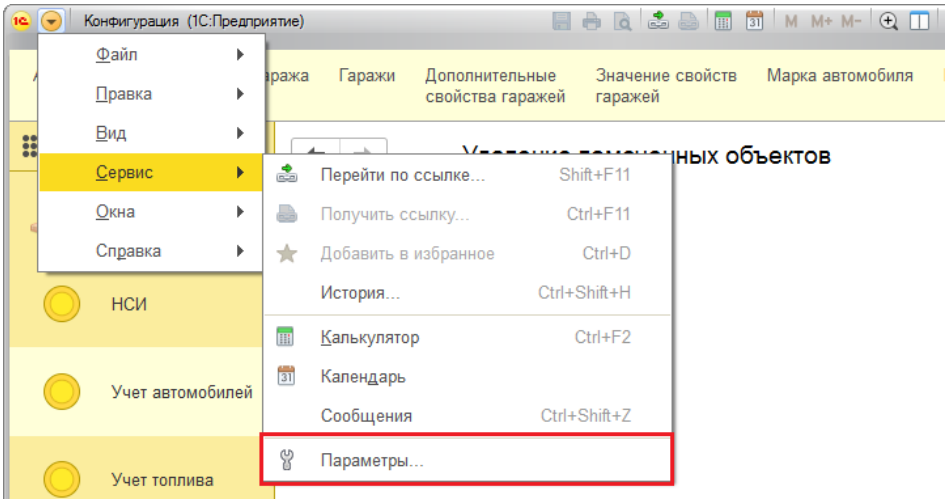


Рис. 6.1.24

И в параметрах установим флаг «Отображать команду «Все функции» (см. рис. 6.1.25).

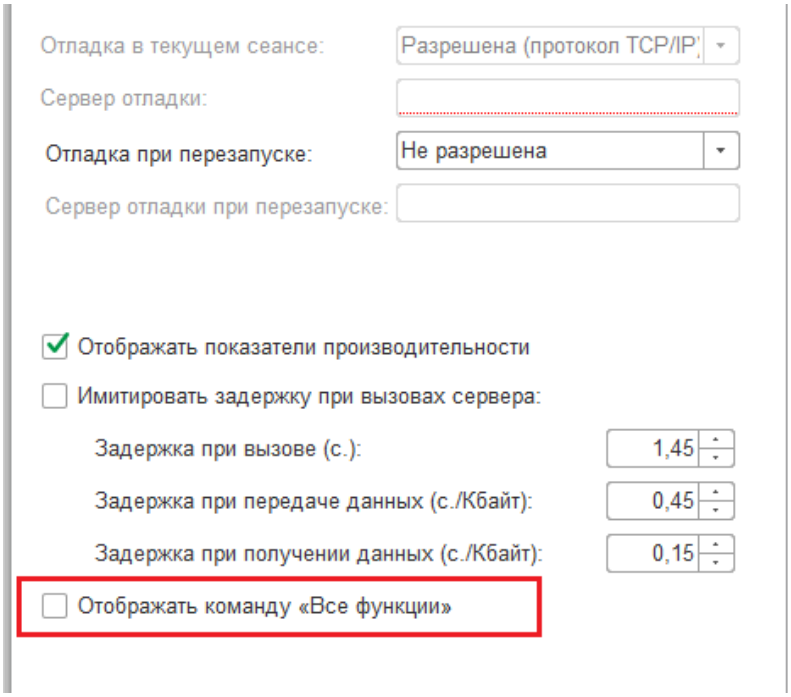


Рис. 6.1.25

У пользователя будет доступна команда «Все функции» только в том случае, когда у него есть хотя бы одна роль, у которой на конфигурацию будет установлено право *Режим "Все функции"* (см. рис. 6.1.26).

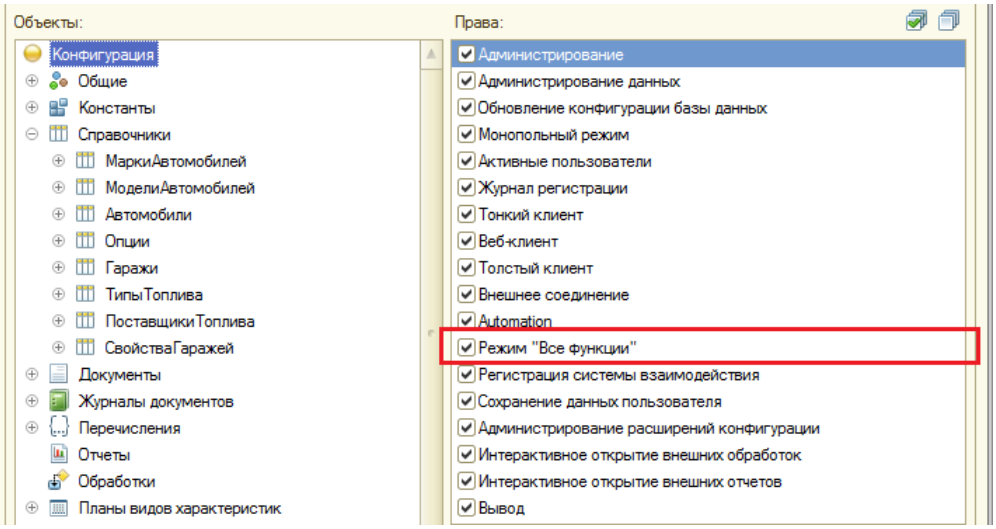


Рис. 6.1.26

После того, как мы установим флаг «Отображать команду «Все функции», в Главном меню пользовательского приложения появится пункт «Все функции».

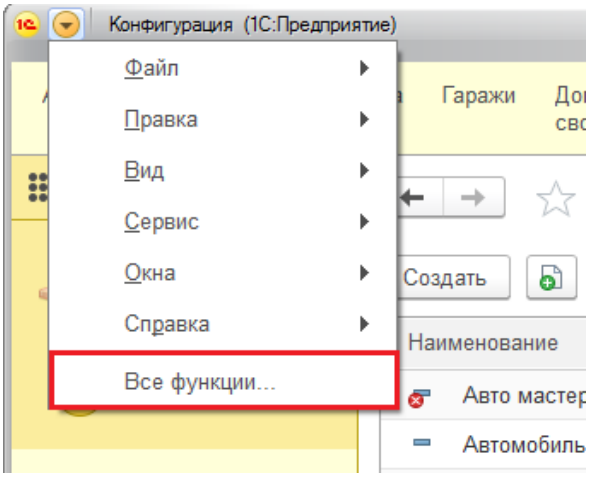


Рис. 6.1.27

В окне «Все функции» в узле «Стандартные» нас интересует обработка «Удаление помеченных документов» (см. рис. 6.1.28). После выбора откроется форма обработки удаления помеченных объектов (см. рис. 6.1.29). В этой обработке мы можем выбрать или полное удаление всех помеченных объектов, или выбрать самостоятельно.

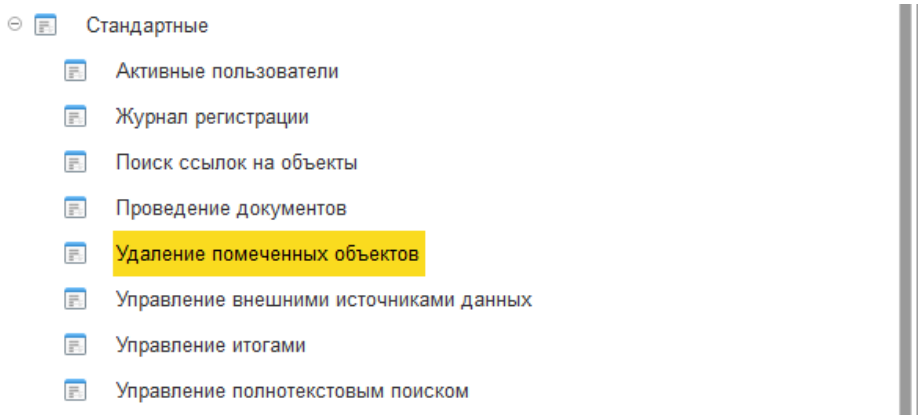


Рис. 6.1.28

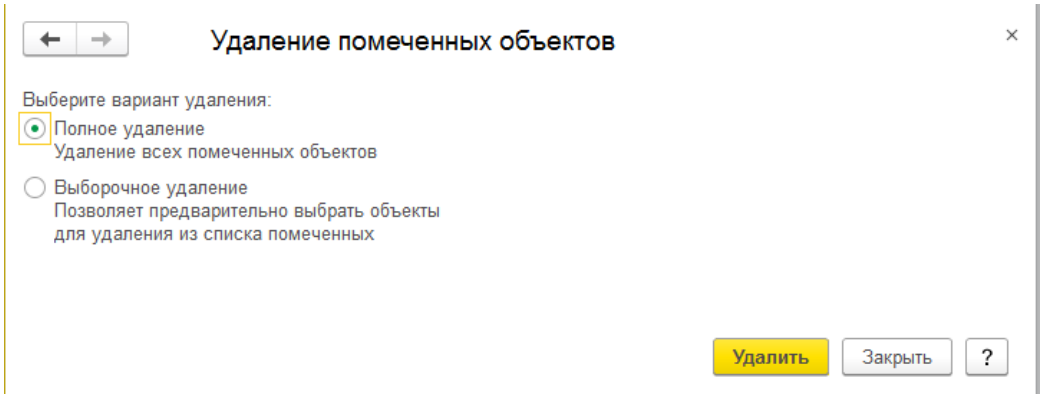


Рис. 6.1.29

Если мы отметим «Выборочное удаление» и нажмем кнопку «Далее», то появятся помеченные на удаление объекты, сгруппированные в виде дерева, где узлы - это объекты

метаданных (см. рис. 6.1.30). Нажмем кнопку «Удалить», и если нет ссылок на отмеченные объекты, они будут удалены.

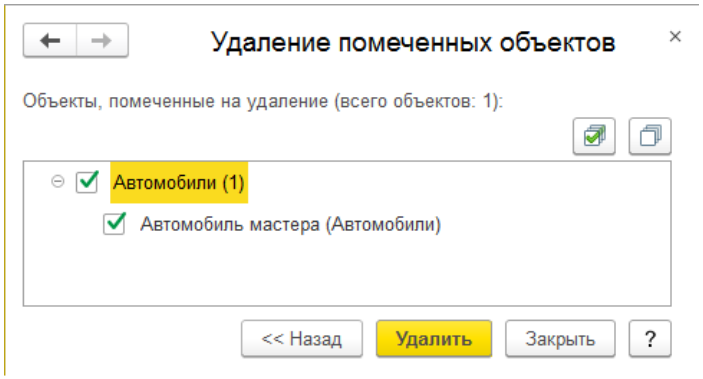


Рис. 6.1.30

Также *ПометкаУдаления* является свойством таких объектов, как *Справочник*, *Документ, ПланВидовХарактеристик* и т.д. Данное свойство имеет тип булево, оно принимает значение *Истина*, если объект помечен на удаление, и *Ложь,* если нет.

Рассмотрим, каким образом можно программно удалить объект и установить пометку удаления. Для удаления непосредственно объекта из базы применяется метод объекта *Удалить*, а для установки пометки на удаление метод объекта *УстановитьПометкуУдаления*.

Рассмотрим эти методы в новой обработке, где мы будем удалять элементы справочника. Создайте новую обработку, форму, а на форме реквизит *Автомобиль (тип ссылка на справочник Автомобили)*, и реквизит *Удалить непосредственно* (тип *Булево*), а также команду *Удалить автомобиль* (см. рис. 6.1.31).

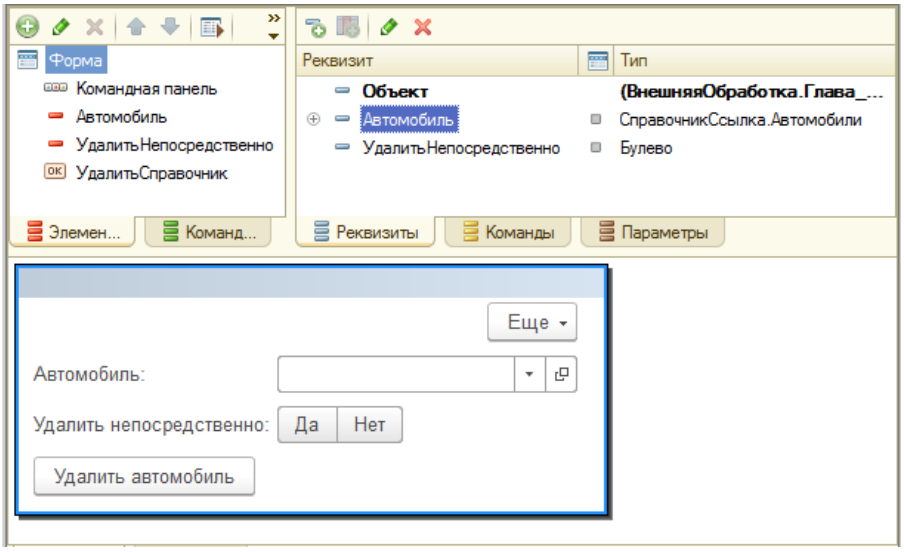


Рис. 6.1.31

В том случае, если флаг установлен – будем удалять объект непосредственно из базы, в противном случае - просто будет установлена пометка на удаление.

В обработчик команды напишем следующий код:

```
&НаСервере
Процедура УдалитьАвтомобильНаСервере()

    Если Автомобиль.Пустая() тогда
        Возврат;
    КонецЕсли;

    ОбъектАвтомобиль = Автомобиль.ПолучитьОбъект();

    Если УдалитьНепосредственно тогда
        ОбъектАвтомобиль.Удалить();
        Автомобиль = Справочники.Автомобили.ПустаяСсылка();
    иначе
        Если Не ОбъектАвтомобиль.ПометкаУдаления тогда
            ОбъектАвтомобиль.УстановитьПометкуУдаления(Истина,Истина);
        КонецЕсли;
    КонецЕсли;

КонецПроцедуры

&НаКлиенте
Процедура УдалитьАвтомобиль(Команда)
    УдалитьАвтомобильНаСервере();
КонецПроцедуры
```

Листинг 6.1.20

Разберем данный код.

Перед непосредственно удалением мы проверяем, есть ли конкретный объект в нашем реквизите, или ссылка пустая. Если она пустая - выходим из процедуры и ничего не делаем. Следующим шагом получаем объект из ссылки. В дальнейшем, если установлен флаг *Удалить непосредственно*, удаляем объект из базы. Иначе, в том случае если не установлена пометка на удаление, устанавливаем ее.

Разберем метод *УстановитьПометкуУдаления* подробно. У данного метода следующий синтаксис:

УстановитьПометкуУдаления(<ПометкаУдаления>, <ВключаяПодчиненные>)

Функция *УстановитьПометкуУдаления* имеет два параметра. Первый параметр *ПометкаУдаления* имеет тип значения булево. Если он *Истина*, то на данный объект будет установлена пометка удаления, а в противном случае она будет снята (если она установлена).

Второй параметр имеется только для объектов *Справочник* и тоже имеет тип значения булево. В том случае, когда его значение *Истина*, будет помечен не только сам объект, но и все подчиненные. Значение по умолчанию обоих параметров – *Истина*.

Помните, что программное удаление справочников и элементов с помощью пометки на удаление выполнит данную процедуру в более щадящем режиме для базы, чем прямое удаление. Поэтому, хотя это и накладно по времени, рекомендую Вам всегда удалять объекты с помощью установки пометки на удаление, это убережет Вас от многих проблем в будущем.

Если Вы запустите эту обработку, то заметите, что элемент справочника удалился, даже когда право *Интерактивное удаление* снято. Все правильно, потому что мы удалили экземпляр

объекта не интерактивным, а программным способом. За удаление объектов программным способом отвечает право *Удаление* (см. рис. 6.1.32). Снимите флаг этого права у справочника *Автомобиль* и попробуйте удалить непосредственно при помощи обработки, возникнет ошибка (см. рис. 6.1.33).

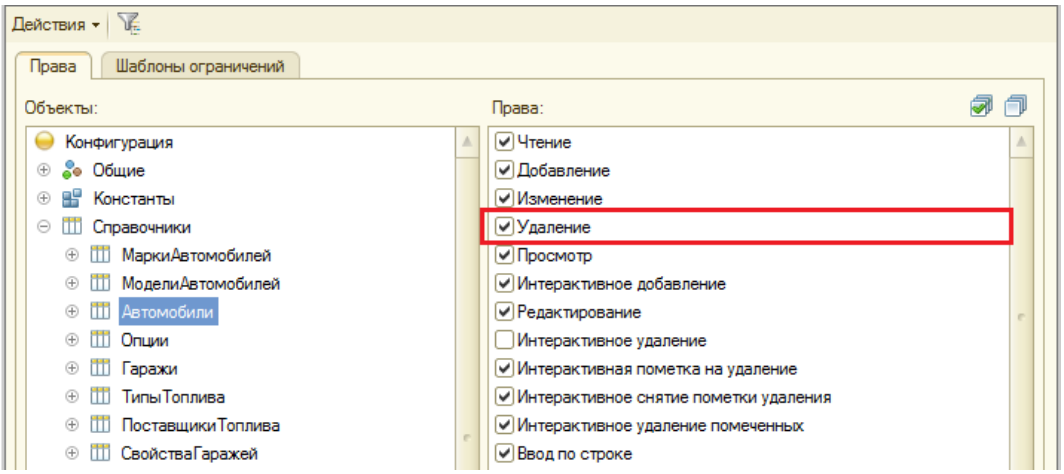


Рис. 6.1.32

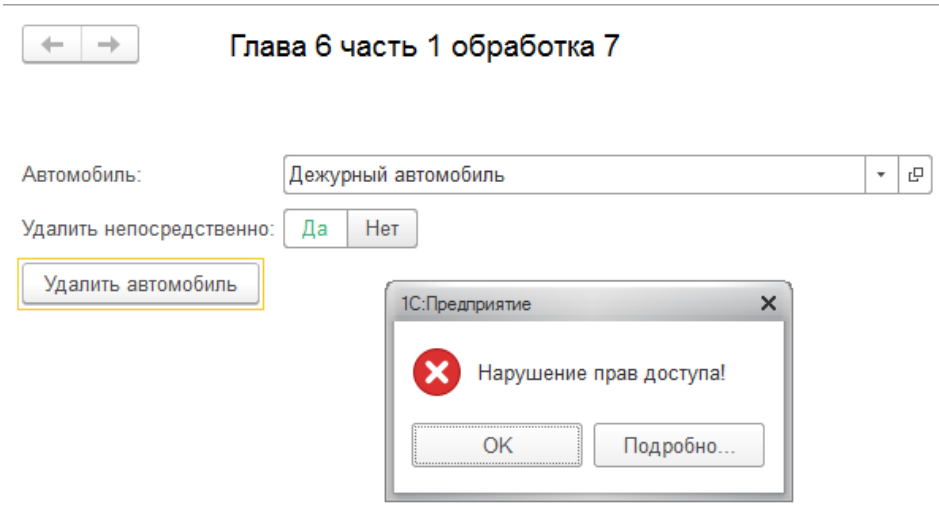


Рис. 6.1.33

Резюме

В этой части мы изучили объекты метаданных и ссылки на них. Научились работать с менеджером объектов, создавать программно объекты и находить ссылки на справочники по коду и наименованию и многое другое, что Вам пригодится в дальнейшей работе. Все это мы делали на примере справочников, поскольку более подробно с документами мы будем работать в следующей части данной главы.

Также в этой части мы изучили основные методы и свойства объектов и справочников. Некоторые из оставшихся мы будем изучать в следующих частях, а с остальными Вы сможете ознакомиться самостоятельно.

Часть 2. Работа с документами

Как Вы помните из четвертой главы, посвященной конфигурированию, в программе 1С можно разрабатывать различные документы, которые необходимы для фиксации периодически повторяющихся событий в деятельности предприятия. Например, это может быть покупка и продажа товара, внесение денежных средств, выдача денежных средств и т.п.

Проведение документов

Когда Вы разрабатываете документ, Вам необходимо продумать, будет ли этот документ делать какие-нибудь движения по регистрам (бухгалтерским, накопления и т.п.). Если никаких движений данный документ осуществлять не должен (например, это просто счет в виде бланка), то необходимо на этапе создания запретить ему осуществлять движения. Сейчас мы научимся делать это.

Откройте Вашу конфигурацию по автотранспорту, которую мы сделали в четвертой главе, и создайте документ *Печать путевых листов*, данный документ не будет проводиться ни по каким регистрам учета, а необходим только для печати путевого листа (работу с печатью мы будем проходить в 11-м уроке). Включите его в подсистему *Учет автомобилей*, а права на создание и редактирование, установку пометки на удаление назначьте роли *Диспетчер*.

После того, как Вы создали документ, перейдите на закладку *«Данные»* и добавьте новый реквизит *Автомобиль* и реквизит *Гараж* с соответствующими типами.

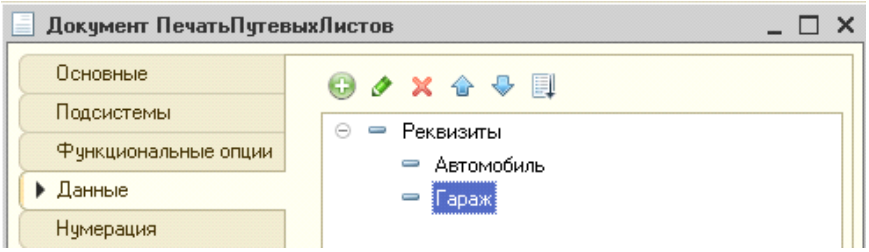


Рис. 6.2.1

Теперь Вам необходимо задать, что Ваш документ не делает никаких движений. Для этого перейдите на закладку *«Движения»* и в пункт *«Проведение»* установите значение *«Запретить».*

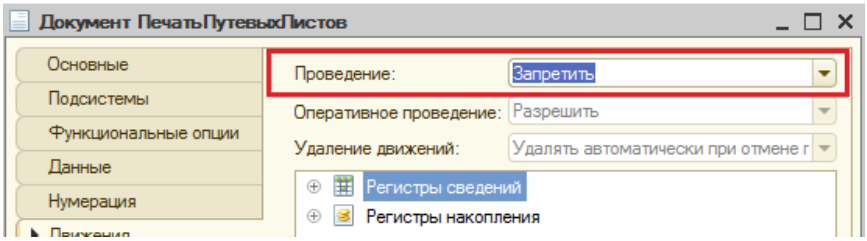


Рис. 6.2.2

Все, теперь данный документ не будет делать никаких движений. Создайте самостоятельно все формы для него.

Алгоритмы, которые обеспечивают проведение документа по регистрам, находятся в предопределенной процедуре *«Обработка Проведения»* модуля объекта. Данная процедура начинает работать тогда, когда пользователь нажимает кнопку *«Провести документ»*, или при программной записи документа с установленным параметром *Проведение* (об этом ниже).

Откройте модуль документа *Прибытие в гараж.*

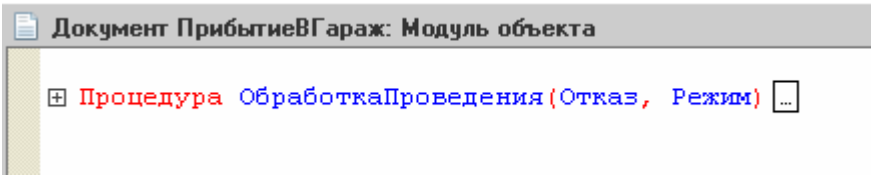


Рис. 6.2.3

Вы видите, что в данном модуле уже существует процедура *«Обработка Проведения»*. Ее создал конструктор проведения, когда в четвертой главе Вы конфигурировали этот документ.

Разверните ее, и Вы увидите некоторый алгоритм, который осуществляет движения по регистру накопления *Пробег автомобиля*. Более подробно про движения мы поговорим ниже.

Оперативное и неоперативное проведение

Если документ может иметь движения (или, грубо говоря, проводки), необходимо еще определиться, будет данный документ проводиться *оперативно* или нет.

Все документы 1С могут быть проведены в *оперативном* и *неоперативном* режиме.

Оперативное проведение - это проведение здесь и сейчас, т.е. в текущий момент времени.

Неоперативное проведение - это проведение документа ранее текущего момента времени.

Т.е. если хотя бы 1 секунда отличается от текущего момента времени, то данное проведение *неоперативное*!

Когда документ проводится в неоперативном режиме, по сути, учитывается уже свершившийся факт, который не требует контроля, осуществляемого при оперативном проведении (например, контроль остатков).

Также необходимо заметить, что если у документа есть возможность оперативного проведения, то такой документ нельзя провести будущей датой.

Возможность оперативно проводить документ можно задать на этапе разработки документа.

Зайдем в конфигурации по учету автотранспорта в уже созданный документ и посмотрим, где устанавливается возможность проведения документа в оперативном режиме.

Пусть это будет документ *Прибытие в гараж*. Откройте его и перейдите на закладку *Движения*.

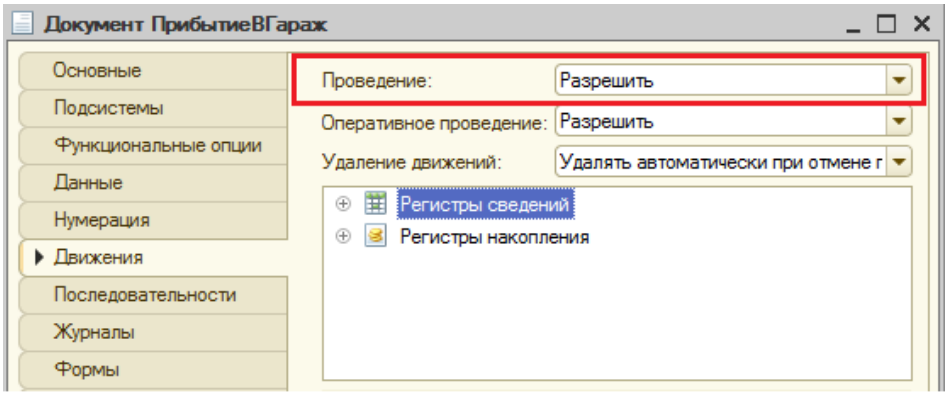


Рис. 6.2.4

Посмотрите на пункт *Оперативное проведение*. Если в данном пункте будет установлено *Разрешить*, то документы могут проводиться в оперативном режиме. Если *Запретить*, то они всегда будут проводиться в неоперативном режиме, в любой момент времени. В каких случаях нам необходимо разрешать оперативное проведение? Однозначно тогда, когда Вы автоматизируете оперативный учет.

Оперативный учет - в двух словах, это когда необходимо знать текущую ежеминутную информацию и принимать соответствующие решения.

Самый яркий пример это розничный магазин: продавец не может в данный момент времени продать покупателю отсутствующий товар. Для того чтобы знать, какие товары есть на складе магазина, а каких нет, необходимо оперативно вести документы прихода товара и выбытия (продажи) товара. Данные документы необходимо вводить в программу по случившемуся факту: пришел покупатель, отгрузили товар, сразу же завели документ в программу. Тогда продавец в любой момент времени может ответить, есть у него товар в магазине или нет.

При автоматизации любого другого учета, к примеру, зарплатного или бухгалтерского, нет необходимости точно владеть ежеминутной информацией о состоянии учета. Как правило, бухгалтера в текущий момент заводят первичные документы, а сводят информацию несколько позже, когда подходит время сдачи бухгалтерской или управленческой отчетности.

Даже если Вы посмотрите уже готовые конфигурации 1С, например, «1С:Управление торговлей» или «1С:Розница», то заметите, что многие документы имеют возможность оперативного проведения. А в то же время программы «1С:Бухгалтерия» или «1С:Зарплата» и «1С:Кадры» имеют документы с неразрешенным оперативным проведением, потому что оно там не требуется.

Теперь решим, для каких документов нам нужно установить возможность оперативного проведения, а для каких запретить.

Начнем с документов *Заправка топлива* и *Отчет о расходе топлива*. Первым делом ответим на вопрос: нужно ли нам знать в текущий момент, сколько топлива в том или ином автомобиле? Очевидно, что да! Как мы отправим автомобиль в рейс, не заправив его топливом или не дав денег водителю этого автомобиля? Перед отправкой автомобиля в рейс нам точно нужно знать, сколько на текущий момент топлива в баке автомобиля, и исходя из этого принимать решение: заправлять автомобиль или нет. Поэтому для этих документов установим разрешение оперативного проведения.

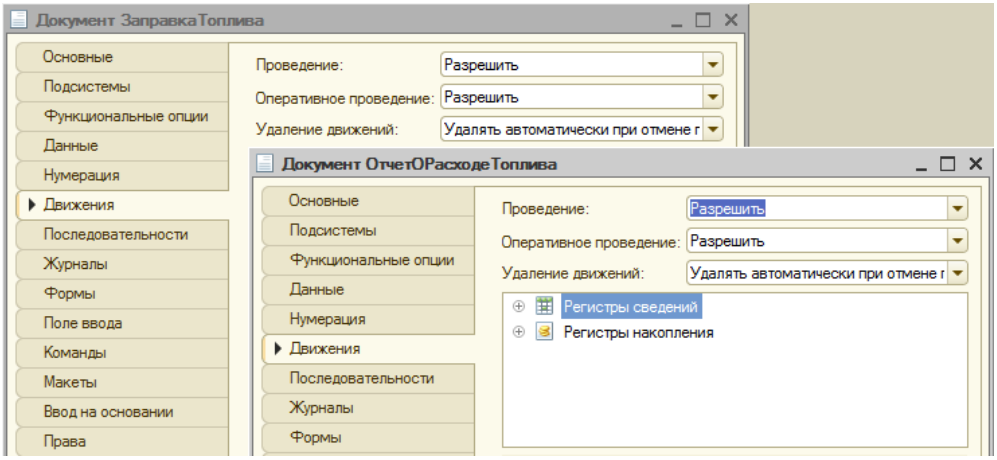


Рис. 6.2.5

Документы *Прибытие в гараж* и *Выбытие из гаража*. Нужно ли нам точно в текущий момент знать, какой автомобиль в рейсе, а какой нет? Конечно, нужно, иначе какой смысл фиксировать выбытие и прибытие автомобилей? Единственно, на данный момент мы не сможем получить информацию об этом, потому что не создан регистр накопления, с помощью которого мы сможем осуществлять контроль наличия автомобиля в гараже.

Поэтому у обоих этих документов установим признак оперативного проведения.

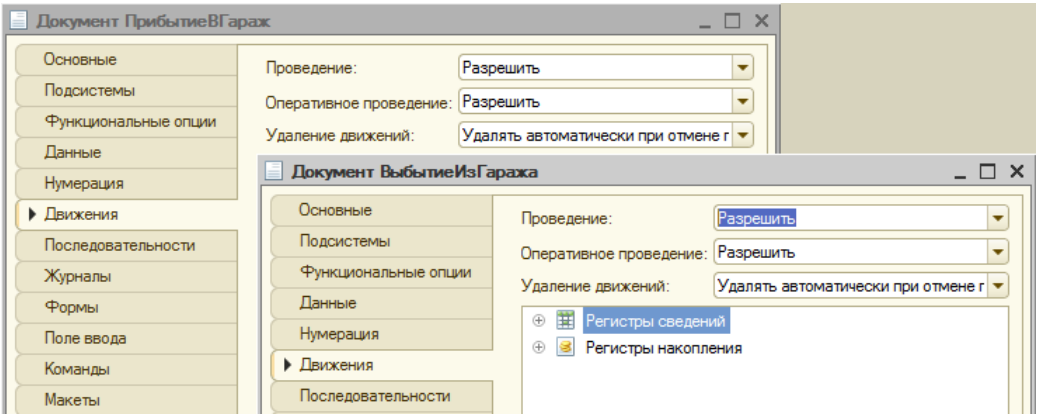


Рис. 6.2.6

А вот документу *Установка цен на топливо* оперативное проведение ни к чему, конечно, понятно, что установить цены будущей датой это нонсенс, но и не факт, что такое не может

произойти. Например, для каких-нибудь прогнозов. Поэтому для данного документа запретим оперативное проведение.

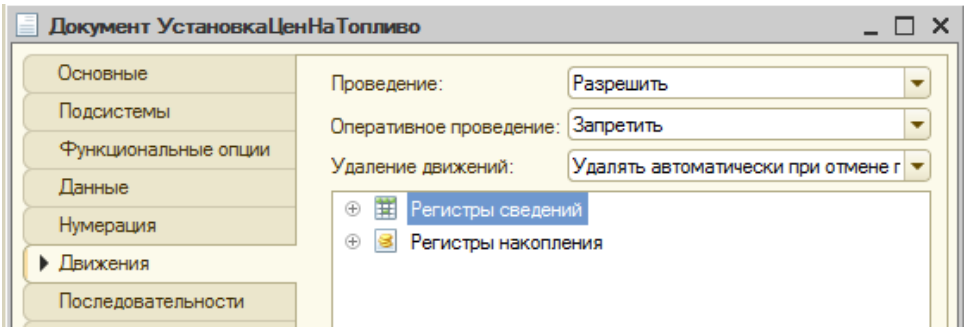


Рис. 6.2.7

Нумерация документов

Каждый документ должен содержать уникальный номер в пределах заданного периода. На закладке *Нумерация* задаются параметры нумерации документов. Откройте данную закладку у документа *Печать путевых листов.*

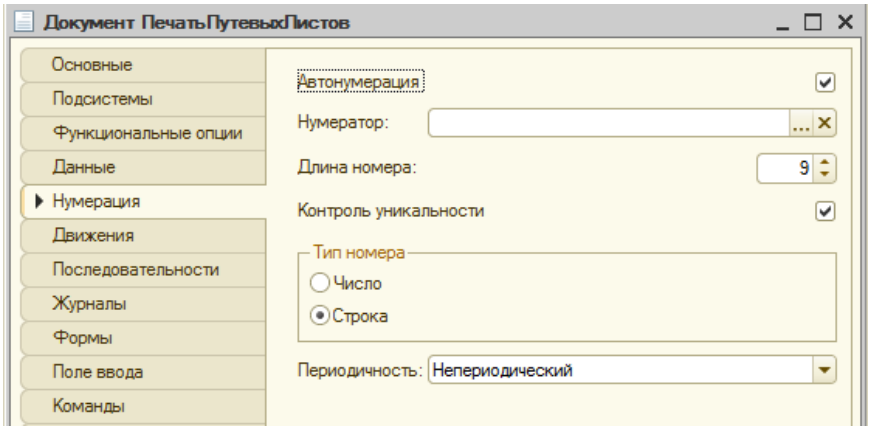


Рис. 6.2.8

В ней обращаем внимание на флажок *Автонумерация*.

Если он установлен, то платформа сама присваивает номера документов. Рекомендую его устанавливать, если Вы сами не хотите присваивать номер документам по определенному алгоритму (разумеется, программно). Про длину номера и тип номера объяснять не буду, это все просто и понятно. *Контроль уникальности* ставьте в том случае, если необходимо запрещать создавать два и более документа с одинаковыми номерами. *Периодичность* указывает, в каком периоде будет осуществляться контроль уникальности. Если установлено свойство *Автонумерация*, то при наступлении следующего периода нумерация автоматически начнется сначала.

Обычно в организациях хотят, чтобы все номера документов со следующего года начинались сначала. Поэтому есть смысл в документах устанавливать для свойства *Периодичность* значение *В пределах года*.

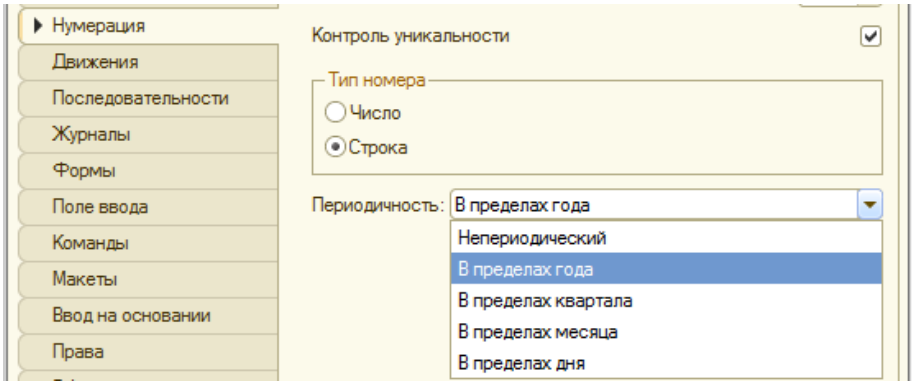


Рис. 6.2.9

Установите это значение для всех документов конфигурации.

И последний момент в нумерации документов, который нас заинтересует, это *Нумератор*. Как Вы уже заметили из работы с программой, все документы нумеруются по отдельности. Но иногда необходимо иметь сквозную нумерацию. К примеру, если нам надо, чтобы мы ввели документ поступления под номером один, потом еще один документ поступления под номером два, а потом документ реализации под номером три, который будет являться продолжением предыдущих двух номеров. Для этого в программе 1С можно использовать нумераторы, они позволяют осуществлять сквозную нумерацию любого количества документов.

Рассмотрим сквозную нумерацию на примере документов *Прибытие в гараж* и *Выбытие из гаража*. Для этого необходимо создать нумератор: перейдите в ветвь *Документы* дерева конфигурации, раскройте, выберите нумераторы и нажмите *Добавить*.

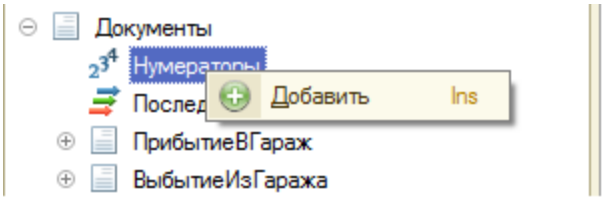


Рис. 6.2.10

Нумератор автоматически будет создан, и слева откроются его свойства. Дайте ему название *«Основной»*. Периодичность установим в пределах года, все остальное оставим как есть.

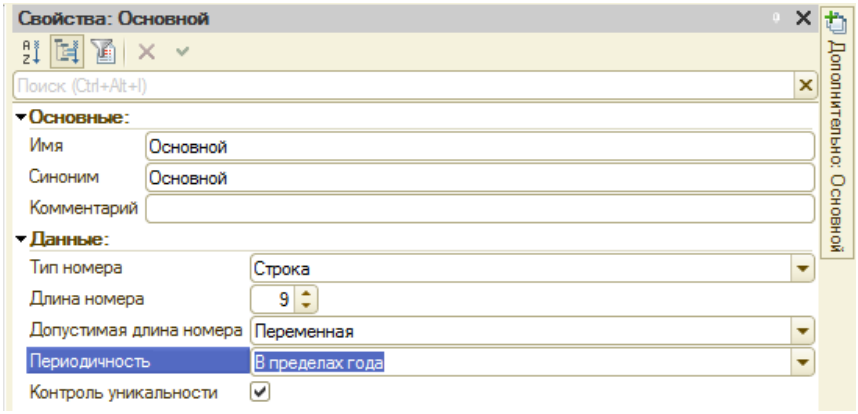


Рис. 6.2.11

Следующим шагом зайдите в документ *Прибытие в гараж*, и на закладке *Нумерация* выберете вновь созданный нумератор.

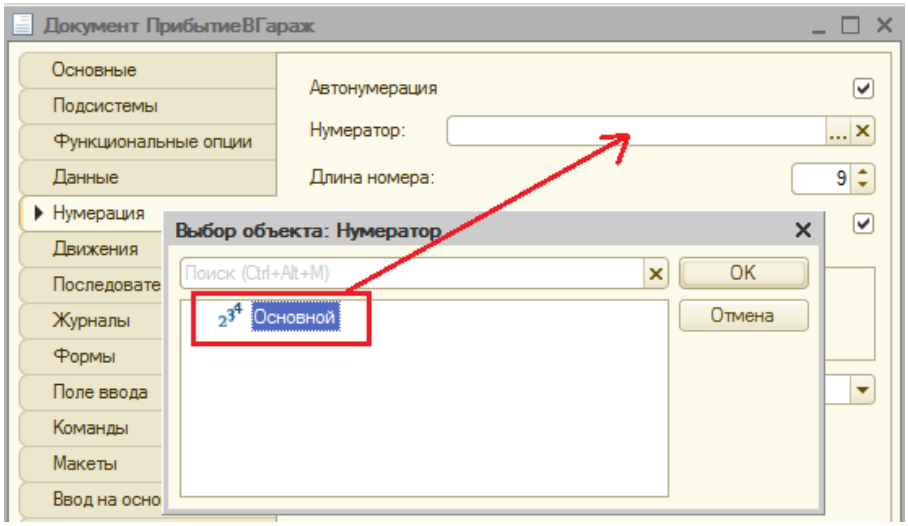


Рис. 6.2.12

То же самое сделайте с документом *Выбытие из гаража*.

Сохраните конфигурацию, если есть уже созданные документы *Прибытие в гараж* и *Выбытие из гаража,* то удалите их, создайте заново и посмотрите, как будет осуществляться нумерация вновь введенных документов (см. рис. 6.2.13).

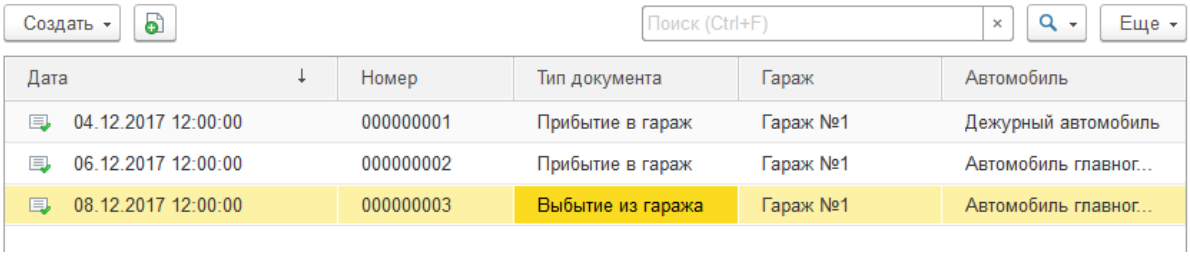


Рис. 6.2.13

На этом мы закончим с конфигурированием документов. Перейдем к изучению свойств и методов документов.

Проведен

Большинство закрытых свойств и методов у документов совпадает со свойствами и методами справочников, которые мы проходили в первой части этой главы. Но есть такие свойства и методы, которые присутствуют только у документов.

Рассмотрим свойство *Проведен.*

Проведен - это свойство документа, которое принимает значение *Истина*, если документ проведен, и *Ложь*, если документ не проведен. Данное свойство доступно только для чтения.

Рассмотрим простой пример: создайте обработку, имеющую один реквизит с типом ссылка на документ *Прибытие в гараж,* и один реквизит с типом *Булево*. Если выбираемый документ будет проведен, то тумблер, соответствующий реквизиту, установится в *Да*, в ином случае - *Нет*.

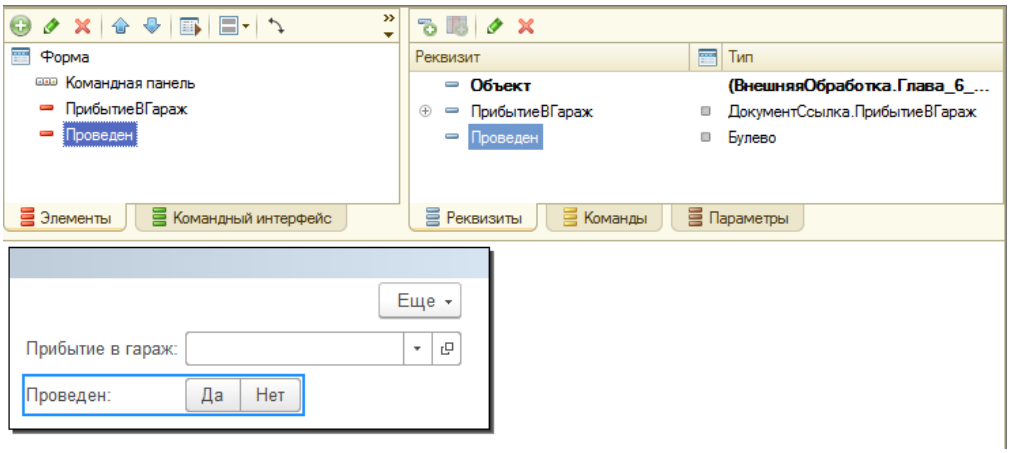


Рис. 6.2.14

Для обработчика *ПриИзменении* элемента *ПрибытиеВГараж* создадим обработчики на клиенте и на сервере.

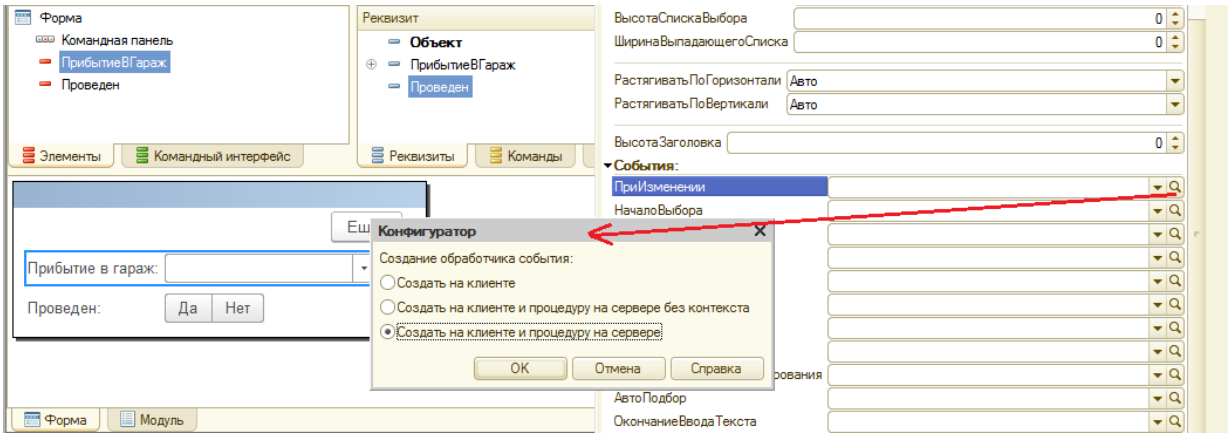


Рис. 6.2.15

В этих обработчиках напишем следующий код:

```
&НаСервере
Процедура ПрибытиеВГаражПриИзмененииНаСервере()
    Проведен = ПрибытиеВГараж.Проведен;
КонецПроцедуры

&НаКлиенте
Процедура ПрибытиеВГаражПриИзменении(Элемент)
    ПрибытиеВГаражПриИзмененииНаСервере();
КонецПроцедуры
```

Листинг 6.2.1

В этом коде мы при изменении значения поля *Прибытие в гараж* булевому реквизиту формы *Проведен* присваиваем значение свойства *Проведен* объекта *Документ*, которое тоже имеет тип *Булево*. К этому свойству мы обратились посредством ссылки на данный объект.

Сохраните и запустите обработку. Выберете теперь любой проведенный документ.

А теперь отменим его проведение.

Тумблер должен меняться в зависимости от того, проведен документ или нет.

Программное создание, запись и проведение документа

Разберем, как программно создать, сохранить и провести документ.

Для этого мы изучим методом *СоздатьДокумент* менеджера объекта нужного документа. Данная функция возвращает новый экземпляр объекта. После создания документа разработчик должен заполнить реквизиты этого документа по своему усмотрению, и в конце, при помощи метода объекта *Записать*, сохранить документ.

Разберем все это на практике. Разработаем обработку, которая будет создавать программно нужный нам документ *Прибытие в гараж*. Создайте обработку, форму и на форме

обработки разместите следующие реквизиты: *Автомобиль, Гараж* и *Дата прибытия* и *Пробег* с соответствующими типами.

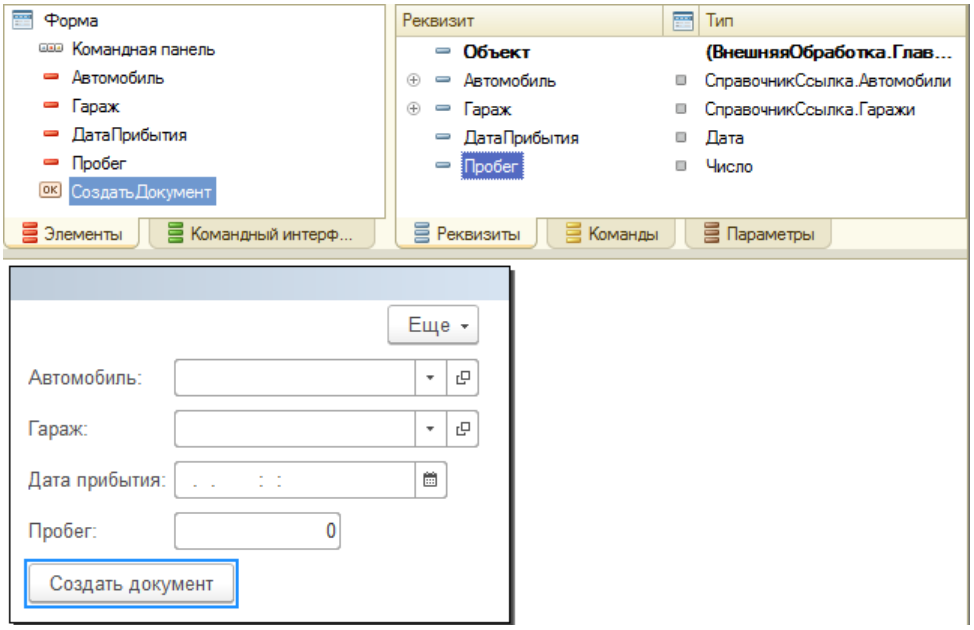


Рис. 6.2.16

В том случае, если все поля заполнены, программа создаст и запишет новый документ, в реквизиты которого будут внесены значения данных полей.

Для команды «Создать документ» создадим обработчики, в которых напишем следующий код:

```
&НаСервере
Процедура СоздатьДокументНаСервере()

    ДокОбъект = Документы.ПрибытиеВГараж.СоздатьДокумент();
    ДокОбъект.Дата = ТекущаяДата();
    ДокОбъект.Автомобиль = Автомобиль;
    ДокОбъект.Гараж      = Гараж;
    ДокОбъект.ДатаПрибытия = ДатаПрибытия;
    ДокОбъект.Пробег = Пробег;
    ДокОбъект.Записать(РежимЗаписиДокумента.Проведение,
                       РежимПроведенияДокумента.Оперативный);


КонецПроцедуры



&НаКлиенте
Процедура СоздатьДокумент(Команда)

    Если Не ЗначениеЗаполнено(Автомобиль) Тогда
         Сообщить("Не заполнили автомобиль");
         Возврат;
    КонецЕсли;
```

```
    Если Не ЗначениеЗаполнено(Гараж) Тогда
        Сообщить("Не заполнили гараж");
        Возврат;
    КонецЕсли;
    Если Не ЗначениеЗаполнено(ДатаПрибытия) Тогда
        Сообщить("Не заполнили дату прибытия");
        Возврат;
    КонецЕсли;
    Если Не ЗначениеЗаполнено(Пробег) Тогда
        Сообщить("Не заполнили пробег");
        Возврат;
    КонецЕсли;

    СоздатьДокументНаСервере();
КонецПроцедуры
```

Листинг 6.2.2

В процедуре *СоздатьДокумент* мы используем метод глобального контекста *ЗначениеЗаполнено*. Он возвращает *Истину*, если переданное в него значение заполнено, и *Ложь*, если нет. Т.е. если мы передадим в метод пустую ссылку на справочник или документ, или пустую дату, или пустую строку, то он вернет *Ложь*.

Разберем код в процедуре *СоздатьДокументНаСервере*.

В первой строке при помощи метода *СоздатьДокумент* мы создали экземпляр объекта. Этот экземпляр объекта содержится в переменной *ДокОбъект*, которая имеет тип *ДокументОбъект.ПрибытиеВГараж*. В последующих пяти строках присвоили значения реквизитам объекта.

*Стандартный реквизит документа **Дата** должен быть всегда заполнен!*

И в последней строке мы записываем документ. Разберем последнюю строку подробнее. Как Вы видите, в коде у процедуры *Записать* объекта *Документ* два параметра.

Первый параметр - это режим записи. *Режим записи документа* - это системное перечисление. Всего существуют три режима записи документа: запись, отмена проведения и проведение. Объяснять каждый режим не надо, поскольку названия их интуитивно понятны. Если данный параметр не будет установлен, то документ будет просто записан без проведения.

Рассмотрим следующий параметр – *Режим проведения документа*. Что такое режим проведения, мы уже знаем. В данном параметре мы устанавливаем, в каком режиме будет проведен данный документ. Обращаю Ваше внимание, что оперативный режим можно установить только в том случае, если у данного документа разрешено оперативное проведение. По умолчанию данный параметр принимает значение неоперативный.

Самостоятельно посмотрите на выпадающий список параметра режима проведения и режима записи.

Как Вы уже поняли из примера, нет отдельного метода для проведения документа. Если Вам понадобится провести непроведенные документы, то нужно вызвать экземпляр их объекта с помощью метода *ПолучитьОбъект* и записать его с режимом записи документа *Проведение*.

А также наоборот, если Вам нужно отменить проведение документа, то нужно записать документ с режимом записи *Отмена проведения*.

Обратите внимание, если режим проведения установлен Оперативный, то в реквизит Дата должна быть записана или текущая дата, или дата не меньшая даты последнего оперативно проведенного документа.

Изменим нашу обработку: создадим два переключателя, в одном будут перечислены режимы проведения документов, а в другом режимы записей. Для этого создадим два реквизита формы с типом число (целое). Разместим эти реквизиты на форме и вид поля у обоих установим *Поле переключателя*.

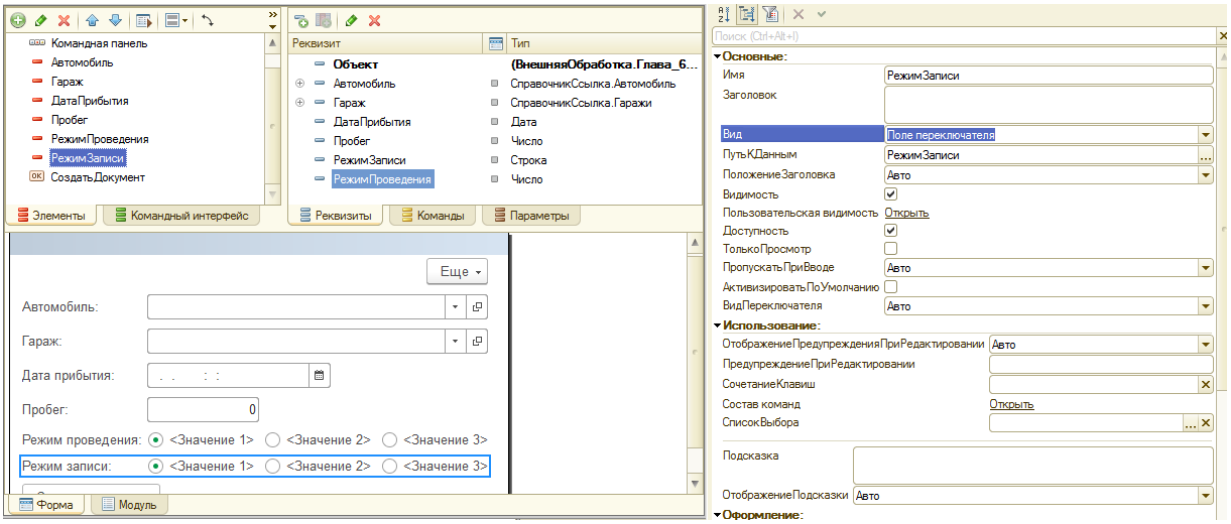


Рис. 6.2.17

У поля *Режим проведения* заполним список выбора двумя значениями: *Оперативный* и *Неоперативный.* Для первого значение будет 1, а для второго – 2.

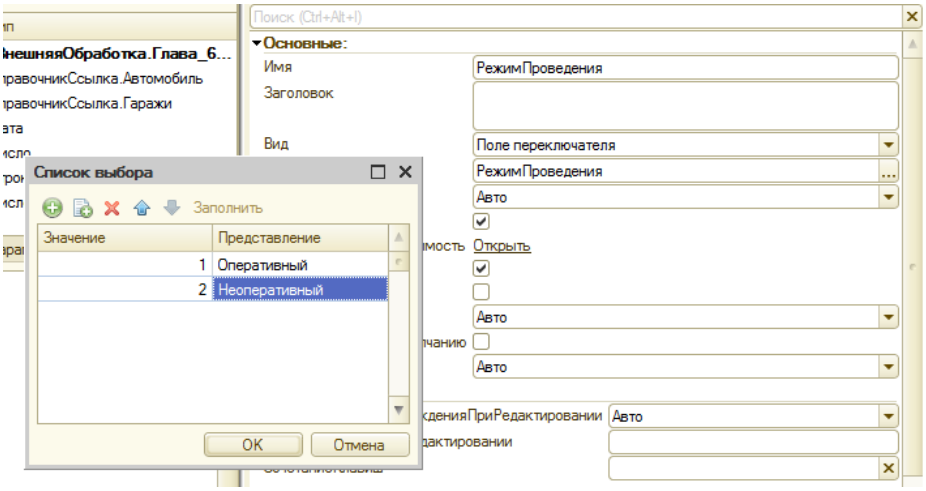


Рис. 6.2.18

А для поля *Режим записи* заполним список выбора двумя значениями: *Запись, Проведение.* Для первого значение будет 1, а для второго – 2.

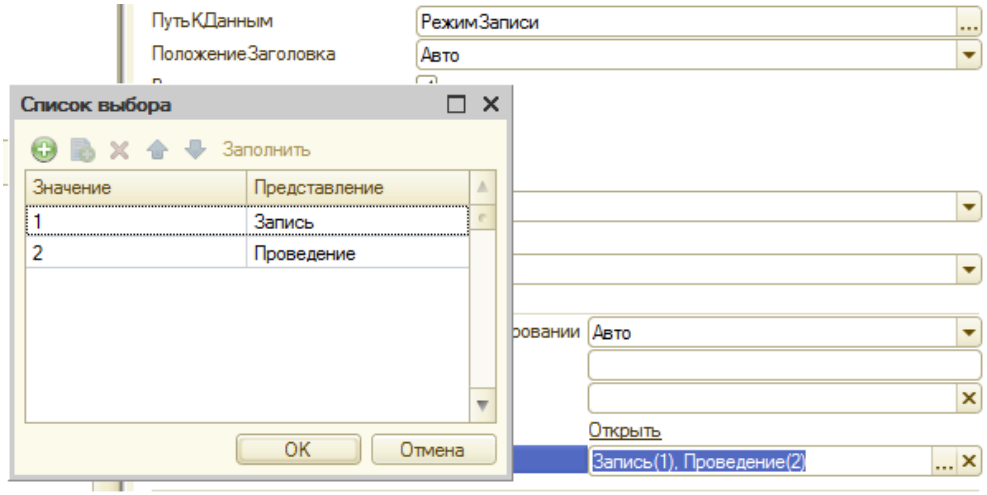


Рис. 6.2.19

Всем полям переключателям установим вид тумблер.

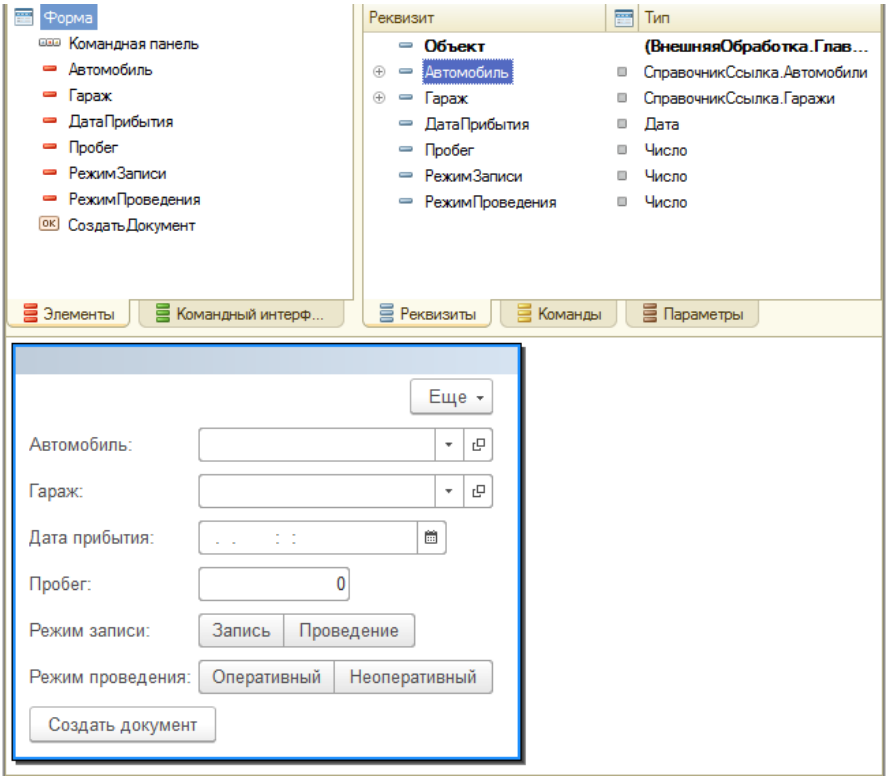


Рис. 6.2.20

Работу с переключателями на форме мы изучали в предыдущей главе, в этом примере мы научимся программно работать с ними.

Немного доработаем код в обработчиках команды «Создать документ».

```
&НаСервере
Процедура СоздатьДокументНаСервере()

    ДокОбъект = Документы.ПрибытиеВГараж.СоздатьДокумент();
    ДокОбъект.Автомобиль = Автомобиль;
    ДокОбъект.Гараж      = Гараж;
    ДокОбъект.ДатаПрибытия = ДатаПрибытия;
    ДокОбъект.Пробег = Пробег;
    Если РежимЗаписи = 1 Тогда
        ДокОбъект.Дата = ДатаПрибытия;
        ДокОбъект.Записать(РежимЗаписиДокумента.Запись);
    ИначеЕсли РежимЗаписи = 2 Тогда
        Если РежимПроведения = 1 Тогда
            ДокОбъект.Дата = ТекущаяДата();
            ДокОбъект.Записать(РежимЗаписиДокумента.Проведение,
            РежимПроведенияДокумента.Оперативный);
        ИначеЕсли РежимПроведения = 2 Тогда
            ДокОбъект.Дата = ДатаПрибытия;
            ДокОбъект.Записать(РежимЗаписиДокумента.Проведение,
            РежимПроведенияДокумента.Неоперативный);
        КонецЕсли;
    КонецЕсли;
КонецПроцедуры




&НаКлиенте
Процедура СоздатьДокумент(Команда)

    Если Не ЗначениеЗаполнено(Автомобиль) Тогда
         Сообщить("Не заполнили автомобиль");
         Возврат;
    КонецЕсли;
    Если Не ЗначениеЗаполнено(Гараж) Тогда
         Сообщить("Не заполнили гараж");
         Возврат;
    КонецЕсли;
    Если Не ЗначениеЗаполнено(ДатаПрибытия) Тогда
         Сообщить("Не заполнили дату прибытия");
         Возврат;
    КонецЕсли;
    Если Не ЗначениеЗаполнено(Пробег) Тогда
         Сообщить("Не заполнили пробег");
         Возврат;
    КонецЕсли;
    Если РежимЗаписи = 0 Тогда
         Сообщить("Установите режим записи");
         Возврат;
    КонецЕсли;
    Если РежимПроведения = 0 Тогда
         Сообщить("Установите режим проведения");
         Возврат;
    КонецЕсли;
    СоздатьДокументНаСервере();
КонецПроцедуры
```

Листинг 6.2.3

Как видно из листинга 6.2.3, в использовании переключателей нет ничего сложного: просто у реквизитов формы *РежимЗаписи* и *РежимПроведения* во время отработки кода те значения, которые соответствуют выбранному в текущий момент значению переключателя.

В процедуре *СоздатьДокументНаСервере* в условии проверяем значение переключателя *РежимЗаписи* и если оно равно 1, что соответствует установленному тумблеру «Запись», то просто записываем объект, при этом дате документа присваивается значение реквизита *ДатаПрибытия*. Если реквизит *РежимЗаписи* равен 2, то значит проводим документ. В этом случае все зависит от значения реквизита *РежимПроведения*, если оно равно 1 (оперативное проведение), то проводим в оперативном режиме, а иначе - в неоперативном.

Найти документ

Узнаем, каким образом в программе 1С можно найти какой-нибудь конкретный документ и впоследствии изменить его.

Для поиска документа имеются два метода менеджера объекта *Документ*. Это *НайтиПоНомеру* и *НайтиПоРеквизиту*.

Метод *НайтиПоНомеру* осуществляет поиск документа по номеру. Разберем сначала синтаксис данного метода, а после посмотрим его применение на практическом примере.

НайтиПоНомеру(<НомерДокумента>, <ДатаИнтервала>)

Рассмотрим параметры метода. Первый параметр это номер документа, по которому осуществляется поиск в базе. Тут все понятно.

Второй параметр используется только для документов с периодической нумерацией (что это, см. выше). Поиск будет осуществляться по тому интервалу, в который входит эта дата. К примеру, если номера документов уникальны в пределах квартала, а дата интервала 15 марта 2013 года, то поиск будет вестись в интервале дат от 1 января 2013 года до 31 марта 2013 года.

Данный параметр не обязателен, но если для документов с периодической нумерацией он не будет установлен, то поиск будет вестись с пустой датой, и на выходе всегда будет пустая ссылка. Поэтому если Вам нужно искать документ в текущем периоде, всегда устанавливайте текущую дату.

Данный метод возвращает ссылку на документ, а если по искомому номеру нет документов, то возвращается пустая ссылка. Если параметр *НомерДокумента* пустой или равен нулю, то возвращается *Неопределено*.

Разберем этот метод: сделаем обработку, в которой пользователь вводит вручную номер документа и при нажатии на кнопку осуществляется поиск документа.

Создайте обработку, форму и на форме два реквизита, один текстовый с длиной текста *9* символов, а второй имеет тип значения *Ссылка на документ Прибытие в гараж*. А также команду «Найти документ», которую разместите на форме.

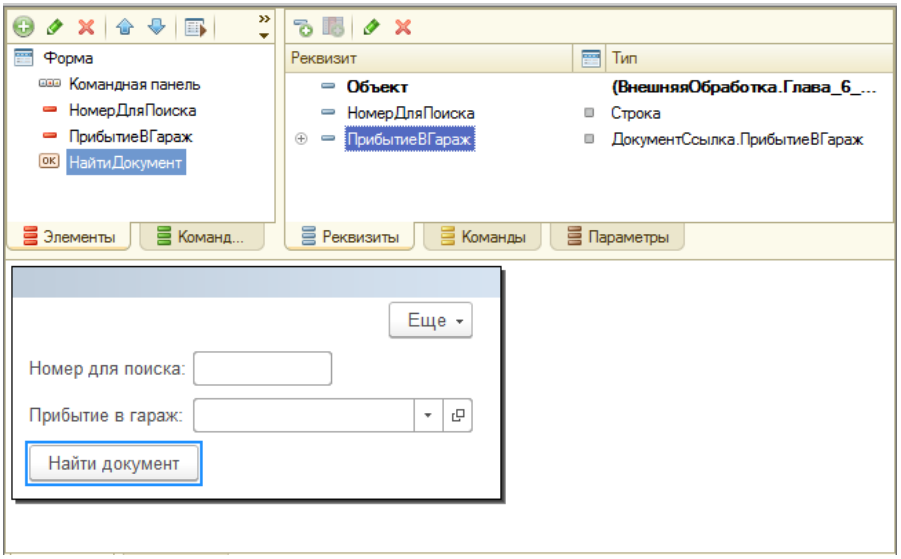


Рис. 6.2.21

В обработчиках команды *«Найти документ»* напишите следующий код:

```
&НаСервере
Процедура НайтиДокументНаСервере()

    ПрибытиеВГараж =
Документы.ПрибытиеВГараж.НайтиПоНомеру(НомерДляПоиска,ТекущаяДата());

КонецПроцедуры

&НаКлиенте
Процедура НайтиДокумент(Команда)
    НайтиДокументНаСервере();
КонецПроцедуры
```

Листинг 6.2.4

Поскольку у документа *Прибытие в гараж* установлена периодическая нумерация, то вторым параметром мы указали текущую дату.

Посмотрите, как работает данная обработка.

Рассмотрим второй метод поиска документа, это *НайтиПоРеквизиту*.

Данный метод осуществляет поиск документа по реквизиту, он аналогичен одноименному методу для справочников, с той разницей, что у него только два параметра: название реквизита и значение реквизита. Если документов по данному реквизиту несколько, то будет возвращен один из них - какой именно, точно предугадать невозможно. Данный метод на практике редко применяется, поэтому мы не будем рассматривать его подробно, а Вы в качестве факультатива можете переделать имеющийся поиск документа, в поиск по интересному Вам реквизиту.

Изменение имеющихся документов

Если Вам необходимо программно изменить имеющийся документ, то необходимо использовать знакомый Вам метод *ПолучитьОбъект*. Данный метод является методом ссылки документа и возвращает объект документа, на который указывает ссылка.

Доработаем предыдущую обработку: добавим на форму новый реквизит *Дата прибытия*, и если документ по номеру найден, то будем изменять одноименный реквизит в документе на данное значение, а после этого проводить документ в неоперативном режиме.

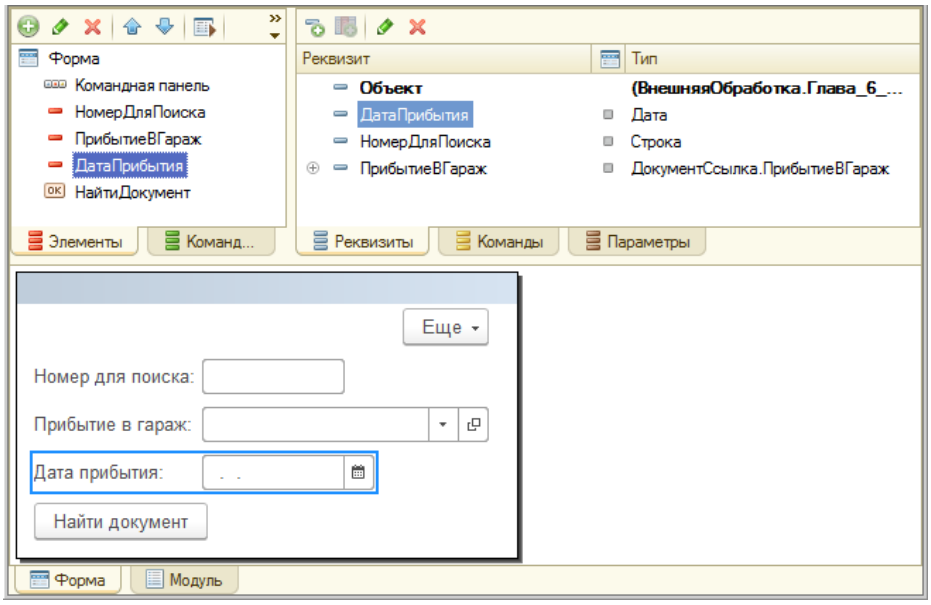


Рис. 6.2.22

Допишем код в обработчике:

```
&НаСервере
Процедура НайтиДокументНаСервере()
    ПрибытиеВГараж =
Документы.ПрибытиеВГараж.НайтиПоНомеру(НомерДляПоиска,ТекущаяДата());
    Если ПрибытиеВГараж.Пустая() Тогда
        Возврат;
    КонецЕсли;
    ПрибытиеВГаражОбъект = ПрибытиеВГараж.ПолучитьОбъект();
    ПрибытиеВГаражОбъект.ДатаПрибытия = ДатаПрибытия;
    ПрибытиеВГаражОбъект.Записать(РежимЗаписиДокумента.Проведение,
                                  РежимПроведенияДокумента.Неоперативный);
КонецПроцедуры
```

Листинг 6.2.5

В данном коде мы получили объект из ссылки на документ, заменили дату прибытия и записали его с проведением, в неоперативном режиме проведения.

Резюме

Во второй части этой главы мы существенно углубили свои знания в работе с документами, изучили нумерацию, научились создавать и изменять документы программным способом. Удаление документов и установка пометки на удаление полностью аналогично соответствующим действиям для справочников. Поупражняйтесь в программном удалении документов самостоятельно.

Часть 3. Форма как объект

Любая форма, как и любой элемент на форме, является объектом. Это значит, что у формы и у элементов формы имеются методы и свойства. А также мы можем работать с событиями, которые есть у формы, и с событиями, которые есть у элементов формы.

Основной реквизит

Прежде чем начать изучать свойства, методы и события форм, изучим такое понятие, как *основной реквизит формы*. Для любой формы можно назначить *основной реквизит*. Обычно он назначается автоматически при создании формы с помощью конструктора. Основной реквизит формы определяет стандартную функциональность формы. Т.е. от основного реквизита зависит, какие у формы будут стандартные функции. Например, если основной реквизит формы *ДокументОбъект*, то у формы будут такие стандартные команды, как «Проведение» и «Отмена проведения» (если у документа не запрещено проведение по регистрам).

У формы элемента справочника основной реквизит имеет соответствующий тип *СправочникОбъект* (см. рис. 6.3.1). А формы документа основной реквизит имеет соответствующий типа *ДокументОбъект* (см. рис. 6.3.2). Т.е. при открытии формы в этот реквизит загружается экземпляр объекта справочника или документа.

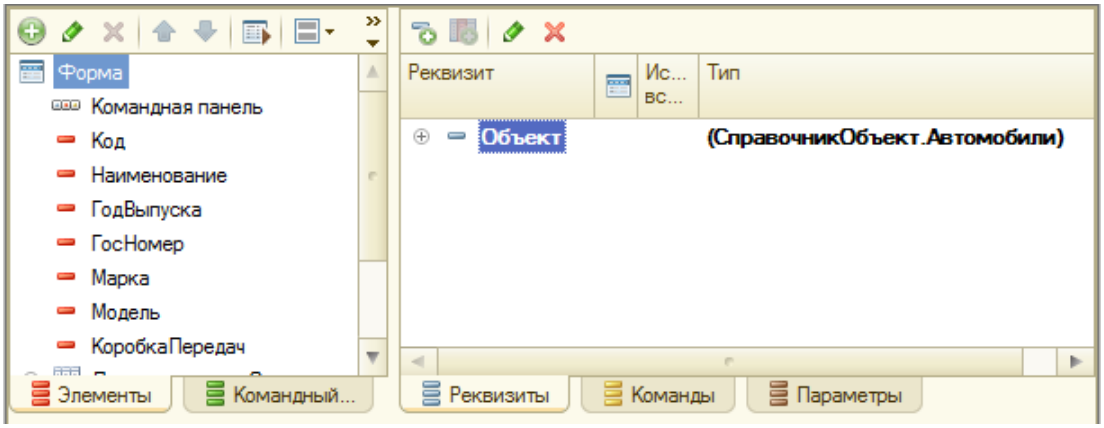


Рис. 6.3.1

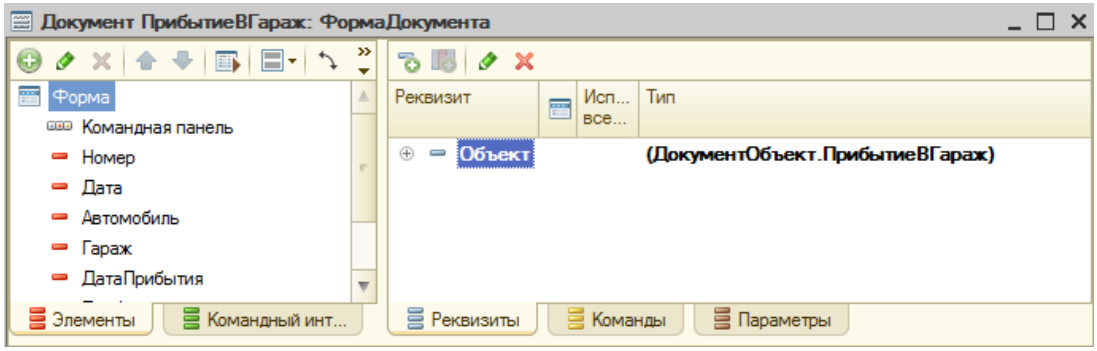


Рис. 6.3.2

Обычно основной реквизит всегда выделяется в списке жирным, а также у него установлен признак «Основной реквизит» (см. рис. 6.3.3).

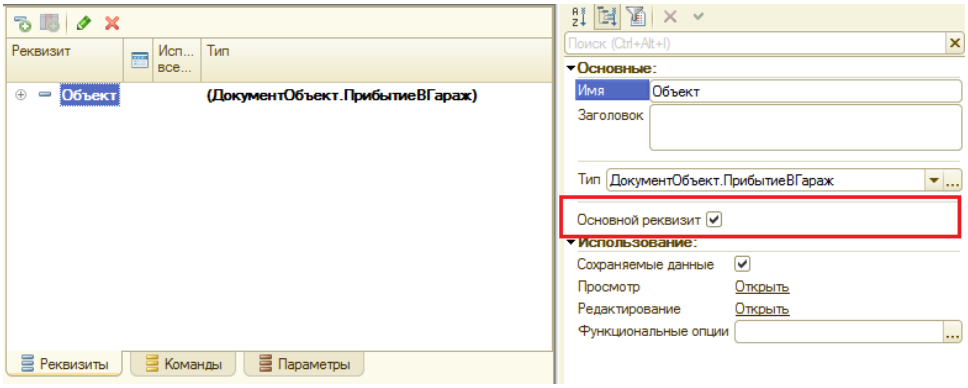


Рис. 6.3.3

Каким образом заполняется основной реквизит с типом «ДокументОбъект» или «СправочникОбъект» мы узнаем, когда будем изучать открытие формы.

Данные формы

Если Вы внимательно посмотрите на тип основного реквизита формы документа и вспомните то, о чем мы говорили в предыдущей главе про клиент-серверную архитектуру управляемой формы, то у Вас возникнет небольшой диссонанс: «Как так? Тип основного реквизита самый что ни на есть тяжелый, но он спокойно размещен на форме?» В действительности основной реквизит документа имеет тип *ДокументОбъект.<НазваниеДокумента>*, но на форму он загружается совсем под другим типом. Под типом *ДанныеФормыСтруктура*. Для того, чтобы убедиться в этом, установим точку останова в конце команды *ВывестиСообщение* на форме документа *Прибытие в гараж*, которую мы создали ранее, и посмотрим в отладке, какой тип имеет реквизит формы «Объект» (см. рис. 6.3.4).

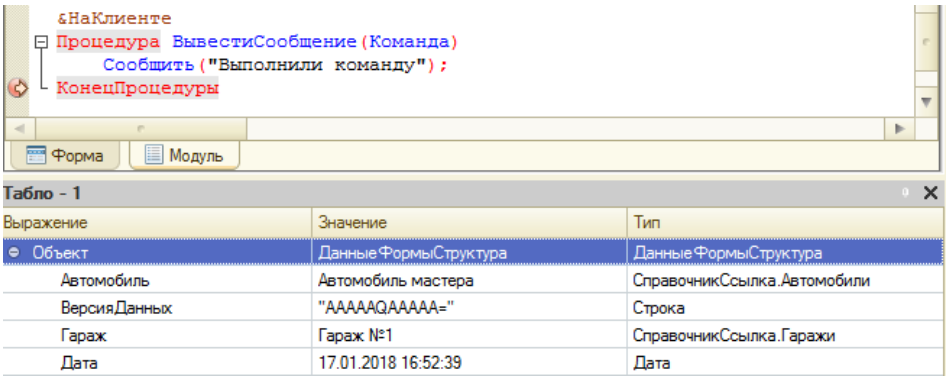


Рис. 6.3.4

Как видно из рисунка 6.3.4, основной реквизит формы *Объект* имеет тип *ДанныеФормыСтруктура* (структуры мы будем изучать в следующей части), и что самое важное, через эту структуру мы можем обращаться к реквизитам того объекта, который сейчас загружен. Мы можем их читать и даже, в принципе, можем их изменять.

Более подробно работа с данными формами разбирается во второй главе третьей части книги «Основы разработки в 1С: Такси».

На данном этапе обучения мы не будем глубоко изучать работу с данными формами. Просто пока уясните, что при открытии форм документов и элементов справочников данные соответствующего объекта загружаются в структуру, посредством которой эти данные можно и читать, и изменять. Для подтверждения этого тезиса выполним небольшую задачу: на форме документа «Прибытие в гараж» установим команду «Установить текущую дату прибытия» (см. рис. 6.3.5). Для этой команды создадим обработчик на клиенте, в котором будем проверять, заполнена ли дата прибытия. Если не заполнена, то присвоим ей текущую дату. Если заполнена, но день, месяц и год не соответствует текущим, то опять присвоим ей текущую дату. А иначе - ничего не делаем.

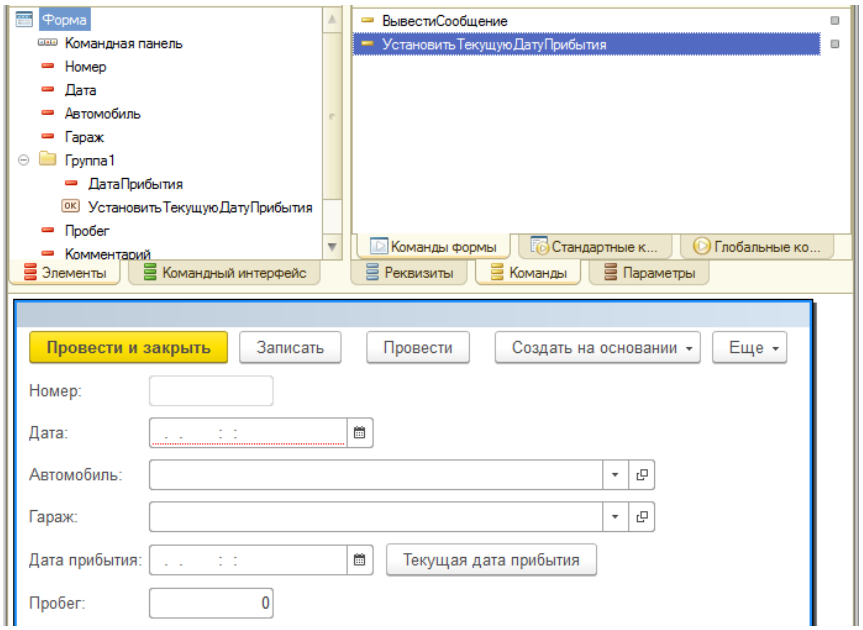


Рис. 6.3.5

Создадим для команды обработчик *на клиенте* и напишем в нем следующий код:

```
&НаКлиенте
Процедура УстановитьТекущуюДатуПрибытия(Команда)

    Если Не ЗначениеЗаполнено(Объект.ДатаПрибытия)  Тогда
        Объект.ДатаПрибытия = ТекущаяДата();
        Модифицированность = Истина;
    Иначе
        Если НачалоДня(Объект.ДатаПрибытия) <> НачалоДня(ТекущаяДата()) Тогда
            Объект.ДатаПрибытия = ТекущаяДата();
            Модифицированность = Истина;
        КонецЕсли;
    КонецЕсли;

КонецПроцедуры
```

Листинг 6.3.1

В этом коде мы читаем и изменяем реквизит *ДатаПрибытия* основного реквизита формы *Объект*. *Модифицированность* - это свойство формы (их бы будем изучать ниже), при установке этому свойству значения *Истина* на форме появится знак (*) и при закрытии формы будет задаваться вопрос о сохранении.

Доработайте данное задание: проверяйте, проведен ли текущий документ или нет, и если не проведен, то не изменяйте этот реквизит. Обратите внимание на работу в тонком клиенте! Возможно, потребуется вызов сервера для проверки этого признака!

Свойства, методы и события формы

Приступим к изучению свойств и событий формы. К свойствам и событиям формы можно обратиться как программно, так и посредством конфигуратора. Сначала разберем, как обратиться к свойствам формы посредством конфигуратора и как создавать события формы из конфигуратора. Откроем в конфигураторе 1С произвольную форму какого-нибудь документа, пусть это будет документ «Прибытие в гараж». В конфигураторе к свойствам формы и событиям можно обращаться при помощи палитры свойств. Для того, чтобы открыть палитру свойств, необходимо в закладке «Элементы» выделить самый верх дерева элементов «Форма» и либо сделать двойной щелчок левой клавишей мышки, либо вызвать контекстное меню и выбрать пункт «Свойства» (см. рис. 6.3.6).

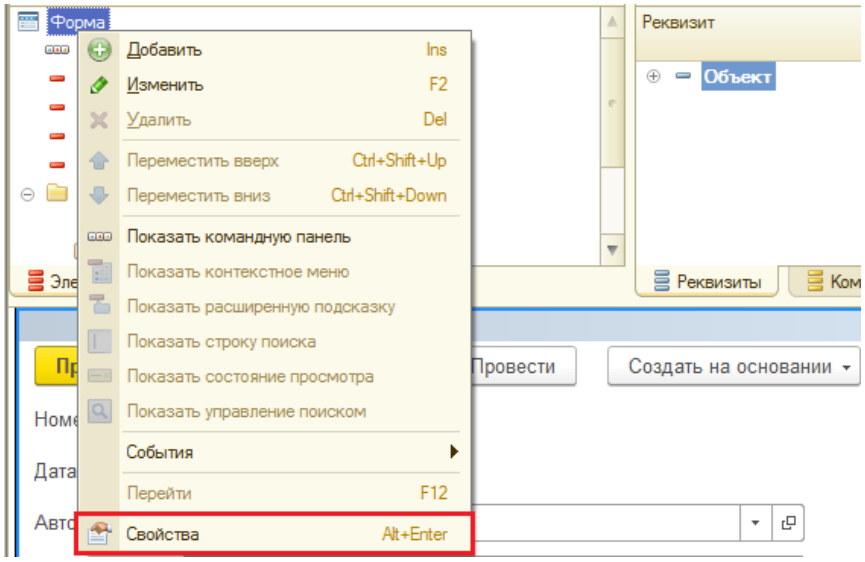


Рис. 6.3.6

После этих действий в правой части экрана должна открыться палитра свойств формы, где перечислены свойства и значения свойств текущей формы, а также события.

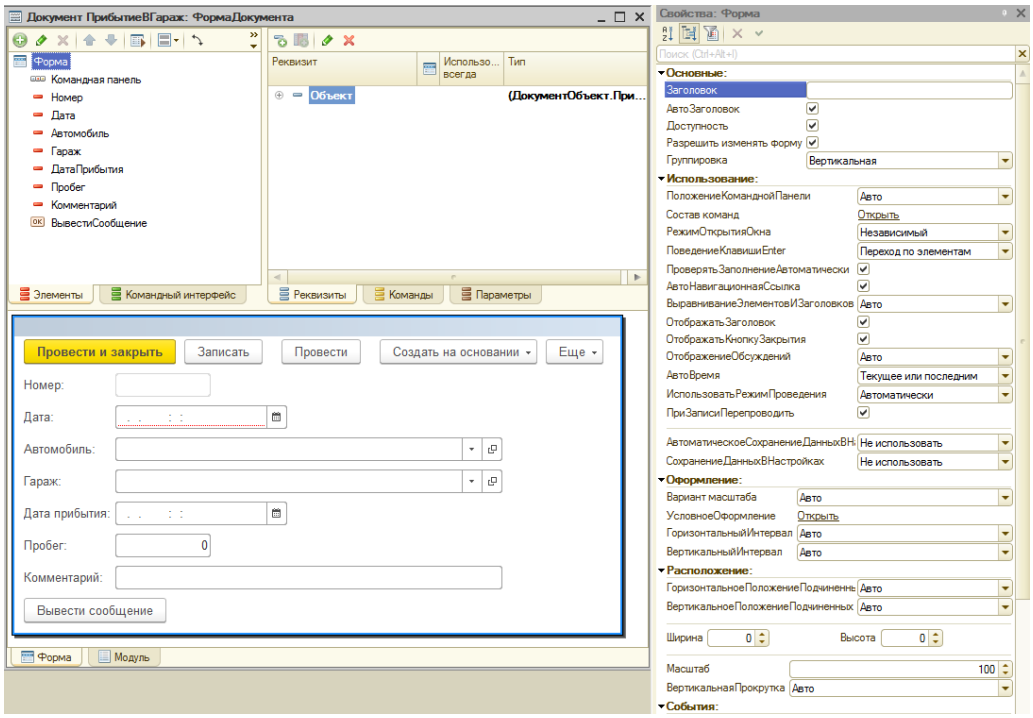


Рис. 6.3.7

Нет смысла разъяснять каждое свойство из палитры. В предыдущей главе мы разобрали свойство «Режим открытия окна» (см. стр. 298). Мы не будем разбирать каждое свойство в отдельности, я только расскажу, как работать с синтаксис-помощником. В синтаксис-помощнике все свойства управляемых форм находятся по пути «Интерфейс(управляемый) – Управляемая форма – Управляемая форма – Свойства» (см. рис. 6.3.8).

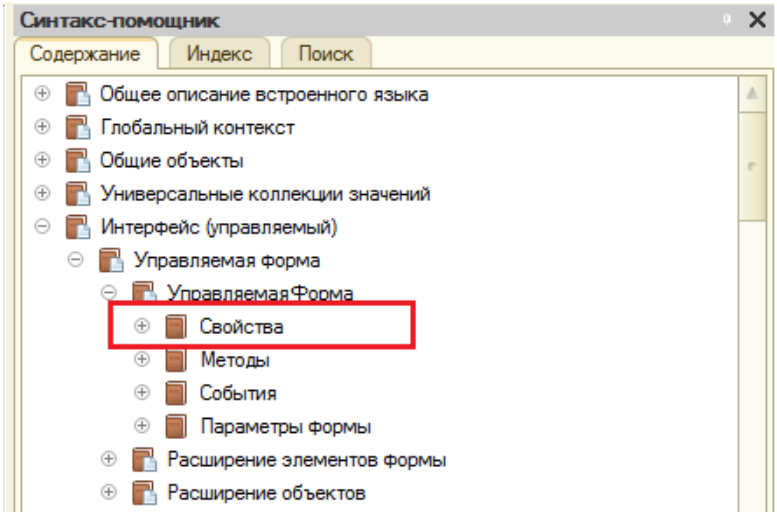


Рис. 6.3.8

В папке *Свойства* перечислены все свойства управляемой формы, причем их больше, чем в палитре свойств. В описании свойств следует особо обращать внимание на два пункта - это **Использование** и **Доступность**. Например, возьмем свойство *Заголовок* (см. рис. 6.3.9).

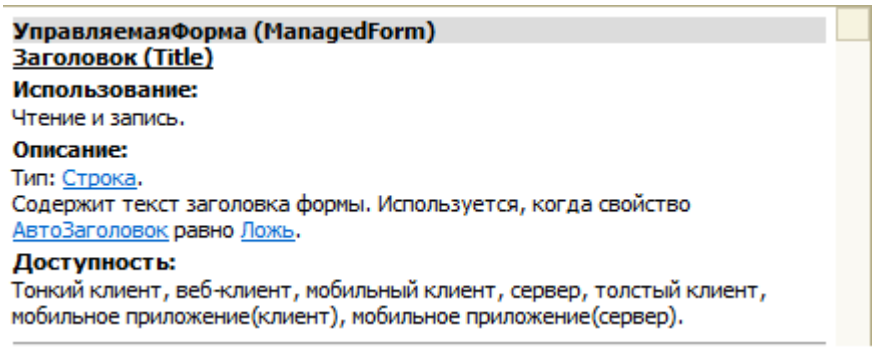


Рис. 6.3.9

Под пунктом **Использование** написано «Чтение и запись», это значит, что мы можем прочитать значение данного свойства, но также и изменить его. Причем как программно в процессе работы, так и в палитре свойств.

А под пунктом **Доступность** написано «Тонкий клиент, веб-клиент, ...». Это значит, что чтение и запись этого свойства доступны практически под всеми видами клиентов, включая самые «медленные» - тонкий клиент и веб-клиент.

А вот у свойства *ИмяФормы* под пунктом **Использование** написано «Только чтение» (см. рис. 6.3.10). Это значит, что мы сможем только прочитать это свойство, но нигде не сможем его изменить (программно).

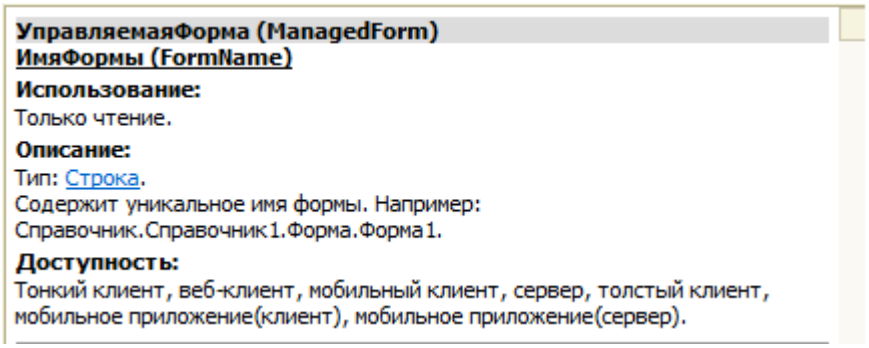


Рис. 6.3.10

Как я рассказывал в начале этой части, у каждой формы может быть один основной реквизит. Тип этого основного реквизита влияет на состав свойств, методов и событий форм. Для форм документов, у которых тип *ДокументОбъект,* будет один состав свойств, методов и событий, а для формы элементов справочника с типом *СправочникОбъект* - другой.

Причем описания свойств, методов и событий, которые есть у форм с *любым* основным реквизитом, находятся по уже знакомому нам пути «Интерфейс(управляемый) – Управляемая форма – Управляемая форма».

А свойства, методы и события управляемых форм с тем или иным типом основного реквизита называются *расширениями управляемой формы*. Они добавляются к стандартным свойствам, методам и событиям.

В синтаксис-помощнике описание расширений управляемой формы находятся в каталоге «Интерфейс(управляемый) – Управляемая форма» (см. рис. 6.3.11)

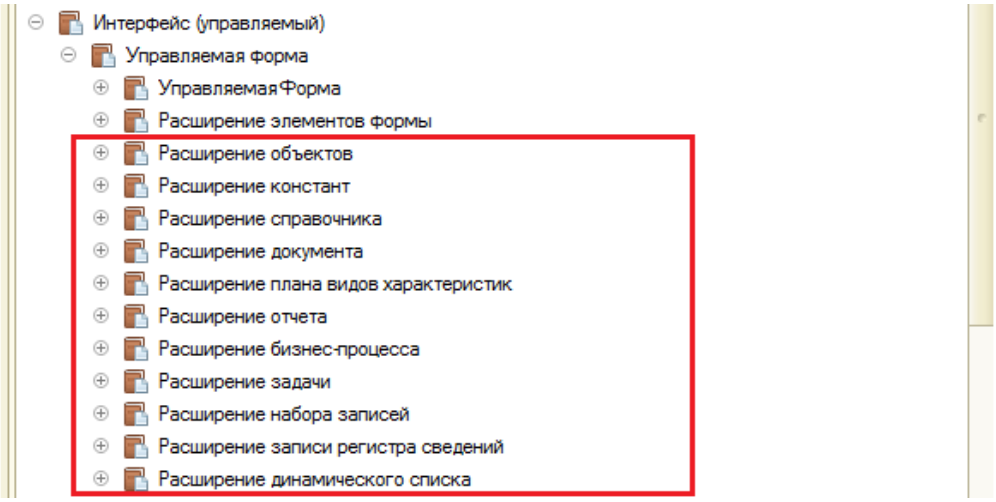


Рис. 6.3.11

Таким образом, полная информация о свойствах, методах и событиях формы документа содержится в трех папках справочной информации:

1) Интерфейс – Управляемая форма – Управляемая форма – типовые свойства и т.д. для всех управляемых форм;
2) Интерфейс – Управляемая форма – Расширение объектов – свойства и т.д. для форм, у которых основной реквизит какой-нибудь объектный тип (*СправочникОбъект*, *ДокументОбъект* и т.д.);
3) Интерфейс – Управляемая форма – Расширение документа – свойства и т.д. для форм, у которых основной реквизит объектный тип кого-нибудь документа.

Разберем события формы. Перечень всех событий управляемой формы располагается в палитре свойств внизу списка (см. рис. 6.3.12).

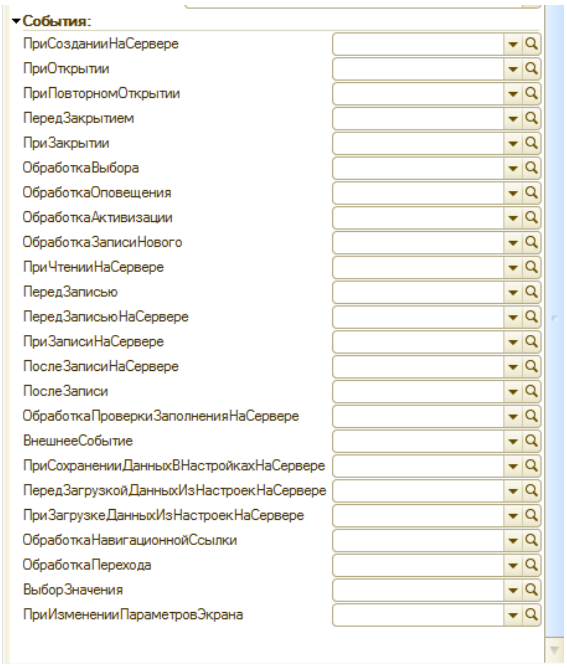


Рис. 6.3.12

Причем заметьте, само по себе событие ничего не значит, для того чтобы оно как-то работало, необходимо создать обработчик этого события – процедуру в модуле формы *на клиенте*, которая привяжется к соответствующему событию.

Для того, чтобы создать обработчик какого-нибудь события на форме, нужно кликнуть на пиктограмму «Лупа» рядом с этим событием, после этого Вам или будут предложены варианты расположения обработчиков (см. рис. 6.3.13), или событие сразу будет созданным (в основном, это касается событий, которые выполняются *на сервере*, например, *ПриСозданииНаСевере*).

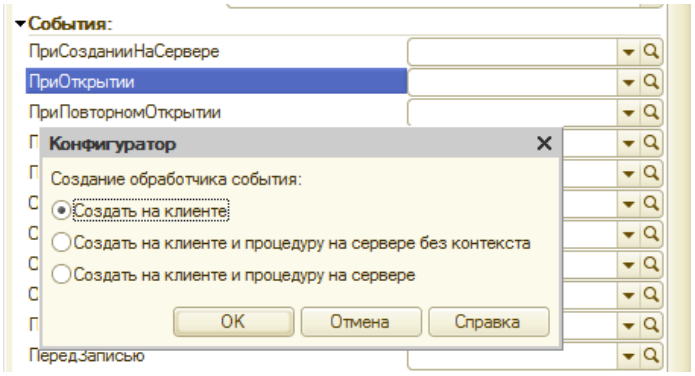


Рис. 6.3.13

С описанием всех событий формы также можно познакомиться в справочной информации. В том же самом разделе: «Интерфейс(управляемый) – Управляемая форма – Управляемая форма», только в папке *События* (см. рис. 6.3.14).

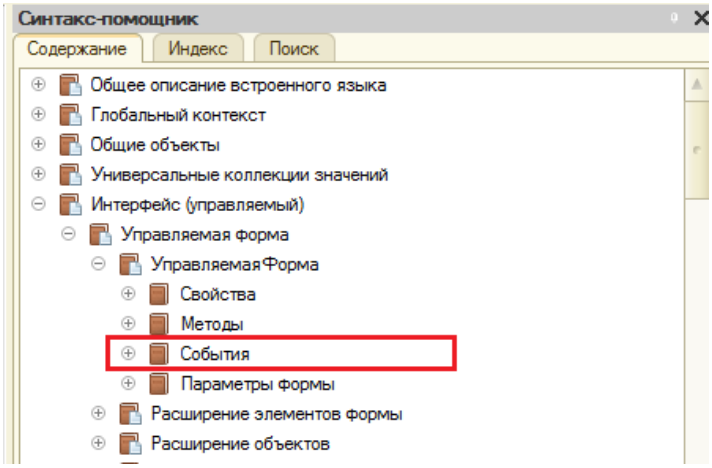


Рис. 6.3.14

В этой папке перечислены стандартные события для любой формы. События же расширений формы перечислены в соответствующих папках.

Научимся к свойствам формы обращаться непосредственно из формы. Для этого выполним небольшую задачу: при открытии формы документа «Прибытие в гараж» в заголовке формы будем выводить автомобиль и время прибытия.

Делать мы это будем в событии формы *ПриОткрытии*. Это событие вызывается после создания формы, но до её открытия. Но перед этим выключим свойство формы «АвтоЗаголовок» (самостоятельно узнайте, что это за свойство и за что отвечает).

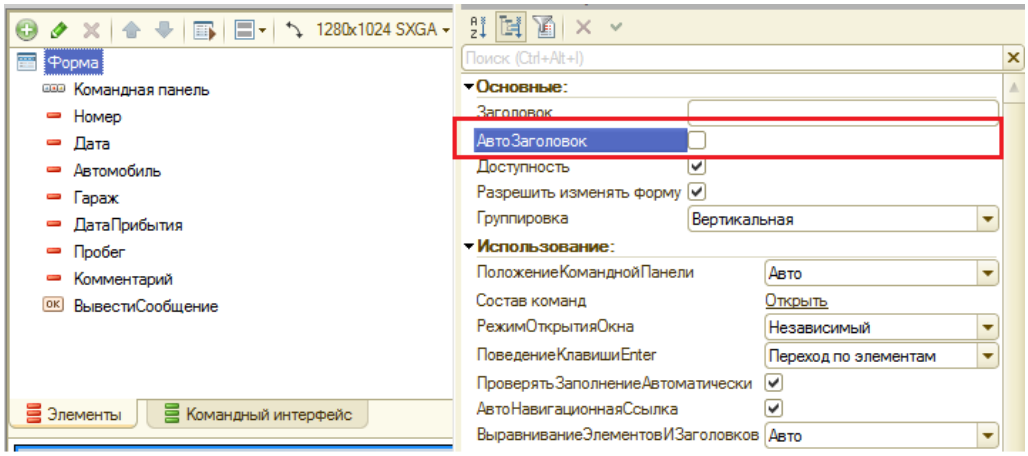


Рис. 6.3.15

Создадим обработчик для события формы «ПриОткрытии» на клиенте (см. рис. 6.3.16).

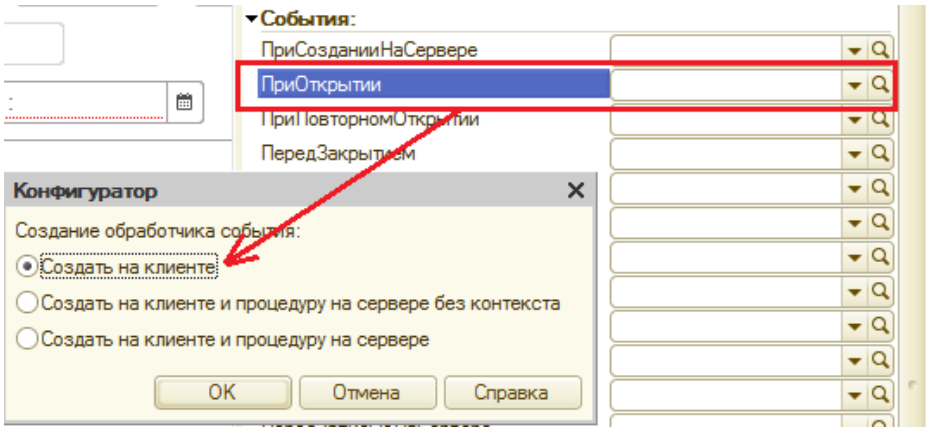


Рис. 6.3.16

В созданном обработчике напишем следующий код:

```
&НаКлиенте
Процедура ПриОткрытии(Отказ)

    Если ЗначениеЗаполнено(Объект.Автомобиль) и
ЗначениеЗаполнено(Объект.ДатаПрибытия) Тогда

        Заголовок = Строка(Объект.Автомобиль) +
                    " прибыл " +
                    Формат(Объект.ДатаПрибытия,"ДЛФ=DD");
    иначе
        АвтоЗаголовок = Истина;
    КонецЕсли;

КонецПроцедуры
```

Листинг 6.3.2

Разберем код в листинге 6.3.2.

Первое, обратите внимание, к свойствам формы можно обращаться напрямую просто написав это свойство. Не нужно писать никаких «ЭтаФорма» и т.д., просто пишете свойство в коде и с ним работаете.

В обработчике события формы «ПриОткрытии» мы первым делом проверяем наполненность реквизитов объекта *Автомобиль* и *Дата прибытия* при помощи уже знакомого нам метода *ЗначениеЗаполнено*. Обращаемся мы к этим реквизитам через основной реквизит формы *Объект*, который, как Вы должны помнить, имеет тип *ДанныеФормыКоллекция*. Если они не заполнены, то свойству «АвтоЗаголовок» присваиваем значение «Истина», и у нас будет выведен заголовок по умолчанию. А если они заполнены, то свойству *Заголовок* присваиваем определенный текст. При помощи функции преобразования значений *Строка* мы преобразуем ссылку на справочник *Автомобили* в её строковое значение. А с помощью функции *Формат* преобразуем дату в читаемый вид. Посмотрим, как теперь открывается форма уже существующего (см. рис. 6.3.18) документа и форма нового (см. рис. 6.3.17).

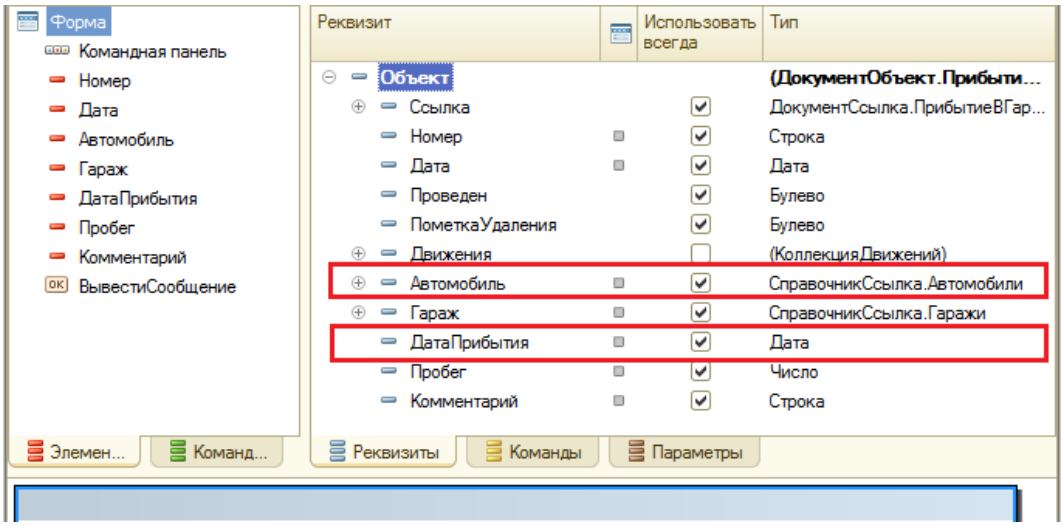


Рис. 6.3.17

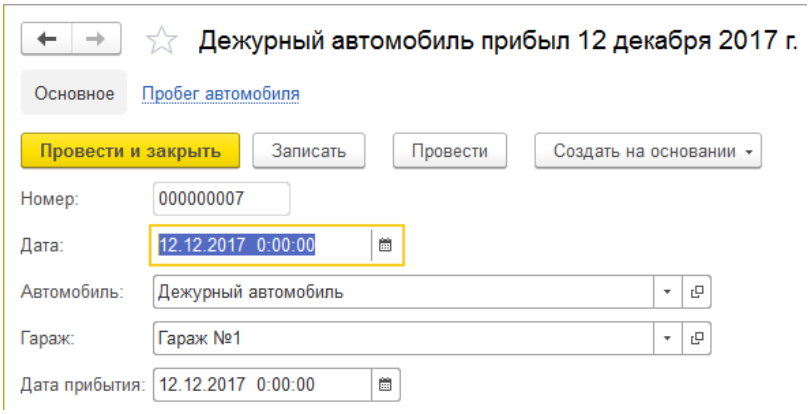


Рис. 6.3.18

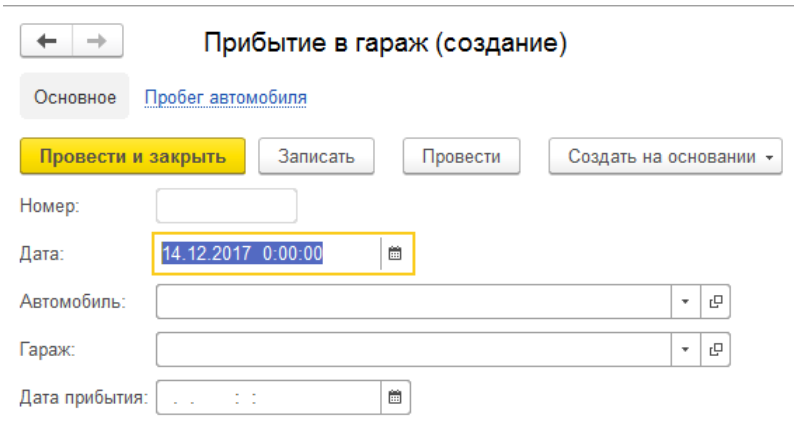


Рис. 6.3.19

Из всего множества методов форм рассмотрим только один - *РеквизитФормыВЗначение*. Как Вы помните, основной реквизит формы для элементов справочников и документов имеет тип

ДанныеФормыКоллекция, а что если нам нужно получить все-таки объектный тип? ДокументОбъект.<НазваниеДокумента> или СправочникОбъект.<НазваниеСправочника> ?

В основном обращение к объекту может понадобиться, если мы хотим получить доступ к каким-нибудь методам объекта (как закрытым, так и собственным). Не рекомендую получать доступ к объекту для изменения реквизитов объекта! Изменение реквизитов успешно можно сделать посредством основного реквизита формы, и мы уже это делали ранее.

Чтобы показать, как работает метод формы *РеквизитФормыВЗначение,* выполним небольшую задачу: сделаем эмулятор печати документа «Прибытие в гараж» в модуле объекта этого документа: в экспортной процедуре Печать() просто будем выводить сообщение о том, что напечатали документ. А на форму добавим команду «Печать», при нажатии на которую будет срабатывать процедура печати из модуля документа.

В модуле объекта документа «Прибытие в гараж» напишем следующую процедуру:

```
Процедура Печать() Экспорт

      Сообщить("Документ прибытие в гараж №" +
                Номер + " от " + Формат(Дата,"ДЛФ=DD"));

КонецПроцедуры
```

Листинг 6.3.3

Создадим команду формы «Печать», которую разместим на командной панели формы (см. рис. 6.3.20).

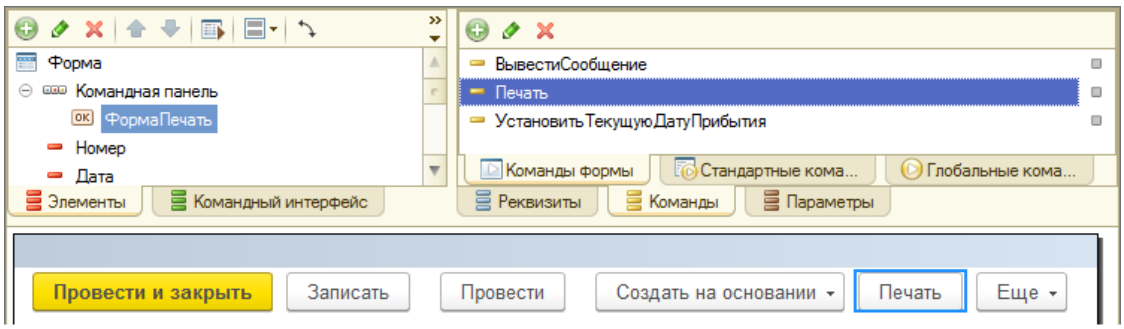


Рис. 6.3.20

Создадим для команды «Печать» обработчики на клиенте и на сервере. Почему в том числе *на севере*? Ответьте на этот вопрос самостоятельно, посмотрев описание метода *РеквизитФормыВЗначение*.

В обработчиках команды формы напишем следующий код:

```
&НаСервере
Процедура ПечатьНаСервере()

    ОбъектДок = РеквизитФормыВЗначение("Объект");
    ОбъектДок.Печать();

КонецПроцедуры
```

```
&НаКлиенте
Процедура Печать(Команда)
    ПечатьНаСервере();
КонецПроцедуры
```

Листинг 6.3.4

Сохраним конфигурацию, обновим базу данных и попробуем выполнить нашу команду (см. рис. 6.3.21).

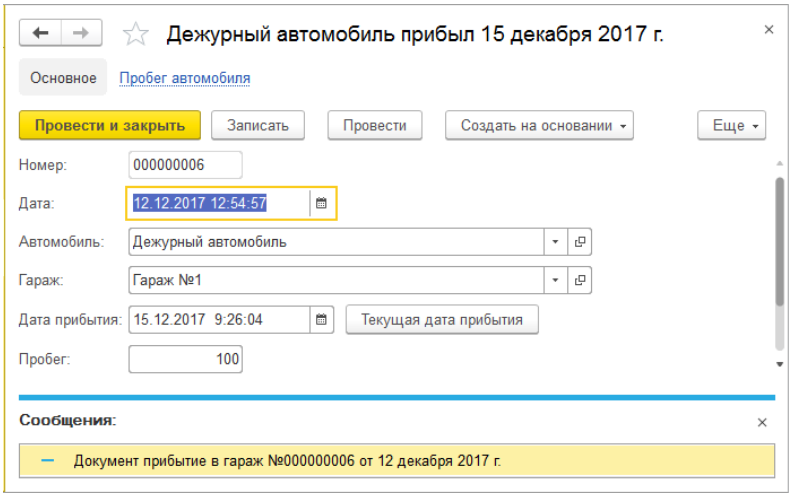


Рис. 6.3.21

Свойства и события элементов формы

В прошлой главе мы научились работать с элементами формы в конфигураторе (добавлять, изменять свойства и т.д.). В этой главе мы научимся программно работать с элементами формы. Элемент формы является, по сути, объектом и тоже имеет свои собственные свойства, методы и события.

Узнаем, каким способом можно обращаться к элементам формы. Для этой тренировки создадим обработку, форму обработки, на форме разместим пару реквизитов примитивных типов и пару элементов: поля ввода реквизитов, группы и декорации. Также на форме создадим команду, которую назовем «Доступ к элементам», и обработчик для этой команды на клиенте (см. рис. 6.3.22).

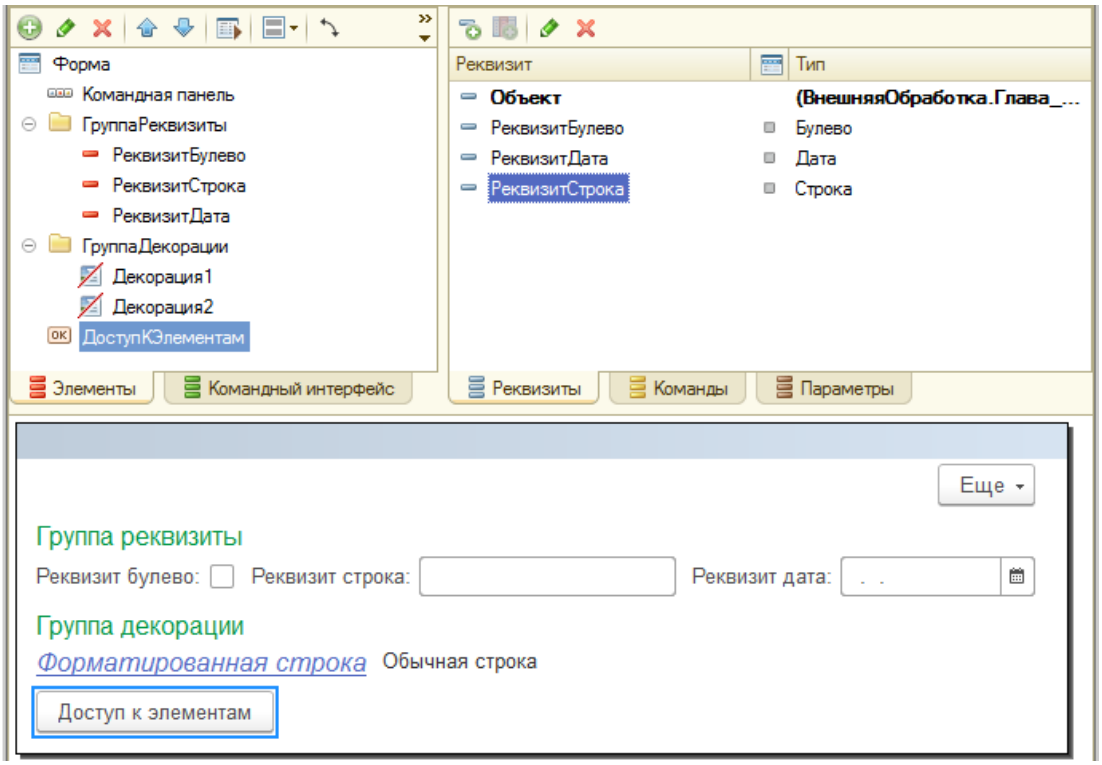


Рис. 6.3.22

Доступ ко всем элементам формы осуществляется посредством свойства формы *Элементы*, которое является коллекцией элементов формы. Поставим в конце процедуры обработчика команды «Доступ к элементам» точку останова и выполним эту команду (см. рис. 6.3.23) в пользовательском режиме.

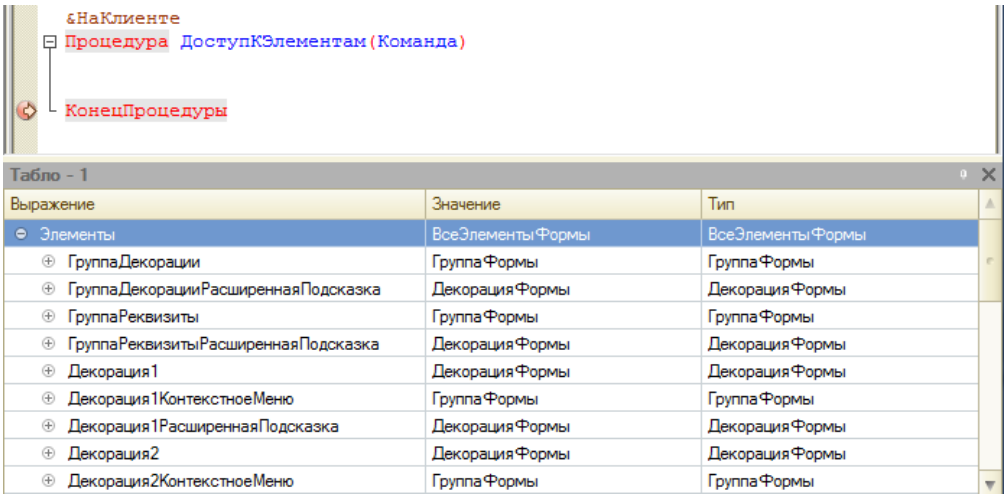


Рис. 6.3.23

Как видно из рисунка, через свойство формы *Элементы* мы можем получить доступ ко свойствам всех элементов, которые размещены на форме.

У каждого вида элемента много различных свойств, и объяснять все свойства не входит в рамки этой книги. Поэтому, как и со свойствами формы, научимся работать со справочной информаций.

Справочная информация об элементах формы находится в папке «Интерфейс (управляемый)». Нас интересуют четыре элемента: поле формы, кнопка формы, группа формы и декорация формы (см. рис. 6.3.24).

Рис. 6.3.24

Возьмем, к примеру, *Поле формы*. У этого элемента есть свойства, события и методы (см. рис. 6.3.25).

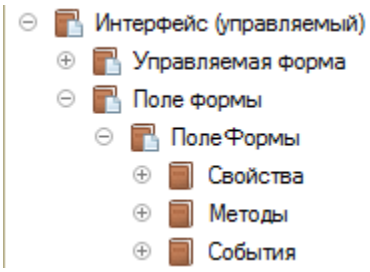


Рис. 6.3.25

Но поле формы не бывает само по себе, обычно оно имеет какой-нибудь вид, это может быть *поле ввода*, *поле флажка*, *поле переключателя* и т.д. Все эти виды поля ввода являются его расширениями и для них есть дополнительные свойства, методы и события. Их описание расположено в соответствующих каталогах (см. рис. 6.3.26).

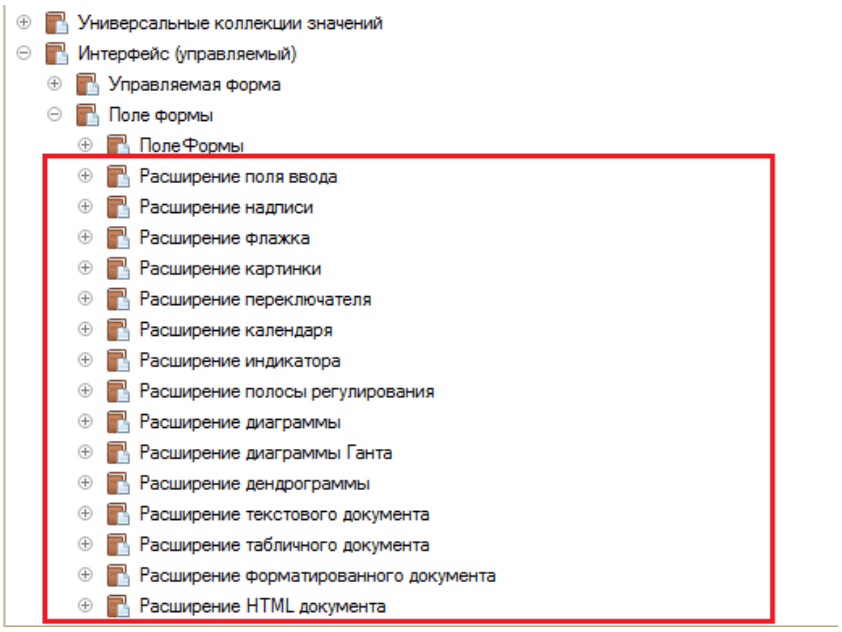


Рис. 6.3.26

Причем Вы видите, что расширений поля ввода гораздо больше, чем видов полей ввода, которые мы проходили в этой книге. Также расширения имеются для групп (см. рис. 6.3.27) и для декораций (см. рис. 6.3.28).

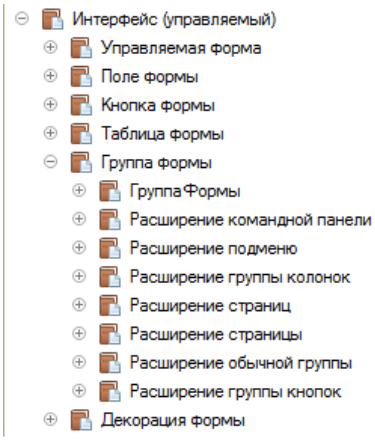


Рис. 6.3.27

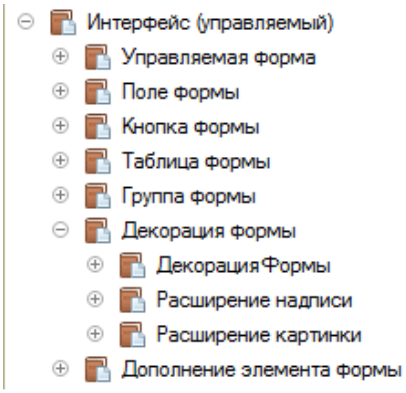


Рис. 6.3.28

Сделаем следующий пример: у нас на форме размещен реквизит Булево, который размещен в виде поля формы с видом *Поле флажка*. По умолчанию у этого поля стоит вид флажка *Авто* (см. рис. 6.3.29). Изменим программно вид флажка с «Авто» на «Тумблер», воспользовавшись уже созданным обработчиком команды *ДоступКЭлементам*.

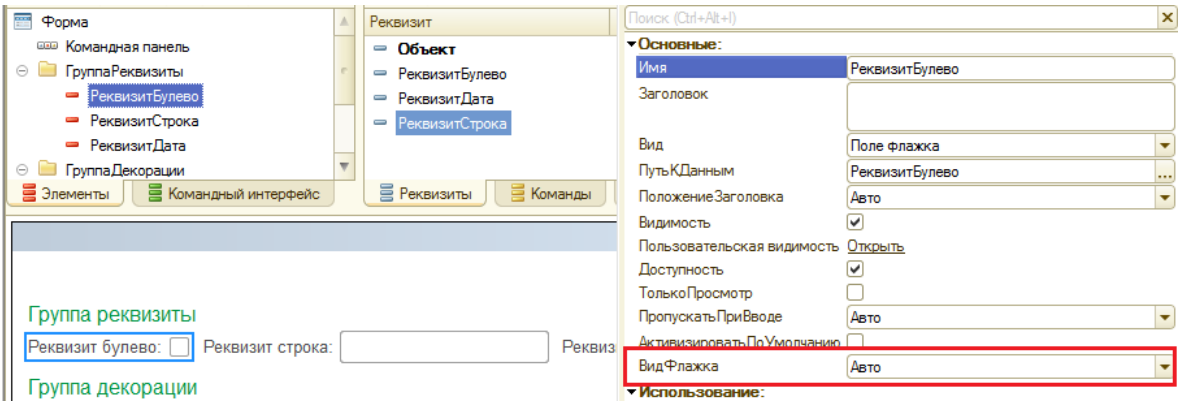


Рис. 6.3.29

Перед тем как программно изменять свойство элемента, найдем его в справке (сделайте это самостоятельно) и посмотрим его описание (см. рис. 6.3.30).

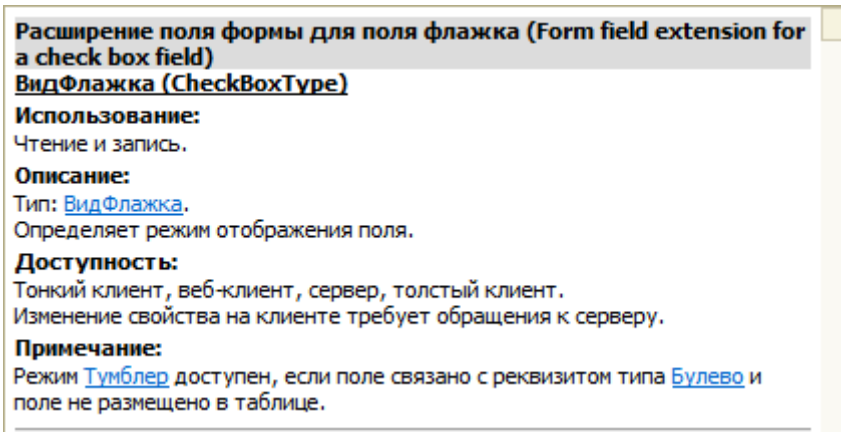


Рис. 6.3.30

Нас интересует возможность изменения этого свойства (раздел «Использование»), а также доступу к нему под тонким клиентом (раздел «Доступность»), иначе пришлось бы делать вызов сервера. И то, и то мы можем делать. А также узнаем, какой тип имеет данное свойство (раздел «Описание»). Это системное перечисление «ВидФлажка», посмотрим, можно ли обращаться к нему на тонком клиенте (см. рис.6.3.31).

ВидФлажка (CheckBoxType)

Значения

Авто (Auto)
Тумблер (Tumbler)
Флажок (CheckBox)

Описание:

Содержит варианты режимов отображения поля вида ПолеФлажка в управляемой форме.

Доступность:

Тонкий клиент, веб-клиент, мобильный клиент, сервер, толстый клиент, внешнее соединение.

Рис. 6.3.31

Напишем в обработчике команды *ДоступКЭлементам* код, в котором будем менять вид флажка (см. листинг 6.3.5).

```
&НаКлиенте
Процедура ДоступКЭлементам(Команда)

    Элементы.РеквизитБулево.ВидФлажка = ВидФлажка.Тумблер;

КонецПроцедуры
```

Листинг 6.3.5

Самостоятельно опробуйте работу этого кода.

Рассмотрим еще один пример: для полей ввода даты и строки (реквизиты «РеквизитСтрока» и «РеквизитДата») установим значение свойства *ТолькоПросмотр* в *Истина* (см. рис. 6.3.32).

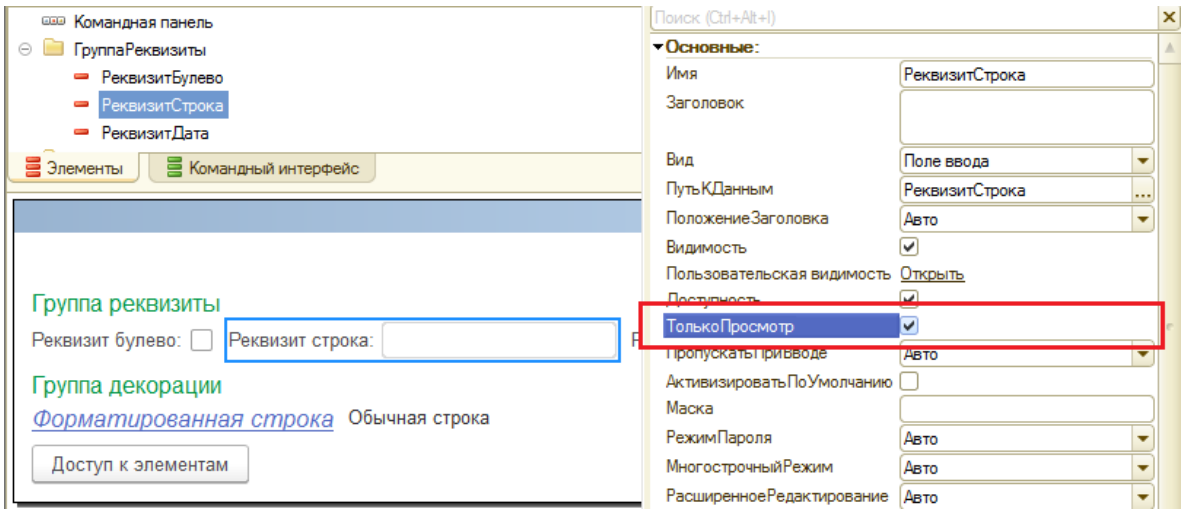


Рис. 6.3.32

После этих действий при открытии формы мы не сможем редактировать информацию в этих полях. Но, сделаем так, чтобы после нажатия на кнопку «Доступ к элементам» оба этих поля открывались для редактирования (см. листинг 6.3.6).

Самостоятельно найдите свойство «ТолькоПросмотр» поля ввода и посмотрите доступность этого свойства.

Напишем код в обработчике события команды *ДоступКЭлементам*.

```
&НаКлиенте
Процедура ДоступКЭлементам(Команда)

    Элементы.РеквизитДата.ТолькоПросмотр = Ложь;
    Элементы.РеквизитСтрока.ТолькоПросмотр = Ложь;

КонецПроцедуры
```

Листинг 6.3.6

Небольшое задание к этой части: уберите видимость групп на форме, поместите на форму кнопку и при нажатии на эту кнопку показывайте скрытые группы.

Со свойствами элементов разобрались, как Вы поняли, в этом нет ничего сложного: просто обращаетесь к нужному элементу формы и через точку получаете свойство, которое или просто читаете, или присваиваете ему какое-нибудь значение того типа, который определен для этого свойства (если не знаете, смотрите в справочной информации).

Научимся работать с событиями элементов формы. В этой книге мы научимся создавать события элементов только посредством конфигуратор, программное создание событий дается в книге [«Основы разработки в 1С: Такси»](#).

В конфигураторе перечень событий элемента находится в палитре свойств (см. рис. 6.3.33).

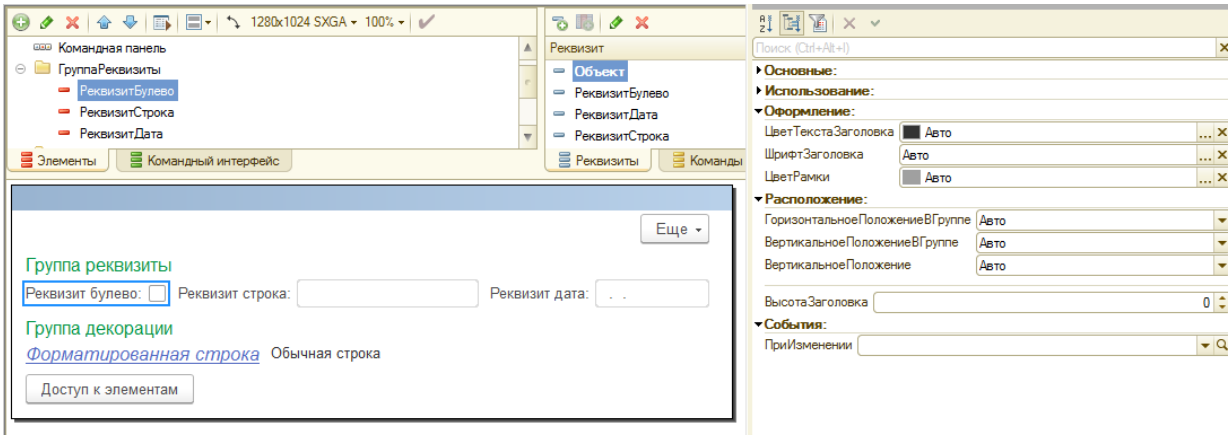


Рис. 6.3.33

Так же, как и со свойствами, наличие или отсутствие того или иного события определяется расширением этого элемента. Например, на рис. 6.3.33 Вы видите события для элемента формы *Поле формы* с видом *Поле флажка*, а для аналогичного элемента формы только с видом *Поле ввода* будет другой перечень событий (см. рис. 6.3.34).

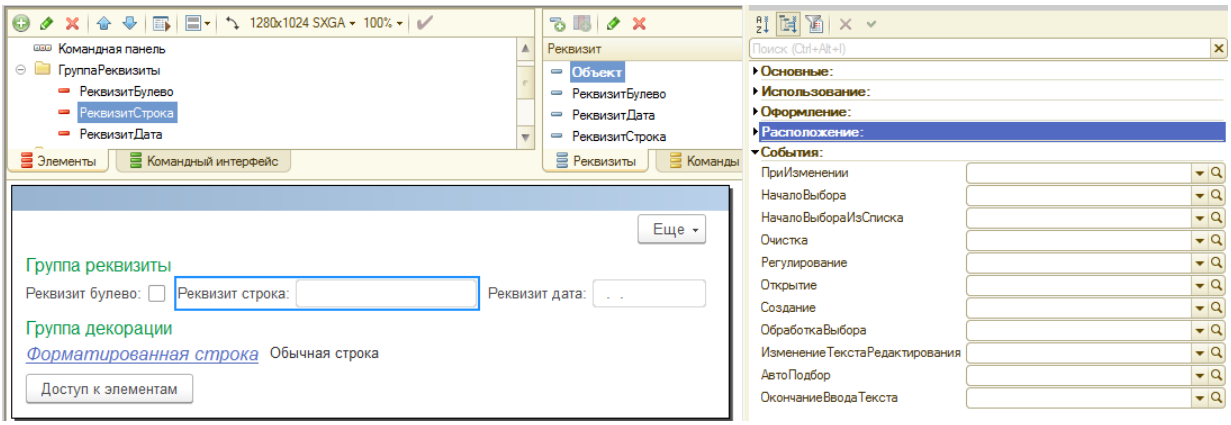


Рис. 6.3.34

Прочитать информацию о том, у каких полей формы какие события имеются, можно во все той же справочной информации. Например, просто у *поля формы* есть только одно событие *ПриИзменении* (см. рис. 6.3.35). Это значит, что это событие есть у всех *полей формы* в независимости от их расширения.

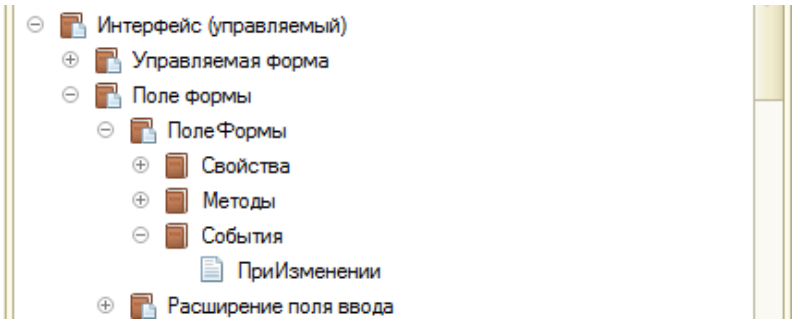


Рис. 6.3.35

Все события *поля формы* с типом *поле ввода* описаны в расширении для поля ввода (см. рис. 6.3.36).

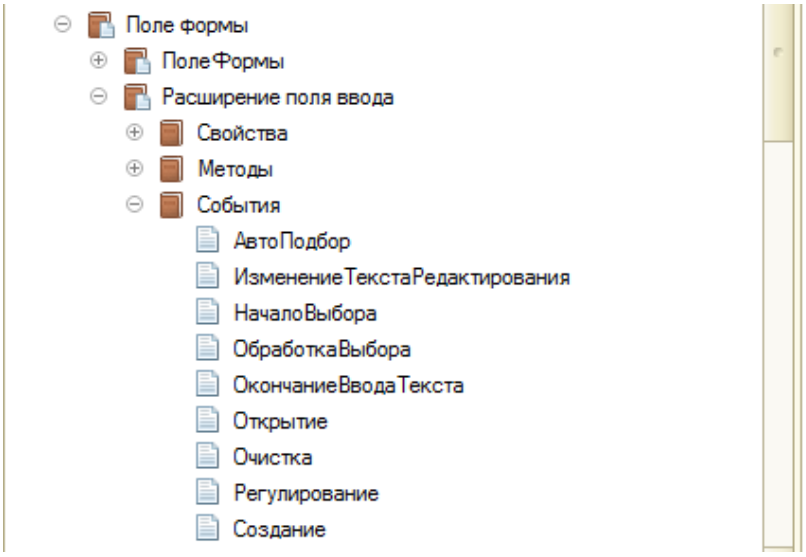


Рис. 6.3.36

Выполним несколько примеров: если у нас флажок (сделайте сами из него тумблер) нажат, то будем показывать поля реквизитов, а если флажок снят, то – нет.

Создадим обработчик события *ПриИзменении* поля формы, для этого зайдем в палитру свойств этого элемента и нажмем на кнопку «Лупа» соответствующего события (см. рис. 6.3.37).

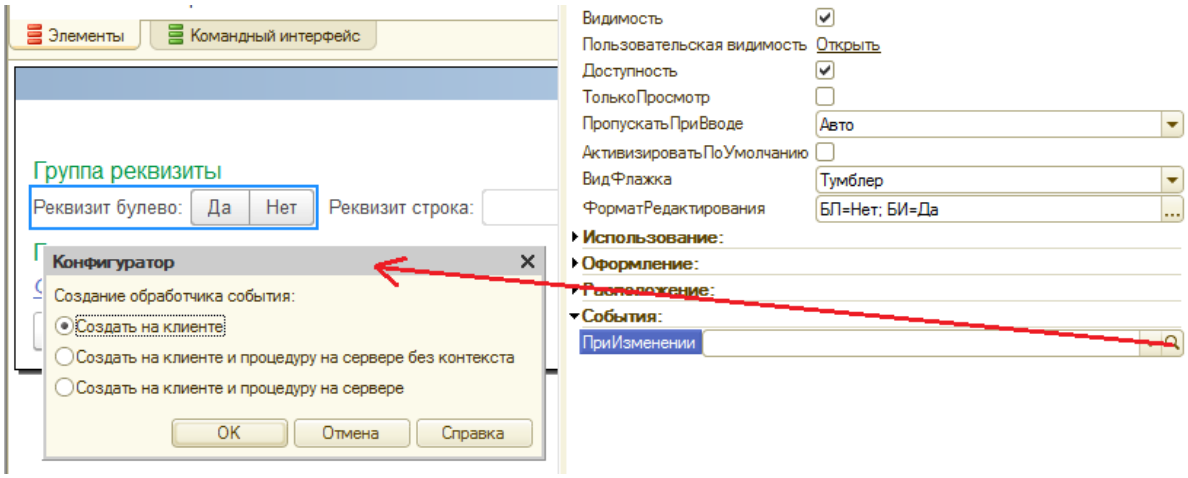


Рис. 6.3.37

За видимость элемента на форме отвечает свойство этого элемента - *Видимость.* Самостоятельно найдите в синтаксис-помощнике информацию о доступности этого свойства. А я забегу вперед и скажу, что оно доступно на тонком клиенте. Поэтому обработчик события создадим только на клиенте. В этом обработчике напишем код, где свойству *Видимость* соответствующих элементов будем присваивать значение реквизита *РеквизитБулево*, который связан с нашим полем флажка: значение этого реквизита изменяется при изменении поля флажка.

```
&НаКлиенте
Процедура РеквизитБулевоПриИзменении(Элемент)

    Элементы.РеквизитДата.Видимость = РеквизитБулево;
    Элементы.РеквизитСтрока.Видимость = РеквизитБулево;

КонецПроцедуры
```

Листинг 6.3.7

Посмотрим на работу этого события (см. рис. 6.3.38 и 6.3.39).

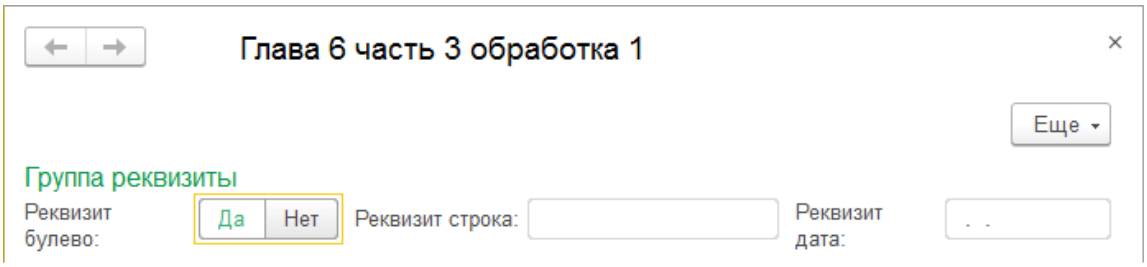


Рис. 6.3.38

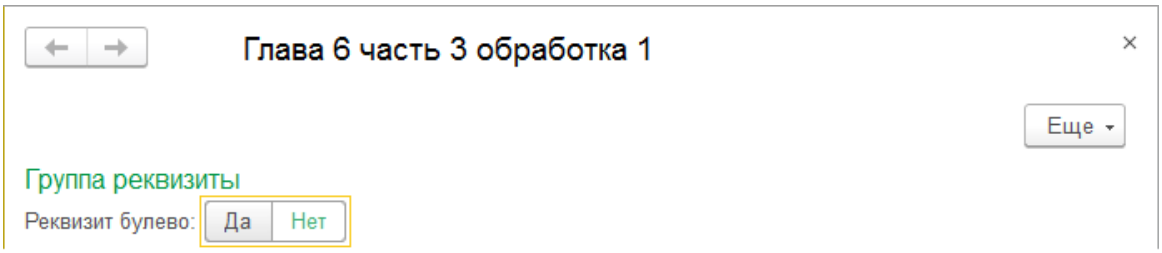


Рис. 6.3.39

Доработайте код: скрывайте точно так же декорации и при открытии формы меняйте заголовок поля флажка на «Скрыть реквизиты».

Рассмотрим еще один пример: для поля ввода «РеквизитСтрока» установим кнопку очистки, для этого необходимо установить флаг у свойства «КнопкаОчистки». Я это сделаю в конфигураторе (см. рис. 6.3.40), а Вы сделайте программно при открытии формы. При нажатии на кнопку очистки происходит очистка поля ввода. Причем у поля ввода есть такое событие «Очистка», которое срабатывает, когда нажимают на эту кнопку. Перехватим это событие и при очистке будем проверять, заполнено ли поле ввода «РеквизитДата», и если заполнено, то не будем очищать это поле и выведем сообщение, что нужно сначала очистить поле ввода *РеквизитДата*.

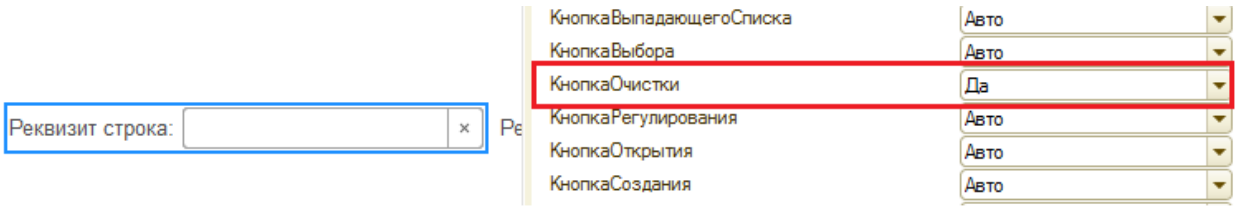


Рис. 6.3.40

Найдем событие «Очистка» у поля у элемента формы *РеквизитСтрока* и создадим обработчик этого события на клиенте (см. рис. 6.3.41).

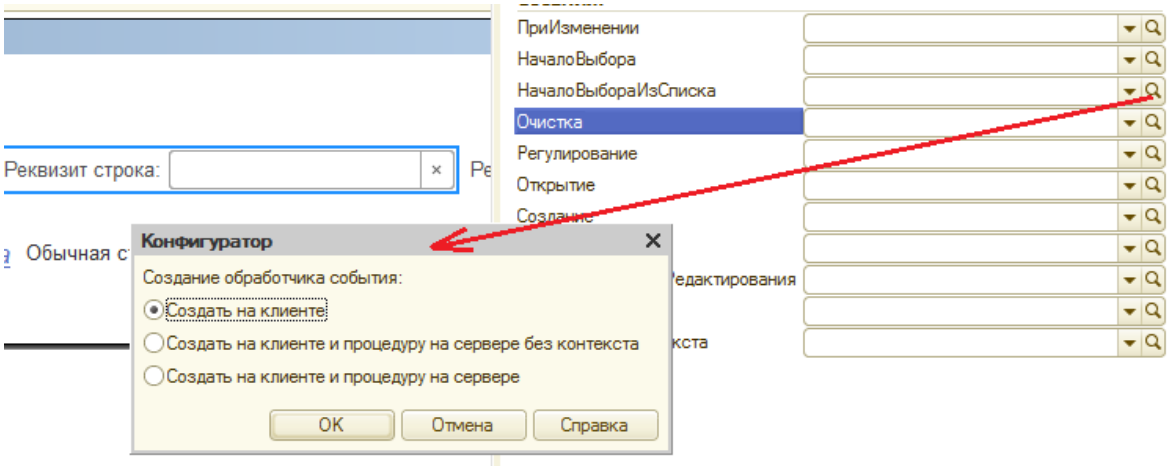


Рис. 6.3.41

Заметьте, у процедуры обработчика появился параметр *СтандартнаяОбработка* (см. рис. 6.3.42). Этот параметр имеет тип Булево и по умолчанию принимает значение *Истина*. Когда

данный параметр имеет значение *Истина*, произойдет стандартная обработка события, в нашем случае поле ввода будет очищено. Если в обработчике события этот параметр примет значение *Ложь*, то стандартной обработки события не произойдет, т.е. поле ввода не очистится.

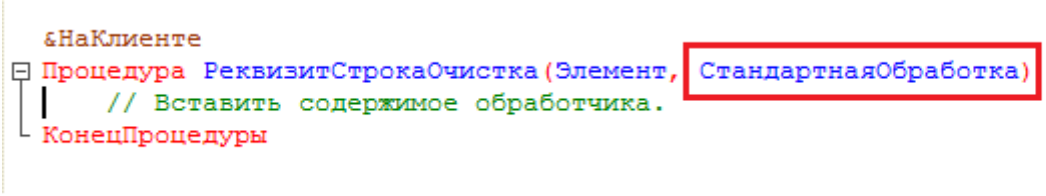


Рис. 6.3.42

Напишем в процедуре код по алгоритму, описанному выше (см. листинг 6.3.8).

```
&НаКлиенте
Процедура РеквизитСтрокаОчистка(Элемент, СтандартнаяОбработка)

    СтандартнаяОбработка = Ложь;

    Если ЗначениеЗаполнено(РеквизитДата) Тогда
        Сообщить("Очистите сначала дату");
    Иначе
        СтандартнаяОбработка = Истина;
    КонецЕсли;

КонецПроцедуры
```

Листинг 6.3.8

Проверьте работу этого кода самостоятельно.

Открытие формы

И напоследок, изучим основы открытия форм. Разберем, что происходит в системе во время создания и открытия формы.

Нас интересуют два варианта создания формы. Первый – открытие формы существующего объекта. Второй – открытие формы для нового объекта. В обоих вариантах не имеет значения, как открывается форма – интерактивно пользователем или программно.

Открытие формы существующего объекта

Первый шаг. После того, как пользователь инициировал открытие формы существующего объекта (пусть, в нашем случае это форма документа *Прибытие в гараж*), на сервере будет создан *экземпляр объекта* документа (тип *ДокументОбъект.<НазваниеДокумента>*) и произойдет выполнение кода *модуля объекта* (который идет после всех процедур модуля). Впоследствии, происходит считывание данных из базы и заполнение реквизитов экземпляра

объекта документа. Данный процесс осуществляется платформой автоматически и вмешаться в него мы не можем.

Во время второго шага *на сервере* создается и заполняется экземпляр управляемой формы, которая указана в свойстве объекта *Основная форма объекта* (если основной формы нет, то макет формы генерируется автоматически). После того, когда прошло заполнение управляемой формы данными из экземпляра объекта документа вызывается событие формы *ПриЧтенииНаСервере*, которое имеет один параметр *ТекущийОбъект*, содержащий созданный экземпляр объекта.

Заметьте, когда произошел вызов этого события, *основной* реквизит формы *Объект* уже заполнен данными. Но у нас есть еще возможность дозаполнить или перезаполнить основной реквизит формы по данным экземпляра объекта.

Только в этом событии есть доступ к экземпляру объекта документа!

После заполнения основного объекта формы экземпляр объекта документа на сервере будет уничтожен.

Во время третьего шага идет уже работа с заполненной формой, которая носит подготовительный характер. Сначала происходит событие *ПриСозданииНаСервере*, во время выполнения этого события мы можем подготовить форму, когда она еще находится на сервере. А также отменить создание формы.

И в последнем четвертом шаге все данные формы, которые были на сервере, сериализуются и отправляются на клиента. При этом в клиентском контексте срабатывает событие *ПриОткрытии*, в нем нужно выполнять различные манипуляции с формой, которые невозможно сделать на сервере.

Разберем все шаги создания формы на примере. Экспериментировать мы в этот раз будем с формой документа «Установка цен на топливо».

Напишем в модуле объекта документа «Установка цен на топливо» простой код после всех процедур:

```
Сообщить("Шаг 1. В модуле объекта");
```

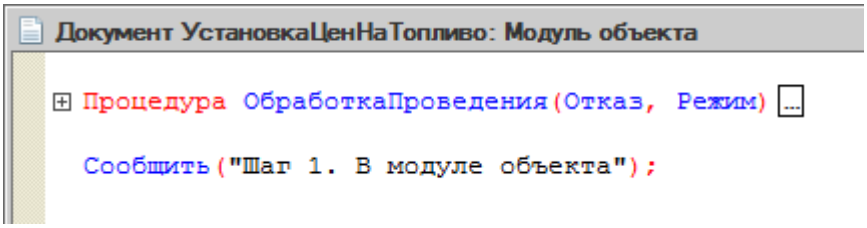


Рис. 6.3.43

Теперь создадим обработчики следующих событий формы; *ПриЧтенииНаСервере*, *ПриСозданииНаСервере* и *ПриОткрытии*. Для этого зайдем в палитру свойств формы и нажмем на кнопку «Лупа» соответствующих обработчиков (см. рис. 6.3.44).

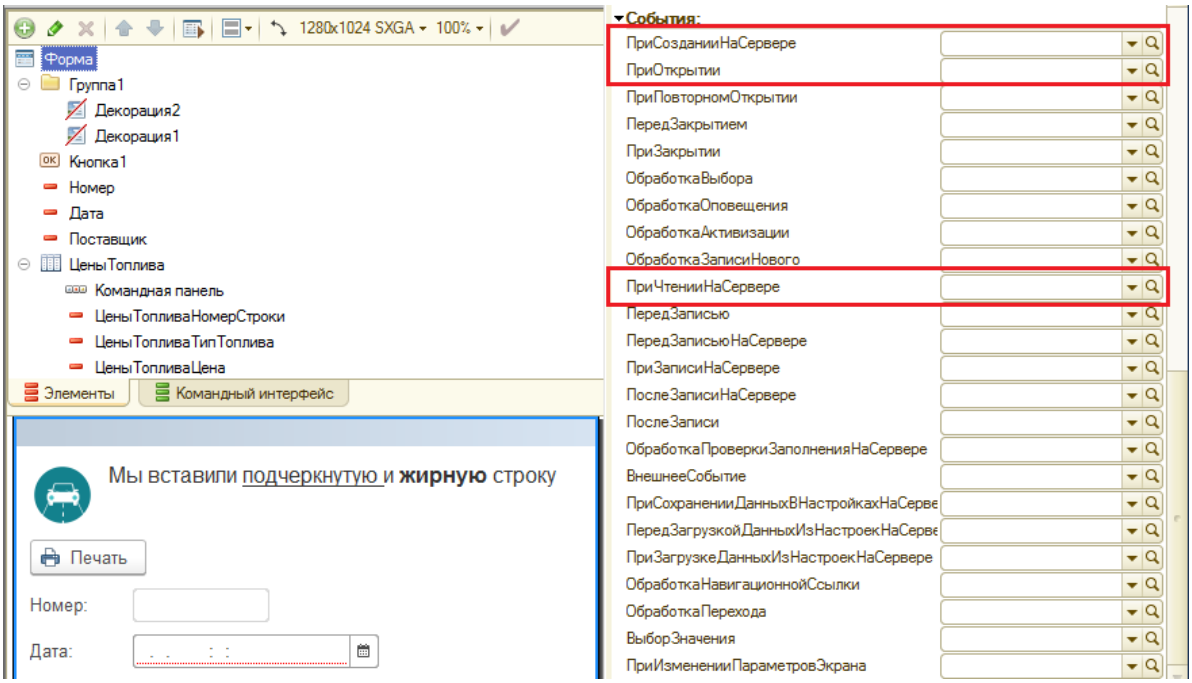


Рис. 6.3.44

В процедурах обработчиках событий напишем следующий код (см. листинг 6.3.9):

```
&НаСервере
Процедура ПриЧтенииНаСервере(ТекущийОбъект)
    Сообщить("Шаг 2. Заполнили данные формы по данным объекта");
КонецПроцедуры

&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)
    Сообщить("Шаг 3. Обработка создания формы на сервере");
КонецПроцедуры

&НаКлиенте
Процедура ПриОткрытии(Отказ)
    Сообщить("Шаг 4. Обработка открытия формы на клиенте");
КонецПроцедуры
```

Листинг 6.3.9

Сохраним конфигурацию, обновим базу данных и откроем форму документа «Установка цен на топливо» (см. рис. 6.3.45).

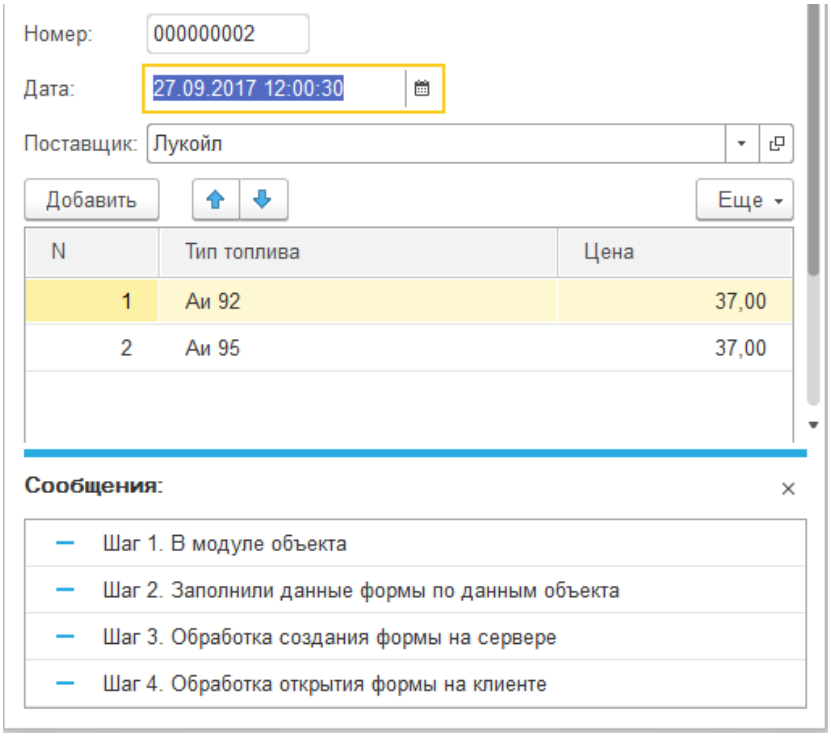


Рис. 6.3.45

Открытие формы нового объекта

При открытии формы нового объекта будут несколько иные шаги создания формы.

Первый шаг. После того, как пользователь инициировал открытие формы нового документа, или запустил команду создания документа на основании, будет создан *новый экземпляр* соответствующего объекта. После этого произойдет выполнение кода в модуле объекта (который после всех процедур), и отработает событие в модуле объекта *ОбработкаЗаполнения*. Если прикладная задача предусматривает создание нового объекта на основании какого-то существующего, то в событии *ОбработкаЗаполнения* необходимо реализовать заполнение реквизитов и табличных частей нового объекта по имеющимся данным.

Во время второго шага, аналогично, как и в случае открытия формы существующего объекта, будет создана форма объекта на сервере и произойдет заполнение основного реквизита формы по тем данным, которые были заполнены в событии модуля объекта *ОбработкаЗаполнения*. Но при открытии формы нового объекта на сервере не будет отрабатываться событие *ПриЧтенииНаСервере*. Потому что в этом случае читать особо и нечего: не создан экземпляр какого-либо объекта.

Шаги третий и четвертый, где используются обработки событий *ПриСозданииНаСервере* и *ПриОткрытии,* работают точно так же, как и тогда, когда идет открытие формы существующего объекта.

Теперь проверим на практике вышесказанную цепочку, для этого создадим в модуле объекта документа «Установка цен» событие *ОбработкаЗаполнения*, где будем выводить сообщение о первом шаге.

Если Вы забыли, то напомню, как создать событие в модуле объекта. Для этого необходимо открыть сам модуль, активизировать его (т.е. установить курсор) и выполнить команду «Процедуры и функции» (см. рис. 6.3.46).

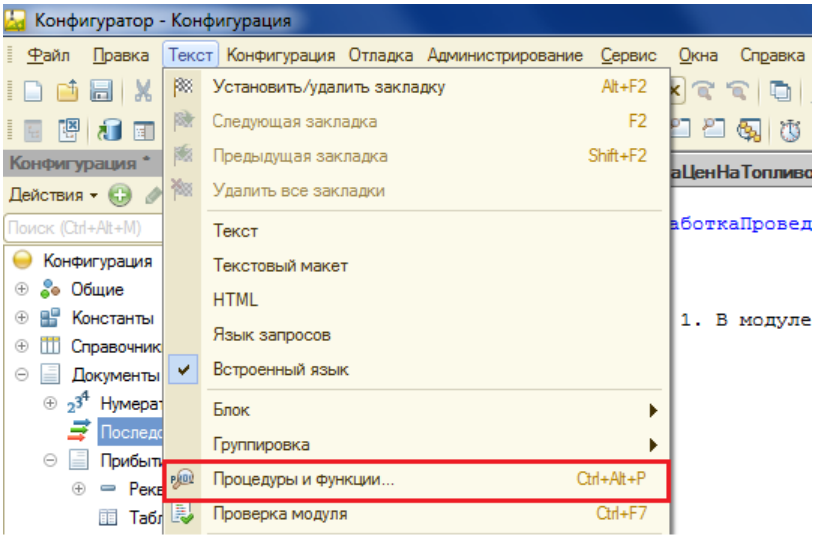


Рис. 6.3.46

После выйдет окно «Процедуры и функции», где выделим пункт «Обработка заполнения» и нажмем кнопку «Перейти» (см. рис. 6.3.47). После этого будет создана пустая процедура, в которой напишем код, как в листинге 6.3.9.

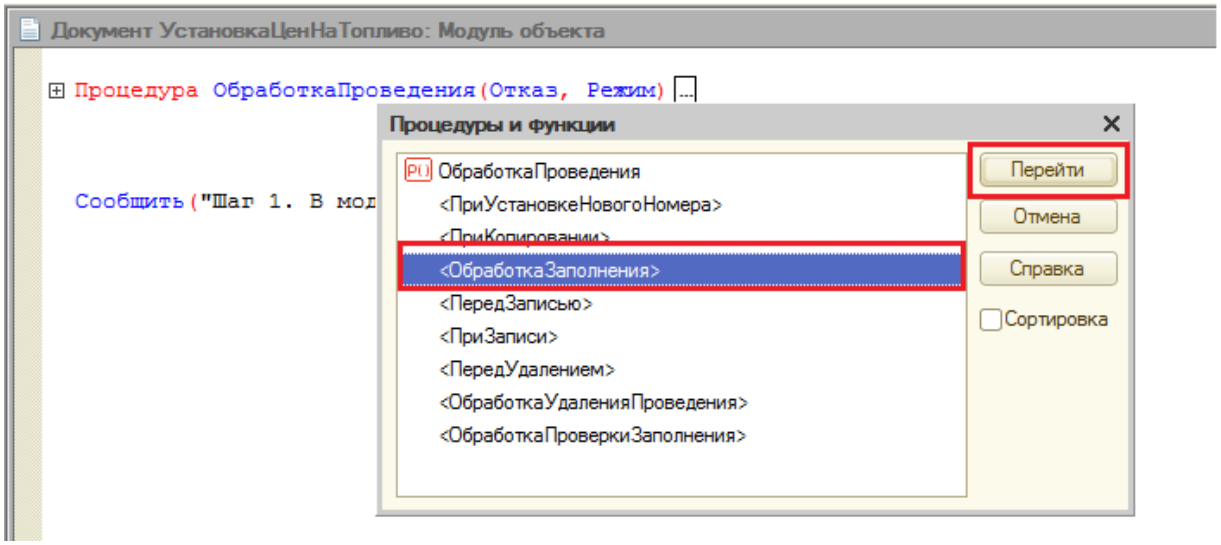


Рис. 6.3.47

```
Процедура ОбработкаЗаполнения(ДанныеЗаполнения, ТекстЗаполнения,
СтандартнаяОбработка)

    Сообщить("Шаг 1.1. Обработка заполнения");

КонецПроцедуры
```

Листинг 6.3.10

Сохраним конфигурацию, обновим базу данных и интерактивно создадим новый документ «Установка цен на топливо». При этом произойдет открытие формы документа (см. рис. 6.3.48).

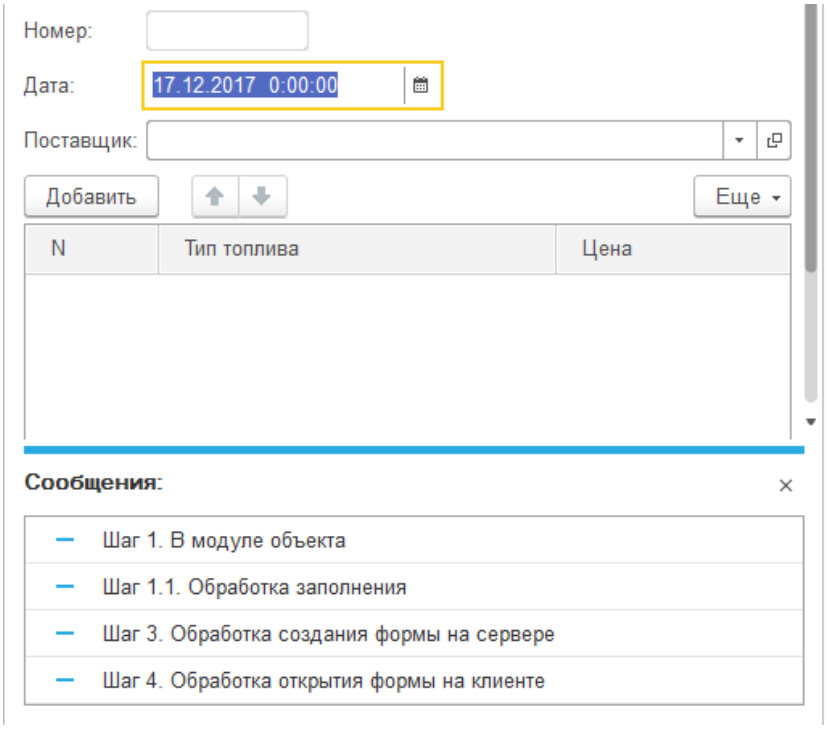


Рис. 6.3.48

Заметьте, на рисунке 6.3.48 в окне сообщений нет информации о шаге 2, потому что не произошла обработка события *ПриЧтенииНаСервере*. В то же время если мы заполним какие-либо реквизиты объекта в обработчике модуля объекта *ОбработкаЗаполнения*, то они отобразятся на форме.

Программное открытие формы

В предыдущих подразделах мы узнали, что происходит при открытии формы и какие обработчики задействованы при открытии формы существующего объекта и открытии формы нового объекта. В этом подразделе мы научимся открывать формы объектов программным способом и узнаем, как использовать параметры форм.

Понятно, что открыть форму мы можем только в клиентском контексте, именно на том компьютере, где работает пользователь. *На сервере* нет никакого смысла в открытии формы. Поскольку форма это основной интерфейсный элемент взаимодействия пользователя с программой. Как следствие запускать методы открытия форм необходимо в процедурах или функциях, которые расположены под директивой *&НаКлиенте*. Форму можно открывать под *Толстым клиентом*, *Тонким клиентом* и *Веб-клиентом*.

Научимся программно открывать формы. Для этого будем использовать новую обработку. Создадим для этой обработки форму, создадим команду «Открытие формы» и разместим её на форме обработки (см рис. 6.3.49). А также *на клиенте* создадим действие для этой команды.

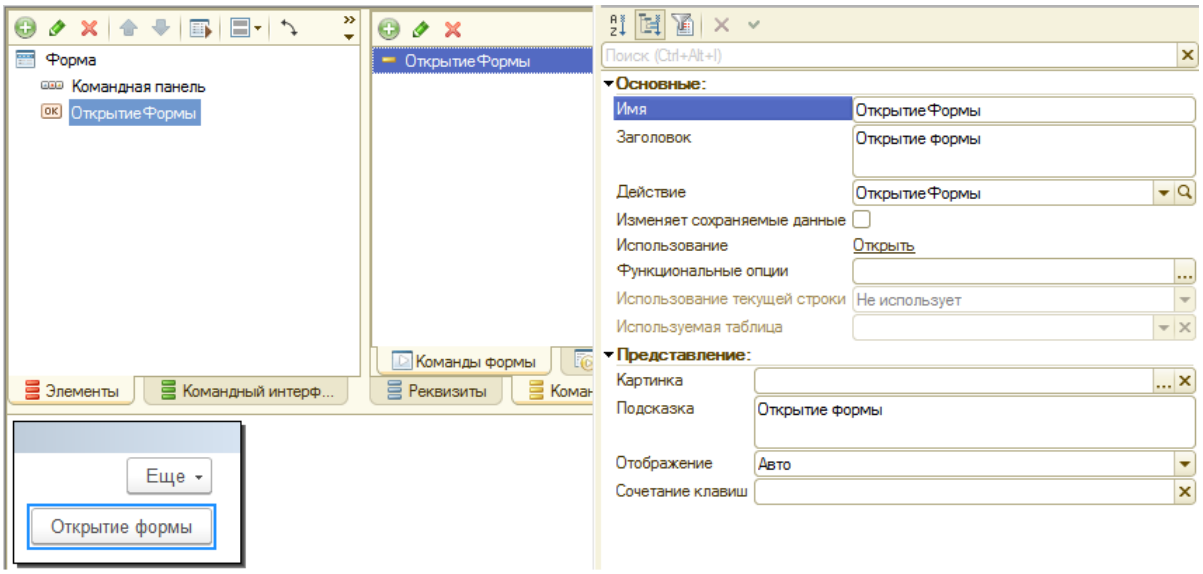


Рис. 6.3.49

В дальнейшем все действия подраздела мы будем осуществлять в процедуре *на клиенте* «ОткрытиеФормы» (см. листинг 6.3.11).

```
&НаКлиенте
Процедура ОткрытиеФормы(Команда)
   // Вставить содержимое обработчика.
КонецПроцедуры
```

Листинг 6.3.11

В управляемом приложении открыть форму можно при помощи трех методов глобального контекста (функций) - это *ПолучитьФорму*, *ОткрытьФорму* и *ОткрытьФормуМодально*. Я рекомендую Вам воздержаться от использования метода *ПолучитьФорму*, поскольку он работает гораздо медленнее методов *ОткрытьФорму* и *ОткрытьФормуМодально*.

На начальном этапе изучения языка программирования 1С Вам достаточно изучить метод «ОткрытьФорму», который имеет следующий синтаксис:

ОткрытьФорму(<ИмяФормы>, <Параметры>, <Владелец>, <Уникальность>, <Окно>, <НавигационнаяСсылка>, <ОписаниеОповещенияОЗакрытии>, <РежимОткрытияОкна>)

Где:
ИмяФормы – непосредственно имя формы (тип строка). В строке должен быть указан полный путь к нужной форме.

Это единственный обязательный параметр!

Например, для формы документа *Прибытие в гараж* имя формы может быть такой: "Документ.ПрибытиеВГараж.ФормаОбъекта" или такой: "Документ.ПрибытиеВГараж.Форма.ФормаДокумента". Ниже я научу Вас, как быстро и без ошибок получать имя формы.

В этой книге мы изучим только такие параметры как: «ИмяФормы», «Параметры и РежимОткрытияОкна». Работа с остальными параметрами дается в 3 части книги «Основы разработки в 1С: Такси».

А теперь научимся безошибочно получать имя формы. Для этого будем использовать контекстную подсказку. Чтобы она включилась, необходимо написать название метода и открыть скобку и кавычку:

```
ОткрытьФорму("
```

После этого выйдет контекстная подсказка с группами объектов метаданных (см. рис. 6.3.49). Если она по какой-то причине не вышла, нажмите Ctrl + Space (пробел).

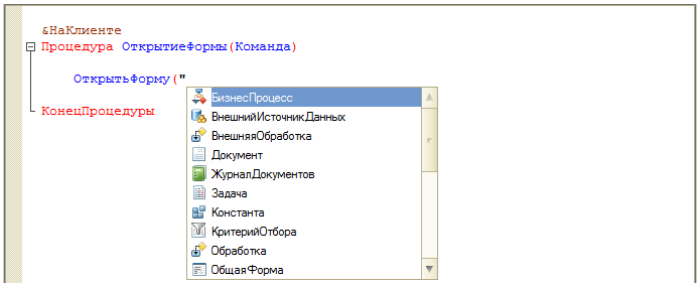


Рис. 6.3.50

Мы выберем группу метаданных «Документ», и после этого выйдет список всех имеющихся в нашей конфигурации документом (см. рис. 6.3.50).

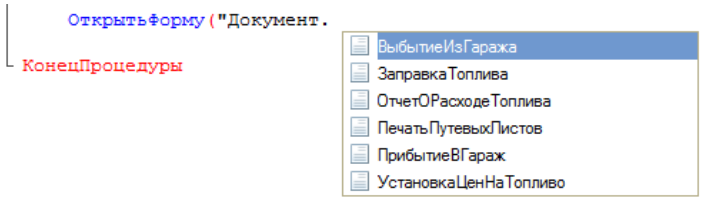


Рис. 6.3.51

Выберем документ *ПрибытиеВГараж*, после этого выйдет контекстная подсказка, в которой или можно будет выбрать какую-либо конкретную форму, или форму из свойства объекта (см. рис. 6.3.51).

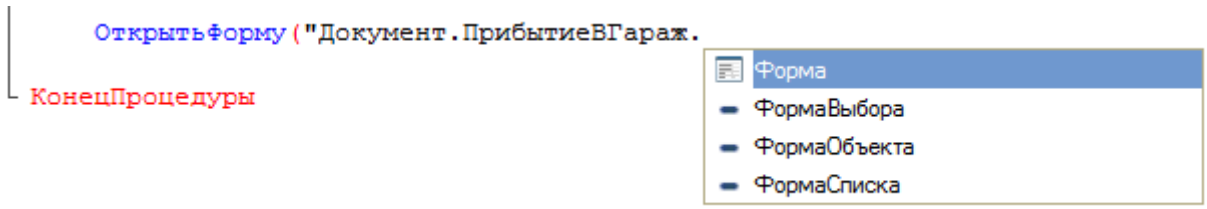


Рис. 6.3.52

Если из выпадающего списка форм Вы выберете пункты: «ФормаВыбора», «ФормаОбъекта» и «ФормаСписка», то будут открыты формы из соответствующих свойств объекта

конфигурации. В нашем случае это свойства «Основная форма объекта», «Основная форма выбора» и «Основная форма списка» (см. рис. 6.3.53). Если в каком-то из этих свойств не указана форма, то форма все равно будет открыта. Она сгенерируется платформой автоматически.

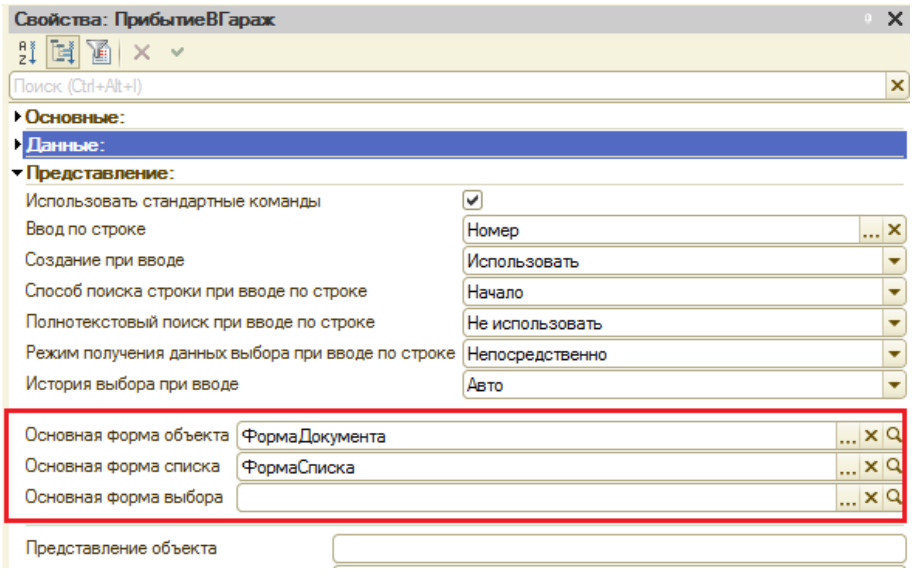


Рис. 6.3.53

Если в контекстной подсказке, изображенной на рис. 6.3.54, выбрать пункт «Формы», то выйдет следующая контекстная подсказка со всеми созданными формами объекта метаданных (см. рис. 6.3.54).

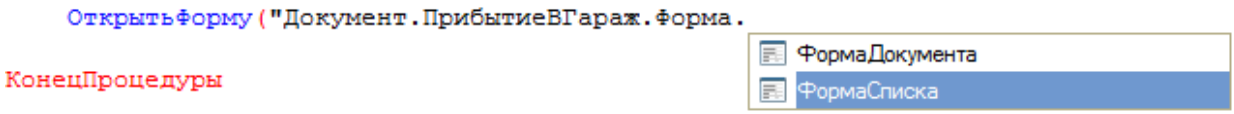


Рис. 6.3.54

В этой контекстной подсказке будут все формы, которые есть у объекта метаданных (см. рис. 6.3.55).

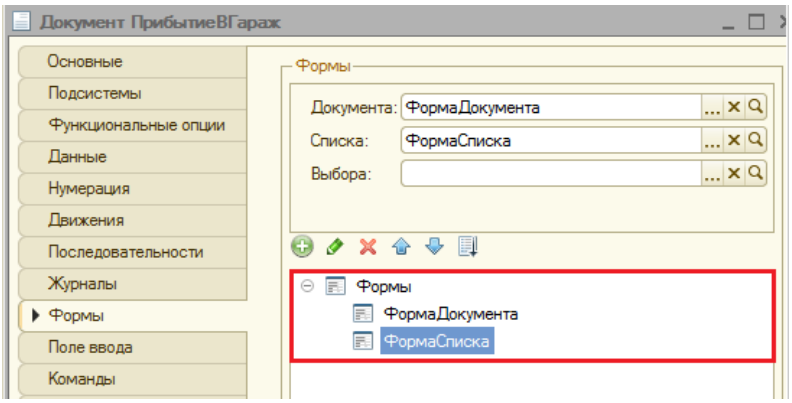


Рис. 6.3.55

Если у Вас нет формы списка у этого документа, то создайте её самостоятельно!

Выберем форму списка. Должен получиться следующий код:

```
ОткрытьФорму("Документ.ПрибытиеВГараж.Форма.ФормаСписка");
```

Если мы сейчас сохраним обработку, откроем её в нашей конфигурации и применим команду «Открытие формы», то откроется форма списка документа «Прибытие в гараж», причем в «соседнем» окне (см. рис. 6.3.56).

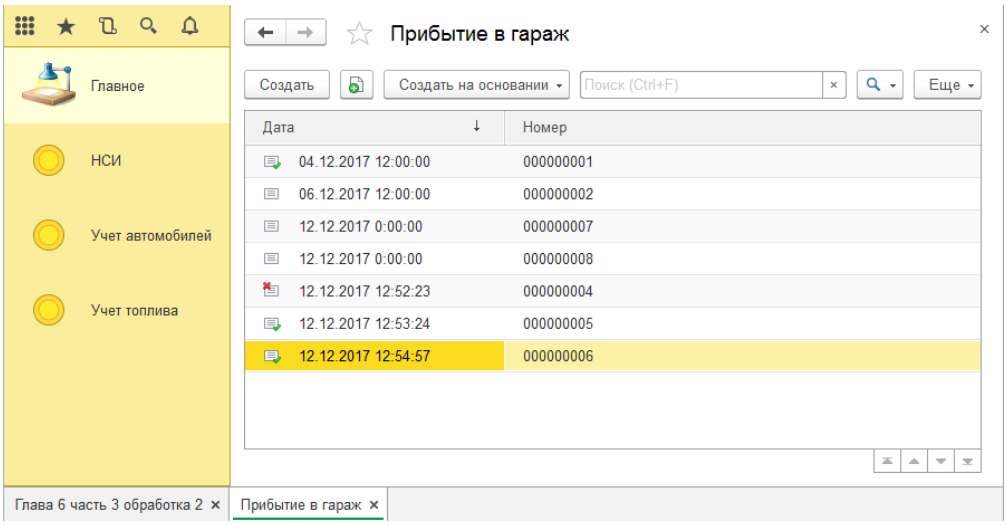


Рис. 6.3.56

А если выберем форму документа:

```
ОткрытьФорму("Документ.ПрибытиеВГараж.Форма.ФормаДокумента");
```

- то откроется пустая форма документа (см. рис. 6.3.57).

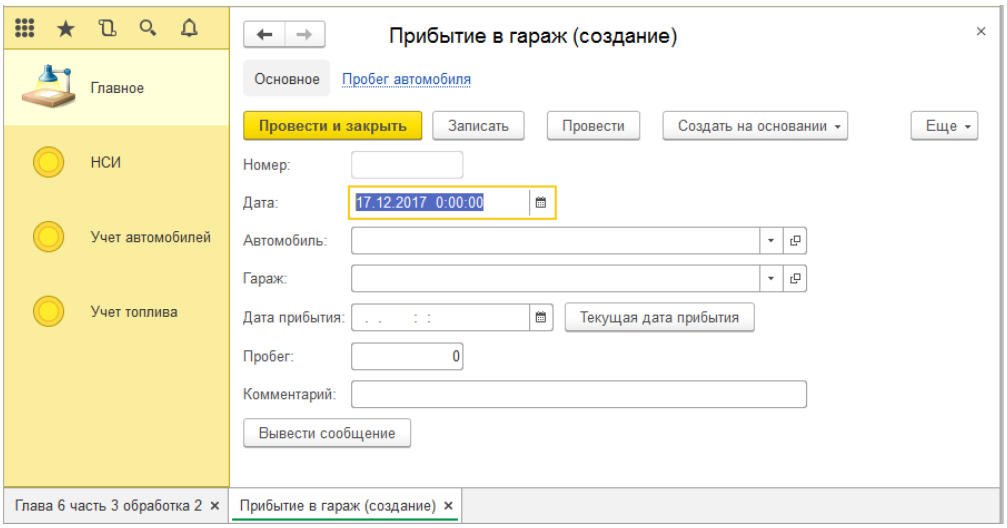


Рис. 6.3.57

Открытие форм с параметрами

Естественно у Вас возникнет вопрос: а каким образом открыть форму с уже существующим документом? Для того чтобы ответить на этот вопрос, нам придется изучить следующий параметр метода *ОткрытьФорму* - **<Параметр>**.

ОткрытьФорму(<ИмяФормы>, <Параметры>, <Владелец>, <Уникальность>, <Окно>, <НавигационнаяСсылка>, <ОписаниеОповещенияОЗакрытии>, <РежимОткрытияОкна>)

Этот параметр должен иметь тип *Структура*, где в качестве ключа элемента структуры должно быть задано *название* параметра, а в качестве значения элемента структуры – *значение* параметра. Он является необязательным и может быть пропущен, если нет необходимости в передаче каких-либо параметров в форму.

Подробно коллекцию «Структура» будем проходить в седьмой главе.

Чтобы открыть форму существующего документа или справочника, необходимо использовать параметр *Ключ*, который есть в расширении справочников и документов, в этот параметр необходимо передать ссылку на открываемый справочник или документ.

Пока звучит немного непонятно, но сейчас Вы поймете, как работает данный параметр. Для этого в уже созданной обработке создадим реквизит «Прибытие в гараж» с типом «ДокументСсылка.ПрибытиеВГараж» и разместим его на форме (см. рис. 6.3.58).

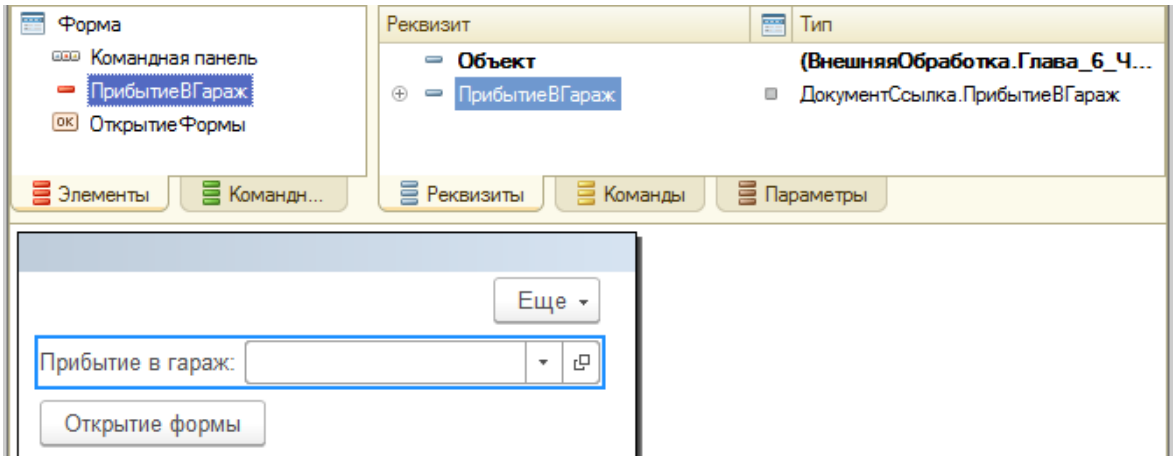


Рис. 6.3.58

Теперь нам осталось изменить обработчик команды «Открытие формы»: создадим новую структуру, с единственным элементом, ключ которого будет называться «Ключ», а в значение подставляться реквизит формы *ПрибытиеВГараж*. И эту структуру будем использовать в качестве второго параметра метода *ОткрытьФорму* (см. листинг 6.3.12).

```
&НаКлиенте
Процедура ОткрытиеФормы(Команда)
    ПараметрыФормы = Новый Структура;
    ПараметрыФормы.Вставить("Ключ",ПрибытиеВГараж);
    ОткрытьФорму("Документ.ПрибытиеВГараж.Форма.ФормаДокумента",
ПараметрыФормы);
КонецПроцедуры
```

Листинг 6.3.12

Если сейчас мы запустим обработку, выберем в поле «Прибытие в гараж» какой-либо документ и выполним команду «Открыть форму», то откроется форма того документа, который выбран нами (см. рис. 6.3.59).

Параметр *Ключ* является расширением управляемой формы для документа. Если при открытии формы объекта (документа, справочника, плана видов характеристик и т.п.) будет передаваться параметр *Ключ* со ссылкой на соответствующий объект, то будут выполнены все шаги, которые мы описали в подразделе «Открытие формы существующего объекта» этой части (см. стр. 416).

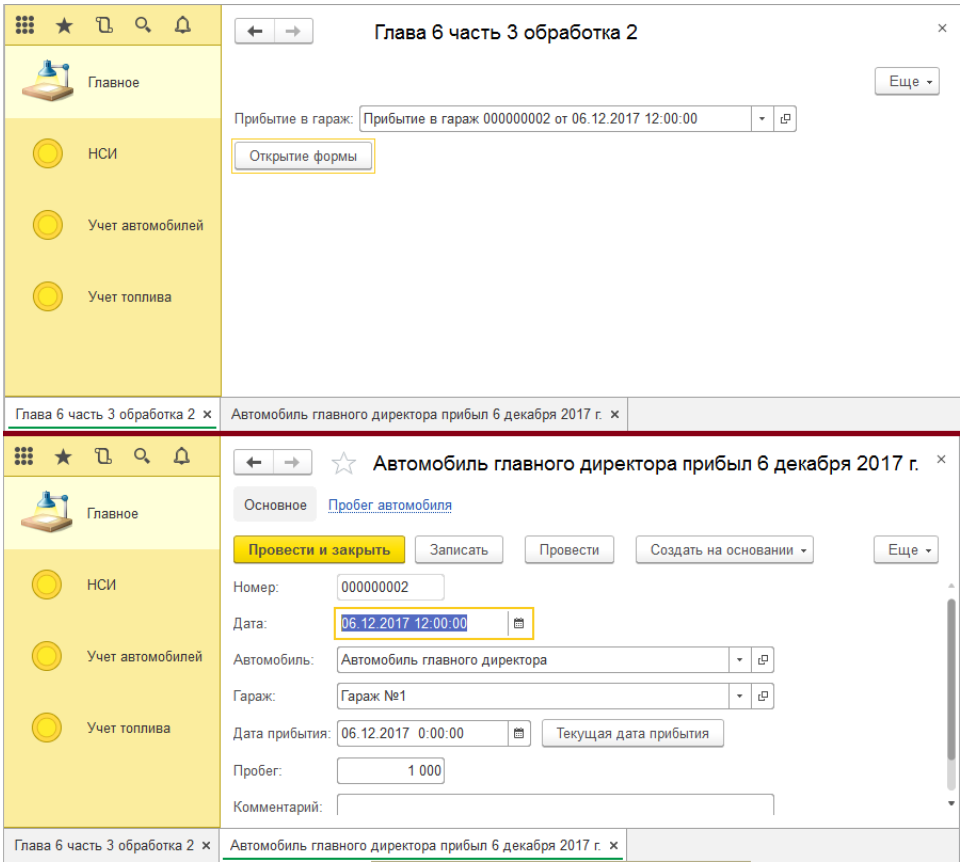


Рис. 6.3.59

Продолжим изучать открытие форм с параметрами.

Выполним небольшую задачку. Создадим новую обработку, форму обработки и на форме два реквизита - «Время» и «Расстояние» с типом Число (см. рис. 6.3.60), а также команду «Рассчитать скорость». *Эта форма будет основной формой обработки, т.е. будет открываться при запуске обработки (см. рис. 6.3.62).*

У новой обработки также создадим форму «Форма расчета скорости», на которой создадим два параметра «Время» и «Расстояние» (см. рис. 6.3.61), а также реквизит «Скорость», который разместим на форме.

При выполнении команды «Рассчитать скорость» основной формы будем открывать форму обработки «Форма расчета скорости», где будем считать скорость, деля расстояние на время и записывая результат в реквизит «Скорость».

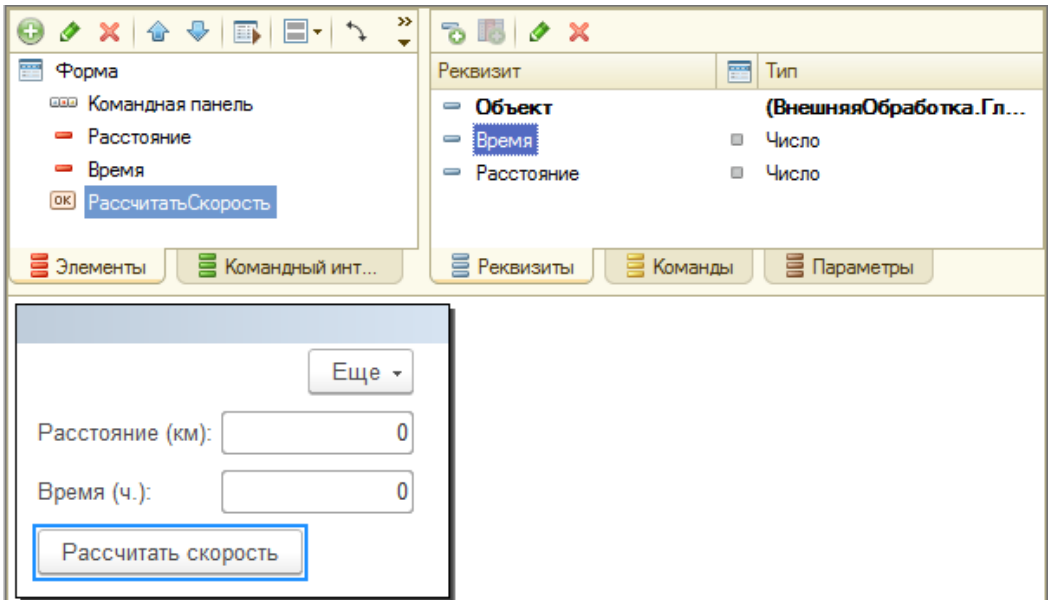


Рис. 6.3.60

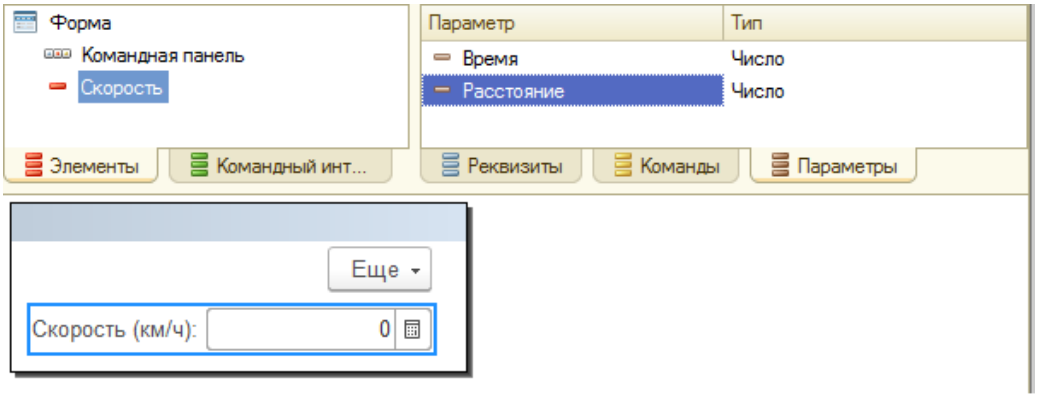


Рис. 6.3.61

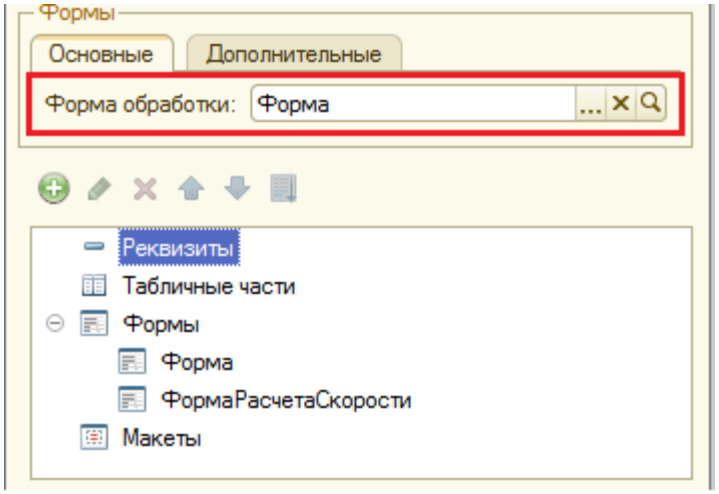


Рис. 6.3.62

Создадим обработчик для команды «Рассчитать скорость», в котором будем открывать форму «Форма расчета скорости» нашей обработки и передавать в метод открытия формы в качестве параметра структуру с двумя элементам. Названия ключей элементов соответствуют

названиям параметрам формы «Форма расчета скорости», а в значения элементов будут передаваться значения соответствующих реквизитов формы (см. листинг 6.3.13).

```
&НаКлиенте
Процедура РассчитатьСкорость(Команда)
    ПараметрыФормы = Новый Структура;
    ПараметрыФормы.Вставить("Расстояние",Расстояние);
    ПараметрыФормы.Вставить("Время",Время);

ОткрытьФорму("ВнешняяОбработка.Глава_6_Часть_3_Обработка_3.Форма.ФормаРасчета
Скорости",ПараметрыФормы);
КонецПроцедуры
```

Листинг 6.3.13

Теперь нам осталось эту скорость рассчитать, делать мы это будем в обработчике события *ПриСозданииНаСервере* формы «Форма расчета скорости», причем если параметр *Время* равен нулю, то форму не будем открывать (см. листинг 6.3.14).

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

    Если Параметры.Время = 0 Тогда
        Сообщить("Время равно 0");
        Отказ = истина;
        Возврат;
    КонецЕсли;

    Скорость = Параметры.Расстояние / Параметры.Время;

КонецПроцедуры
```

Листинг 6.3.14

Сохраним обработку и попробуем вычислить скорость (см. рис. 6.3.63 и 6.3.64).

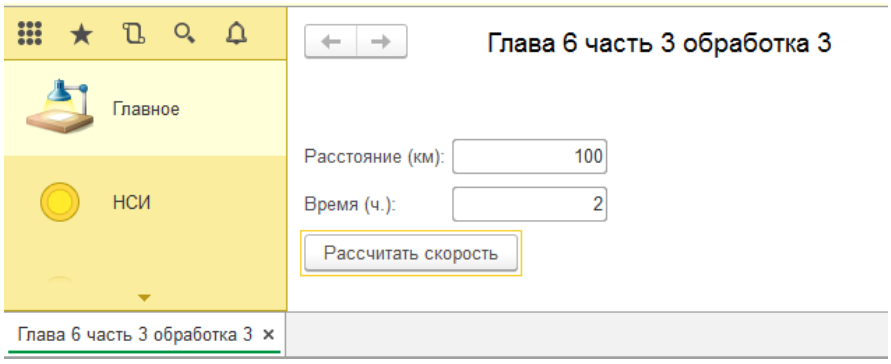


Рис. 6.3.63

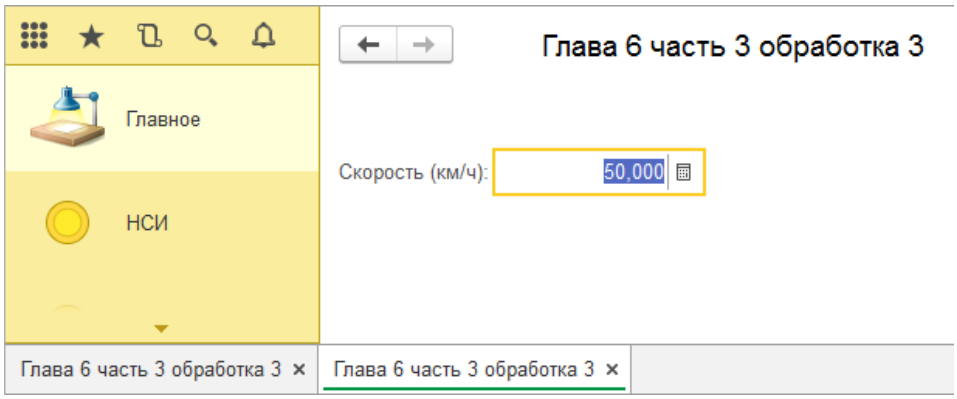


Рис. 6.3.64

Для красоты изменим заголовки обоих форм. Я это сделаю в конфигураторе (см. рис. 6.3.65 и 6.3.66), а Вы программно при открытии формы.

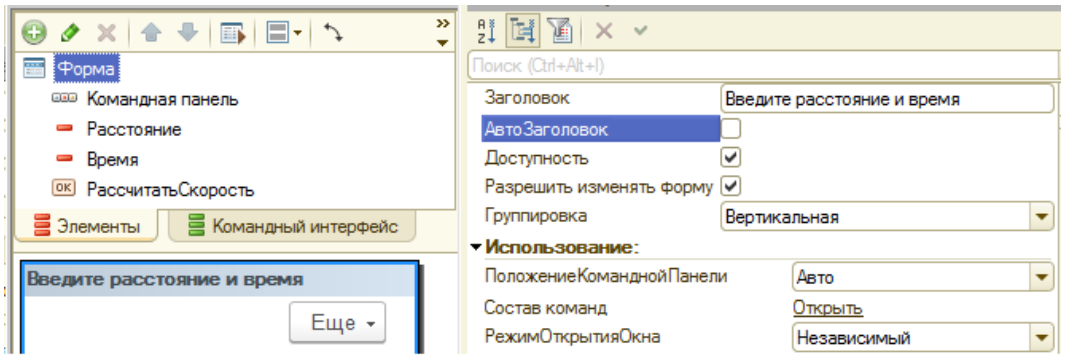


Рис. 6.3.65

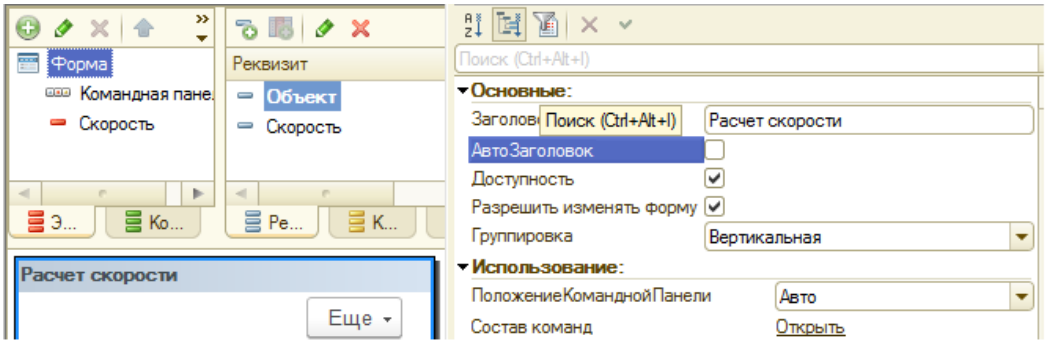


Рис. 6.3.66

Попробуем теперь рассчитать скорость (см. рис. 6.3.67 и 6.3.68).

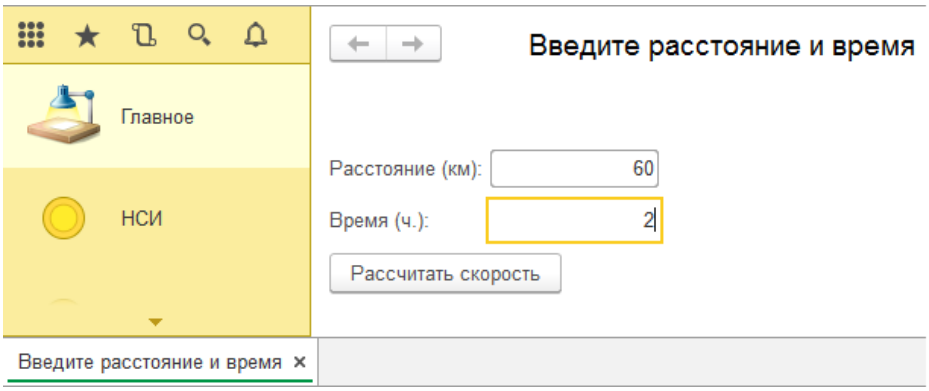


Рис. 6.3.67

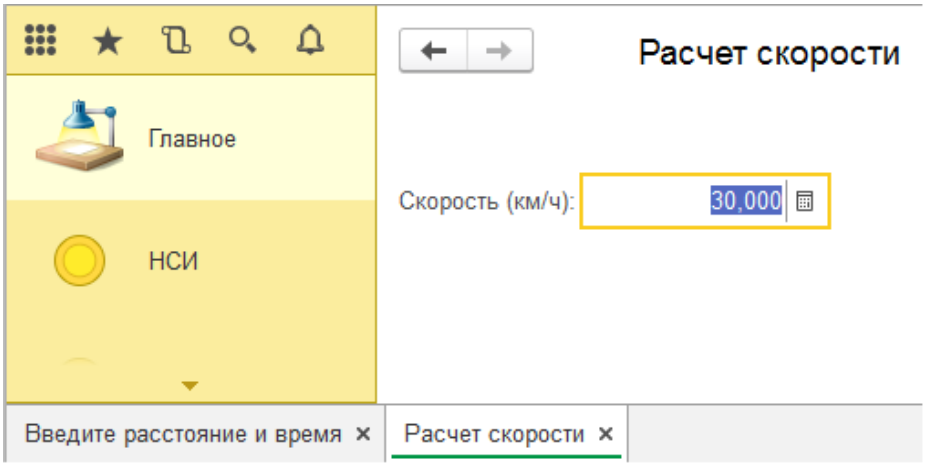


Рис. 6.3.68

Режимы открытия окна

Вы заметили, что каждый раз окна открываются в «соседнем окне», т.е. независимо. В то же время из пятой главы 5 (см. стр. 298) мы знаем, что существуют различные режимы открытия окна, которые задаются при помощи свойства формы *РежимОткрытияОкна*. Если мы присвоим свойству формы *РежимОткрытияОкна* какое-либо значение, то форма всегда будет открываться в этом режиме. В некоторых всегда по умолчанию установлен режим *Независимый.*

Можно ли программно открыть формы в других режимах, не меняя при этом свойство формы? Да, можно. Для этого необходимо использовать последний параметр метода «ОткрытьФорму – РежимОткрытияОкна».

ОткрытьФорму(<ИмяФормы>, <Параметры>, <Владелец>, <Уникальность>, <Окно>, <НавигационнаяСсылка>, <ОписаниеОповещенияОЗакрытии>, <РежимОткрытияОкна>)

В этот параметр необходимо передавать системное перечисление *РежимОткрытияОкнаФормы*, которое содержит следующие значения:

- БлокироватьВесьИнтрефейс
- БлокироватьОкноВладельца
- Независимый

Доработаем нашу предыдущую обработку, пусть форма расчета скорости открывается в отдельном окне, для этого будем её открывать с режимом *БлокироватьОкноВладельца* (см. листинг 6.3.15).

```
&НаКлиенте
Процедура РассчитатьСкорость(Команда)
    ПараметрыФормы = Новый Структура;
    ПараметрыФормы.Вставить("Расстояние",Расстояние);
    ПараметрыФормы.Вставить("Время",Время);

ОткрытьФорму("ВнешняяОбработка.Глава_6_Часть_3_Обработка_3.Форма.ФормаРасчета
Скорости",

ПараметрыФормы,,,,,,РежимОткрытияОкнаФормы.БлокироватьОкноВладельца);
КонецПроцедуры
```

Листинг 6.3.15

Посмотрим, как отработает расчет скорости (см. рис. 6.3.69).

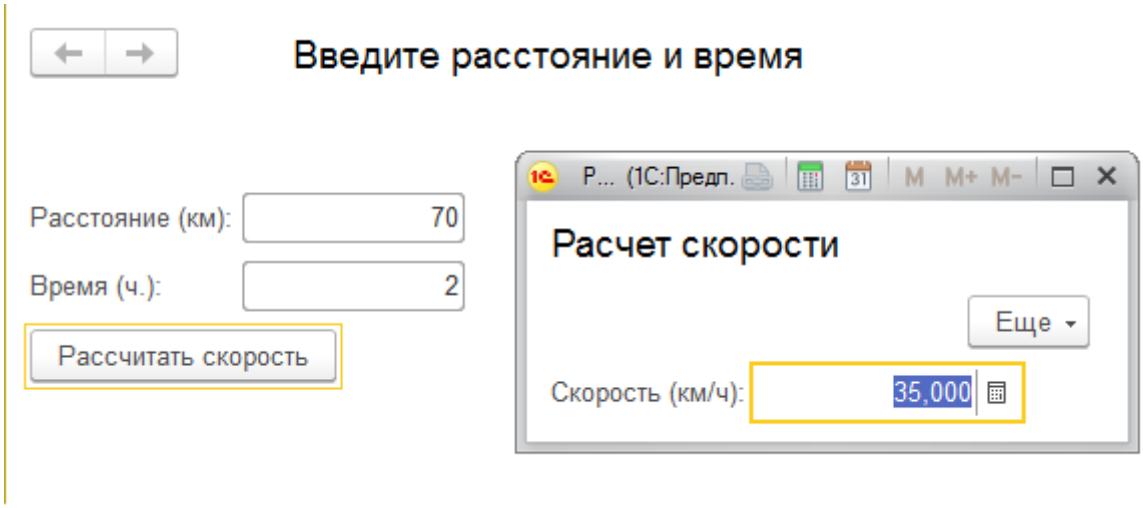


Рис. 6.3.69

Часть 4. Тип «Тип» функции «Тип» и «ТипЗнч»

Рассмотрим один примитивный тип, который мы не рассматривали во второй главе (поскольку это было бы преждевременно), - это тип *«Тип»*.

Да-да, этот тип так и называется - *«Тип»*, переменную данного типа нельзя задать самостоятельно, как переменные других примитивных типов. И она необходима для сравнения типов.

Сейчас мы научимся сравнивать типы. Как Вы уже знаете, все объекты 1С имеют определенный тип значения, это может быть ссылка на элемент справочника или объект какого-нибудь документа. Иногда необходимо определить, какой тип значения у того или иного объекта. И осуществляется это с помощью примитивного типа *«Тип»*. Для сравнения типов используются две функции: функция *Тип* и функция *ТипЗнч*.

Тип(<НазваниеТипа>)

ТипЗнч(<Переменная>)

Как Вы видите, синтаксис данных функций простой: у каждой всего по одному параметру. У функции *Тип* это название типа в кавычках по определенным правилам. А у функции *ТипЗнч* та переменная, тип которой мы желаем узнать.

Рассмотрим следующий пример:

Создайте форму, на которой разместите четыре реквизита: первый будет иметь тип значения *число*, второй – тип значения *дата*, третий – тип значения ссылка на какой-нибудь элемент справочника *Автомобили*, и четвертое – тип значения ссылка на какой-нибудь документ *Прибытие в гараж*. А также команду «ПроверитьТип».

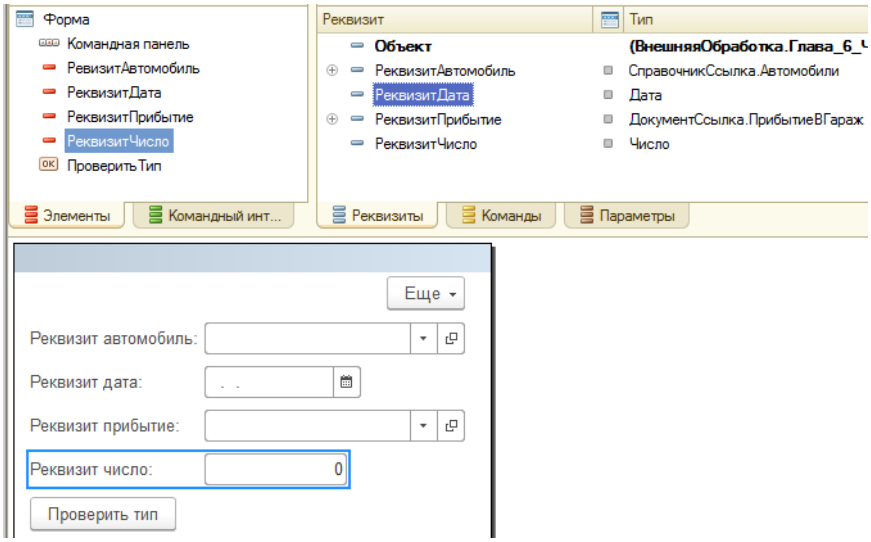


Рис. 6.4.1

Каждый раз, когда будет выбираться значение в поле ввода, глобальная переменная будет заменяться выбранным значением. При выполнении команды нужно будет определить, какое значение хранится в глобальной переменной.

В модуле формы зададим глобальную переменную *«Текущая».*

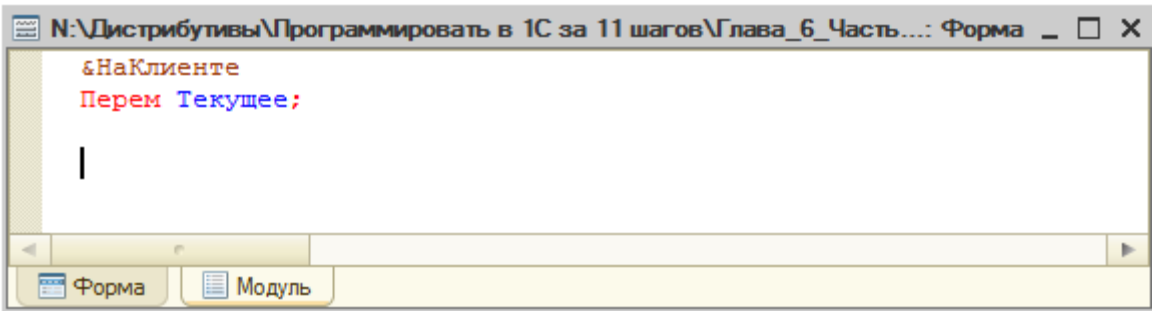


Рис. 6.4.2

Заметьте, мы её задали в клиентском контексте, потому что обращаться к ней будет также в клиентском контексте. Подробно о работе с глобальными переменными формы в книге «Основы разработки в 1С: Такси».

Создадим обработчики события *ПриИзменении* для каждого поля ввода и напишем в них следующий код:

```
&НаКлиенте
Процедура РевизитАвтомобильПриИзменении(Элемент)
    Текущее = РеквизитАвтомобиль;
КонецПроцедуры

&НаКлиенте
Процедура РеквизитДатаПриИзменении(Элемент)
    Текущее = РеквизитДата;
КонецПроцедуры

&НаКлиенте
Процедура РеквизитПрибытиеПриИзменении(Элемент)
    Текущее = РеквизитПрибытие;
КонецПроцедуры

&НаКлиенте
Процедура РеквизитЧислоПриИзменении(Элемент)
    Текущее = РеквизитЧисло;
КонецПроцедуры
```

Листинг 6.4.1

А в обработчике команды напишем следующий код:

```
&НаКлиенте
Процедура ПроверитьТип(Команда)
    Если ТипЗнч(Текущее) = Тип("Число") тогда
        Сообщить("Переменная ""Текущая"" имеет тип число");
    ИначеЕсли ТипЗнч(Текущее) = Тип("Строка") тогда
        Сообщить("Переменная ""Текущая"" имеет тип строка");
    ИначеЕсли ТипЗнч(Текущее) = Тип("СправочникСсылка.Автомобили") тогда
```

```
        Сообщить("Переменная ""Текущая"" имеет тип ссылка на элемент справ.
автомобили");
    ИначеЕсли ТипЗнч(Текущее) = Тип("ДокументСсылка.ПрибытиеВГараж") тогда
        Сообщить("Переменная ""Текущая"" имеет тип ссылку на документ
""Прибытие в гараж""");
    ИначеЕсли ТипЗнч(Текущее) = Тип("Неопределено") тогда
        Сообщить("Переменная ""Текущая"" имеет тип неопределено");
    КонецЕсли;
КонецПроцедуры
```

Листинг 6.4.2

Проверьте самостоятельно, как работает данная обработка.

Как Вы видите, мы точно знаем, какой текущий тип у глобальной переменной. Делается это с помощью двух функций *Тип* и *ТипЗнч*, они возвращают переменную примитивного типа *«Тип»*. Причем при работе с функцией *Тип* программист сам знает, какой тип должна вернуть данная функция, и задает его самостоятельно через кавычки, а функция *ТипЗнч,* наоборот, принимает любую переменную с любым типом (даже типом *неопределено*), и возвращает значение примитивного типа *«Тип»*.

Резюме

В шестой главе книги «Программировать в 1С за 11 шагов» мы узнали, что такое объектные типы и как с ними работать. Объем этой книги не позволяет изучить все объектные типы платформы 1С. Мы изучили самые основные: справочники, документы и формы. Этих знаний достаточно для того, чтобы начать программировать в 1С и делать какие-то элементарные вещи. Потом, в процессе работы, у Вас будет возможность углубить свои знания, но без хорошего базиса Вам будет очень сложно это сделать. Поэтому досконально изучите материал этой главы, выполните все домашние задания к ней. Все эти знания Вам в дальнейшем очень пригодятся.

Глава 7. Универсальные коллекции значений

Универсальные коллекции значений - это объекты, предназначенные для хранения временной информации, как правило, они не разрабатываются в конфигурации и не хранятся непосредственно в базе данных, а создаются только на этапе работы программы и уничтожаются при закрытии сеанса программы.

Тут необходимо небольшое уточнение: все объекты, кроме объектов метаданных, существуют только в рамках того контекста, в котором они созданы. То есть если общий объект или коллекция значений были созданы в модуле формы, то они будут уничтожены при закрытии формы, или если они были созданы в теле процедуре или функции, то также уничтожатся, когда процедура или функция будут выполнены.

Все универсальные коллекции значений создаются с помощью оператора *Новый*.

Рассмотрим создание общего объекта на примере массива.

Массив1 = Новый Массив;

Всё, теперь объект *Массив* создан и будет уничтожен либо когда закончит действие процедура, внутри которой он создан, либо, если *Массив1* - глобальная переменная внутри модуля (например, модуля объекта), пока не закончит работать контекст этого модуля, либо в том случае, когда данной переменной не будет присвоено значение *Неопределено*.

Начнем изучение коллекции значений с массивов.

Часть 1. Массивы

Что такое *Массив*? *Массив* в языке программирования 1С - это коллекция элементов, следующих друг за другом, которые могут быть доступны с помощью индекса. Элементы могут быть любого типа, в том числе и типа *Массив*. У каждого элемента в массиве есть уникальный номер, который называют *Индексом*. Посредством индекса можно получить доступ к данному элементу.

Все элементы в массиве упорядочены, т.е. первый элемент имеет индекс, равный нулю, второй - одному, и так далее.

Запомните! В программировании все индексы во всех коллекциях всегда начинаются с нуля!

Научимся создавать и заполнять массивы.

Конструктор массивов

В языке 1С есть понятие *Конструктор*. *Конструктор* - это то, как создается определенный объект. Все массивы создаются с помощью конструктора *Новый*.

Новый Массив (<КоличествоЭлементов1>…<КоличествоЭлементовN>)

Где:

«КоличествоЭлементов» - необязательный параметр. Он может быть один - тогда мы будем иметь *одномерный массив*, а также их может быть несколько - тогда мы имеем *многомерный массив*. Многомерные массивы мы рассмотрим в конце данной части. А пока все наши примеры будут с одномерными массивами. Коллекция *Массив* имеет следующую доступность: тонкий клиент, толстый клиент, мобильный клиент, сервер и т.д. Т.е. эту универсальную коллекцию значений можно использовать в любом контексте. Также переменные с типом *Массив* можно передавать с клиента на сервер и обратно (возможна сериализация).

Зададим, к примеру, одномерный массив из пяти элементов.

Самостоятельно создайте обработку, форму обработки, команду, которую разместите на форме, и в обработчике команды на клиенте напишите конструктор массива для пяти элементов.

Данный код будет выглядеть следующим образом:

```
&НаКлиенте
Процедура СозданиеМассива(Команда)

    МассивНовый = Новый Массив(5);

КонецПроцедуры
```

Листинг 7.1.1

Обращаю Ваше внимание, что когда мы создали данный массив, его элементы уже существуют, но они не заполнены значениями. Если Вы посмотрите на данный массив в отладчике, то увидите напротив каждого элемента тип *Неопределено* (см. рис. 7.1.1).

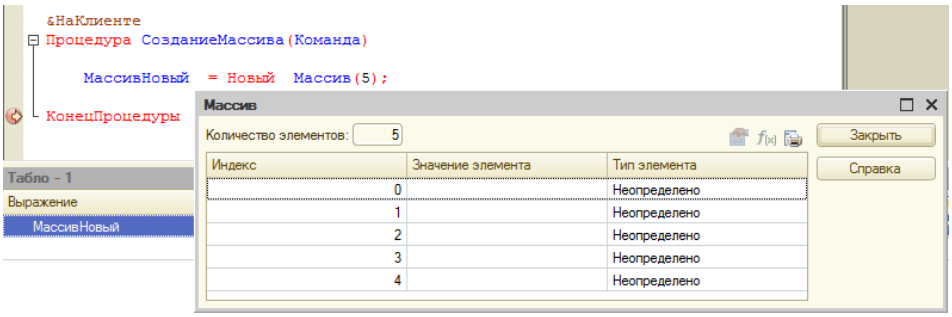


Рис. 7.1.1

Таким образом, мало создать массив, необходимо еще заполнить его.

Узнаем, каким образом обращаться к элементу массива посредством индекса. Делается это с помощью оператора «квадратные скобки». Допишем код в нашей обработке:

```
&НаКлиенте
Процедура СозданиеМассива(Команда)

    МассивНовый  = Новый  Массив(5);
    МассивНовый[0] = "Первый";
    МассивНовый[1] = 2;

КонецПроцедуры
```

Листинг 7.1.2

В данном примере с помощью квадратных скобок мы обратились к первому элементу массива, который имеет индекс ноль, а потом ко второму, имеющему индекс один. Как Вы уже поняли из примера, элементы массивов в 1С могут быть разных типов. Заполните остальные три элемента какой-нибудь информацией.

```
 МассивНовый[2] = 4;
МассивНовый[3] = "Четвертый";
МассивНовый[4] = 8;
```

Листинг 7.1.3

С помощью оператора «квадратные скобки» можно также и читать значения элемента массива.

```
Сообщить("МассивНовый[0]= " + МассивНовый[0] +
         ", МассивНовый[1] = " + МассивНовый[1]);
```

Листинг 7.1.4

Сохраните обработку, запустите ее в «1С:Предприятии» и посмотрите, что получилось.

Помимо создания массива с заранее определенным количеством элементов, можно задать массив, не указывая, сколько элементов он будет содержать. Делается это так:

МассивНовый2 = Новый Массив;

В данном случае будет создан массив, но никаких элементов в нем не будет (он будет пуст). Как работать с такими массивами, мы узнаем несколько позже.

Итак, мы научились создавать одномерный массив и заполнять его значениями, а также получать эти значения. Назовем этот вид наполнения массива *ручным способом*.

Обход массива

Рассмотрим, как нам обойти все элементы массива, чтобы прочитать их, или чтобы заполнить по нужному алгоритму. Можно сделать это, как называют, «китайским кодом», и

присвоить каждому элементу свой индекс. Типичный пример китайского кода это то, как мы заполнили элементы массива ранее. Иногда это бывает необходимо. Но иногда необходимо прочитать каждый элемент массива в цикле. Для этого в теле цикла нужно обойти индексы от первого до последнего. Сделаем это с помощью цикла *Для…Цикл*.

В ранее сконструированной обработке создадим новую команду, в обработчике которой создадим массив, заполним его и обойдем циклом:

```
&НаКлиенте
Процедура ОбходЦиклом(Команда)
    МассивНовый  = Новый  Массив(5);
    МассивНовый[0] = 3;
    МассивНовый[1] = 7;
    МассивНовый[2] = 7;
    МассивНовый[3] = 9;
    МассивНовый[4] = 1;

    Для н = 0 по 4 цикл

        Сообщить("МассивНовый[" + н + "] = " + МассивНовый[н] );

    КонецЦикла;
КонецПроцедуры
```

Листинг 7.1.5

Сохраните, запустите обработку и посмотрите, что получилось.

Тогда у внимательного читателя возникнет вопрос: каким образом узнать количество элементов массива, когда это не задано в конструкторе? Для этого у объекта *массив* существует метод *ВГраница*, который возвращает наибольший индекс в массиве. Индекс будет соответствовать количеству элементов массива минус один. Количество элементов массива также можно определить с помощью метода объекта *массив,* который называется *Количество*.

То есть цикл, который мы только что написали, можно переделать двумя равнозначными способами:

```
Для н = 0 по МассивНовый.ВГраница() цикл

    Сообщить("МассивНовый[" + н + "] = " + МассивНовый[н] );

КонецЦикла;
```

Листинг 7.1.6

И по-другому:

```
Для н = 0 по МассивНовый.Количество() - 1 цикл

    Сообщить("МассивНовый[" + н + "] = " + МассивНовый[н] );

КонецЦикла;
```

Листинг 7.1.7

Я рекомендую Вам использовать метод *ВГраница*, поскольку иной раз Вы можете забыть вычесть единицу из количества, что может привести к ошибкам.

Обход с помощью Для каждого… Цикл

В третьей главе мы с Вами изучили несколько циклов. В этой изучим новый незнакомый Вам вид цикла *Для каждого…Цикл*.

Мы не проходили этот цикл ранее, поскольку он применим только для коллекций значений и табличных частей.

Данный цикл имеет следующий синтаксис:

Для каждого <Элемент> из <Коллекция> цикл

//операторы

[Прервать;]

//операторы

[Продолжить;]

КонецЦикла

Приведенный оператор предназначен для циклического обхода коллекций.

Переменная *Коллекция* представляет именно ту коллекцию, которую мы обходим.

Переменной *Элемент* присваивается очередное значение элемента коллекции при каждом обходе.

Операторы *Прервать* и *Продолжить* Вам уже знакомы.

Рассмотрим этот вид цикла на примере обхода массива. Обойдем с помощью данного цикла уже созданный нами массив.

```
н = 0;
Для Каждого ЭлементМассива из МассивНовый цикл

    Сообщить("МассивНовый[" + н + "] = " + ЭлементМассива);

    н = н + 1;
КонецЦикла;
```

Листинг 7.1.8

Сохраните обработку и посмотрите, как работает данный цикл.

В данном коде переменную *«н»* мы ввели искусственно, чтобы пример был более наглядным. Как Вы уже поняли из примера, переменной *ЭлементМассива* в каждой итерации цикла присваивается соответствующий элемент массива *МассивНовый*.

Я рекомендую Вам использовать для обхода массивов в основном второй цикл, потому что в этом случае Вы получаете прямой доступ к элементу коллекции. И Вам не придется обращаться к нему посредством оператора *Квадратные скобки*. И для всех последующих универсальных коллекций мы будем применять оператор цикла *Для каждого…Цикл*.

Методы массивов

Поскольку массив в языке программирования 1С является объектом, то у него есть собственные методы. Некоторые мы уже разобрали ранее, это *Количество* и *ВГраница*. Сейчас мы рассмотрим остальные методы, которые будут для Вас необходимы.

Метод «Добавить»

С помощью данного метода добавляется новый элемент в конец массива. Метод содержит один параметр, это значение того элемента, который добавляется. Также он может быть и без параметра, тогда будет добавлен элемент со значением *Неопределено*.

Создайте новую команду формы, в которой будете рассматривать метод *Добавить*.

Покажем, как работает метод «Добавить». Для этого создадим массив из трех элементов, а потом добавим еще один. И выведем новый массив в окно сообщений:

```
&НаКлиенте
Процедура МетодМассиваДобавить(Команда)

    Массив1 = Новый Массив(3);
    Для н = 0 по Массив1.ВГраница() цикл
        Массив1[н] = н;
    КонецЦикла;
    Массив1.Добавить(1000);

    н = 0;
    Для Каждого ЭлМассив1 из Массив1 цикл
        Сообщить("Массив1["+ н + "] = " + ЭлМассив1 );
        н = н +1;
    КонецЦикла;

КонецПроцедуры
```

Листинг 7.1.9

Сохраните обработку и посмотрите на результат.

Теперь создадим вообще пустой массив и перебором добавим 10 элементов, а после выведем их в окно сообщений.

```
Массив2 = Новый Массив;

Для н = 1 по 10 цикл
    Массив2.Добавить(Строка(н)+ "* 2 = " + Строка(н*2));
КонецЦикла;

Для Каждого ЭлМассив2 из Массив2 цикл
    Сообщить(ЭлМассив2);
КонецЦикла;
```

Листинг 7.1.10

Сохраните обработку и посмотрите на результат.

Разберем данный код:

Первоначально мы создаем массив из трех элементов и заполняем его числами от *0* до *2*, затем добавляем новый элемент и выводим на экран все элементы (листинг 7.1.9). Потом мы создаем массив без элементов, в цикле добавляем элементы в виде строки и выводим их на экран (листинг 7.1.10).

С помощью метода *Добавить*, можно добавлять любой элемент (и даже массив), причем данный элемент будет иметь последний индекс в массиве. Т.е. вставать в конец массива.

Следующим разберем похожий метод – *Вставить*.

Метод «Вставить»

Данный метод похож на предыдущий, но, в отличие от *Добавить* (который добавляет новый элемент в конец массива), метод *Вставить* вставляет этот элемент по указанному в процедуре индексу. Все остальные элементы после данного индекса сдвигаются на плюс один. Если указанный индекс заходит за границу массива, то недостающие элементы заполнятся значением *Неопределено*.

Синтаксис данного метода следующий:

Вставить(<Индекс>, Элемент).

Здесь параметр *Индекс* – индекс вставляемого значения, *Элемент* – вставляемое значение.

Создайте новую команду, в обработчике которой создадим массив, заполним его в цикле 5-ю элементами, а потом вставим элемент в середину массива. Но кроме этого, добавим элемент гораздо выше границы массива.

```
&НаКлиенте
Процедура МетодМассиваВставить(Команда)

    Массив = Новый Массив;
    Для н = 1 по 5 цикл
        Массив.Добавить(Строка(н)+ "* 2 = " + Строка(н*2));
    КонецЦикла;
    Массив.Вставить(2, 100);
    Массив.Вставить(10,10000);

КонецПроцедуры
```

Листинг 7.1.11

Выведем на экран элементы массива.

```
Для Каждого ЭлМассив из Массив цикл
    Сообщить(?(ЭлМассив = Неопределено, "Неопределено",ЭлМассив));
КонецЦикла;
```

Листинг 7.1.12

Сохраните обработку и выполните команду. Вы увидите, что сначала элемент вставился в центр массива, а во втором - в конец массива, но между предыдущим крайним и последним появились элементы со значением *Неопределено*.

Добавлять элементы массива мы научились, теперь рассмотрим, как удалять их из массива.

Метод «Удалить»

Для удаления конкретного элемента массива используется метод *Удалить*, единственным параметром которого является *индекс* данного элемента. Это значение типа число (целое и положительное) от нуля и выше. Если указан индекс выше верхней границы массива, то ничего не произойдет и ошибки не будет.

Запомните: после того, как Вы удалите данный элемент, все индексы в массиве сместятся, на место данного элемента придут те, которые стояли после него, и т.д. Помните об этом, когда захотите удалить несколько каких-нибудь элементов.

Создадим новую команду и в обработчике этой команды создадим массив, который заполним 10-ю элементами:

```
&НаКлиенте
Процедура МетодМассиваУдалить(Команда)
    Массив = Новый Массив;
    Для н = 1 по 10 цикл
        Массив.Добавить(Строка(н)+ "* 2 = " + Строка(н*2));
    КонецЦикла;
КонецПроцедуры
```

Листинг 7.1.13

Удалим у этого массива первые три элемента сверху.

```
ИндексУдаления = 1;
Пока ИндексУдаления <= 3 Цикл
    Массив2.Удалить(0);
    ИндексУдаления = ИндексУдаления + 1;
КонецЦикла;
```

Листинг 7.1.14

В данном коде сначала мы удаляем три элемента с индексом *0*, поскольку при удалении элемента все индексы смещаются вверх, и каждый последующий элемент в массиве становится первым, и мы его удаляем.

А потом три снизу:

```
ИндексУдаления = 1;
Пока ИндексУдаления <= 3 цикл
    Массив.Удалить(Массив.ВГраница());
    ИндексУдаления = ИндексУдаления + 1;
КонецЦикла;
```

Листинг 7.1.15

Мы удаляем все элементы с индексом верхней границы, понятно, что при удалении элемента верхняя граница уменьшается, и каждый раз в итерации мы удаляем элемент, имеющий индекс верхней границы.

Выведем итоговый массив в окно сообщений:

```
Для Каждого ЭлМассив из Массив цикл
    Сообщить(ЭлМассив);
КонецЦикла;
```

Листинг 7.1.16

Метод «Очистить»

Метод *Очистить* удаляет все элементы массива. Удалим их из массива *Массив*, в обработчике, который сделали в предыдущем подразделе.

```
Массив.Очистить();
Сообщить("Массив.Количество() = " + Массив.Количество());
```

Листинг 7.1.17

Сохраните и запустите обработку. И Вы увидите, что количество элементов в массиве равно нулю. Он очистился.

Метод «Установить»

Как Вы видели из предыдущих примеров, значения элементам массива мы присваивали с помощью оператора квадратные скобки. Есть метод аналогичный данным скобкам, который называется *Установить*. Этот метод устанавливает значение элемента по нужному индексу. Рассмотрим синтаксис данного метода:

Установить(<Индекс>, <Значение>)

Первый параметр - это индекс элемента, которому будет устанавливаться значение. Второй параметр – значение элемента.

Рассмотрим пример: создадим новую команду, в обработчике которой создадим массив с количеством элементов, равным *10,* и заполним его четными числами с помощью метода *Установить*. А потом все это выведем в окно сообщений.

```
&НаКлиенте
Процедура МетодМассиваУстановить(Команда)

    Массив = Новый Массив(10);

    ИндексМассива = 0;
    ИндексСчетчика = 1;

    Пока ИндексМассива <= Массив.ВГраница() цикл

        Если ИндексСчетчика/2 = Цел(ИндексСчетчика/2) тогда
            Массив.Установить(ИндексМассива,ИндексСчетчика);
            ИндексМассива = ИндексМассива + 1;
        КонецЕсли;
        ИндексСчетчика = ИндексСчетчика + 1;

    КонецЦикла;

    Для Каждого ЭлМассива из Массив цикл
        Сообщить(ЭлМассива);
    КонецЦикла;

КонецПроцедуры
```

Листинг 7.1.18

Данный код должен быть Вам понятен, единственно, что мы вместо квадратных скобок использовали метод *Установить*.

Мы рассмотрели основные методы массивов, их Вам будет достаточно, чтобы проводить любые манипуляции с данным объектом.

А теперь узнаем, что такое многомерные массивы.

Многомерные массивы

Многомерным массивом в языке 1С называется массив, элементами которого являются массивы. Если одномерный массив создавался посредством конструктора с одним параметром, то в многомерном массиве может быть два и больше параметров.

Создайте новую обработку, в которой разберем многомерные массивы.

Пример создания многомерного массива с двумя параметрами:

МассивМн1 = Массив(3,2);

В этом примере мы создали массив, у которого будет три элемента, каждый из которых будет массивом, содержащим два элемента.

Заполним его. Для этого нам необходимо будет заполнить внутренние массивы. Сделаем это в команде на форме новой обработки.

```
&НаКлиенте
Процедура ЗаполнитьМногомерныйМассив(Команда)

    МассивМн1 = Новый Массив(3,2);

    Для  н = 0 По МассивМн1.ВГраница() цикл
        МассивВнутренний = МассивМн1[н];
        Для к = 0 по МассивВнутренний.ВГраница() цикл
            МассивВнутренний[к] = Строка(н+1)+ "." + Строка(к+1);
        КонецЦикла;
    КонецЦикла;

КонецПроцедуры
```

Листинг 7.1.19

В данном коде в первом цикле мы получаем элементы основного массива, напомню, что каждый элемент является массивом. Во внутреннем цикле мы заполняем непосредственно внутренний массив.

Выведем этот массив на экран:

```
Для Каждого ЭлМассиваВнеш из МассивМн1 цикл
    СтрокаВывода = "";
    Для Каждого ЭлемМассиваВнутр из ЭлМассиваВнеш цикл
          СтрокаВывода = СтрокаВывода + " " + ЭлемМассиваВнутр;
    КонецЦикла;
    Сообщить(СокрЛ(СтрокаВывода));
КонецЦикла;
```

Листинг 7.1.20

В этом коде мы получаем каждый элемент массива, который тоже является массивом, и уже у этого массива выводим элементы на экран. Сохраните обработку и посмотрите, что получилось.

Должен выйти такой результат:

Рис. 7.1.2

Как Вы видите, вышло три строки, ведь мы задали изначально три верхних массива, и в каждой строке две цифры, которые являются значениями внутреннего массива, который состоит из двух элементов.

Многомерные массивы можно создавать, не используя параметры в конструкторе массивов, для этого достаточно установить значению элемента другой массив.

Рассмотрим пример: зададим массив без параметров и будем добавлять элементы массива внутри цикла, причем каждый элемент будет являться массивом.

```
&НаКлиенте
Процедура МногомерныйМассивБезПараметров(Команда)

    Массив = Новый Массив;
    Для н = 1 по 5 цикл
        МассивВн = Новый Массив;
        Для к = 1 по 7 цикл
            МассивВн.Добавить(Строка(н) +"." + Строка(к));
        КонецЦикла;
        Массив.Добавить(МассивВн);
    КонецЦикла;

КонецПроцедуры
```

Листинг 7.1.21

Как видите, в данном коде в первом цикле мы создаем новый массив и заполняем его элементами. После того, как заполнили массив внутри цикла элементами, мы добавляем его во внешний массив в виде элемента.

Выведите этот массив на экран самостоятельно.

Вы можете создавать массивы какой угодно формы: трехмерные, четырехмерные, или такие, где внутри массива будут массивы с разным количеством элементов, и т.п.

Резюме

Итак, мы научились работать с массивами. Я не стал Вам давать такие вещи, как сортировка и свертка массивов, по двум причинам. Во-первых, это общие понятия программирования, и они есть в любом учебнике по программированию, и к работе с 1С не

имеют никакого отношения. Во-вторых, в языке 1С есть более универсальные объекты типа списков значений и таблиц значений, в которых все нужные методы по сортировке, свертке и т.п. реализованы. Оба этих объекта мы обязательно изучим.

И поэтому если перед Вами стоит задача, в которой имеющиеся данные необходимо будет сортировать или свернуть, то необходимо использовать вышеупомянутые объекты, а не массивы. Тем не менее, сами массивы необходимо знать и уметь с ними работать, поскольку многие методы некоторых объектов возвращают при своем исполнении массивы.

Часть 2. Структура

Структура в языке программирования 1С - это коллекция некоторых значений в связке с ключом. Эта связка ключа со значением называется *«КлючИЗначение»*. Ключ структуры уникален в рамках данной структуры. Причем к значениям структуры можно обращаться как к свойствам объекта, используя название ключа.

Объект *Структура* создается с помощью конструктора *Новый*.

Структура1 = Новый Структура;

Структуру можно создать на всех видах клиентов и в любом контексте. Сериализируется, т.е. можно передать переменную с типом структура с клиентского контекста на серверный. Хоть и в значение можно записывать любой тип, использование типов в этом случае очень сильно зависит от контекста и от вида клиента: мы не можем на клиенте задать в значение тип, который работает только в серверном контексте (например, ДокументОбъект.<>).

Работа со структурой

Изучим, как нам создавать структуру, создавать пару *«Ключ и значение»*, и как читать значения структуры, используя ключи.

Для этого создадим обработку, в которой и будете проводить все наши эксперименты со структурами.

Как создать новую структуру, Вы знаете, теперь выясним, как создаются новые элементы данного объекта. Делается это с помощь метода *Вставить*.

Вот его синтаксис:

Вставить(<Ключ>,<Значение>);

Первый параметр - это ключ устанавливаемого элемента.

Второй параметр - непосредственно устанавливаемый элемент.

Параметр *Ключ* имеет тип значения *Строка*. Он может иметь любое название, какое захочет разработчик. Параметр *Значение* может иметь любой тип.

Обращаю Ваше внимание, что связка *«Ключ и значение»* уникальна, поэтому если Вы напишете для одной структуры два метода *Вставить* с одинаковыми ключами и разными значениями, то все равно в структуре будет одна связка *«Ключ и значение»*, причем значение возьмется с последнего метода.

Создайте форму, команду и обработчик этой команды на клиенте, а в обработчике создадим простую структуру с двумя парами *«Ключ и значение».*

```
&НаКлиенте
Процедура СоздатьСтруктуру(Команда)

    Структура1 = Новый Структура;
    Структура1.Вставить("Ключ1",12.3);
    Структура1.Вставить("Ключ2",33.4);

КонецПроцедуры
```

Листинг 7.2.1

В этом случае мы не стали думать над названиями ключей, Вы же в процессе работы можете давать ключам любые названия, какие захотите. Главное, чтобы тип ключа был *Строка*.

Посмотрим в отладчике, как выглядит данный объект (см. рис. 7.2.1).

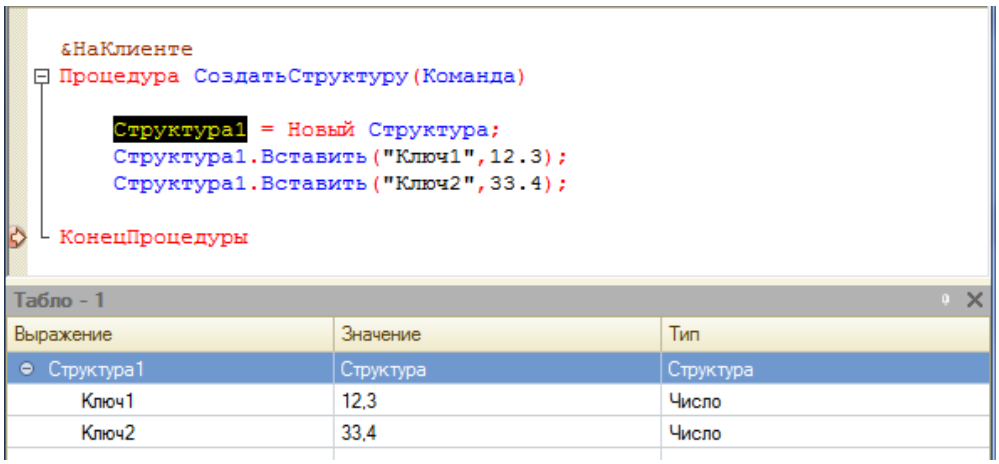


Рис. 7.2.1

Как видите, в отладке данная структура представляет собой некоторый список, где напротив каждого ключа есть то значение, которое мы привязали к данному ключу с помощью метода *Вставить*. Данный ключ является свойством структуры как объекта, и мы можем обращаться к нему.

Для того чтобы посмотреть, как это работает, допишем наш код:

```
Сообщить("Структура1.Ключ1 = " + Строка(Структура1.Ключ1));
Сообщить("Структура1.Ключ2 = " + Строка(Структура1.Ключ2));
```

Листинг 7.2.2

Как Вы видите из кода, мы обращаемся к обоим ключам как к свойствам структуры. По сути, они и есть свойства данной структуры, где хранятся соответствующие значения.

Сохраним и запустим обработку. Выполним команду и посмотрим, что получится.

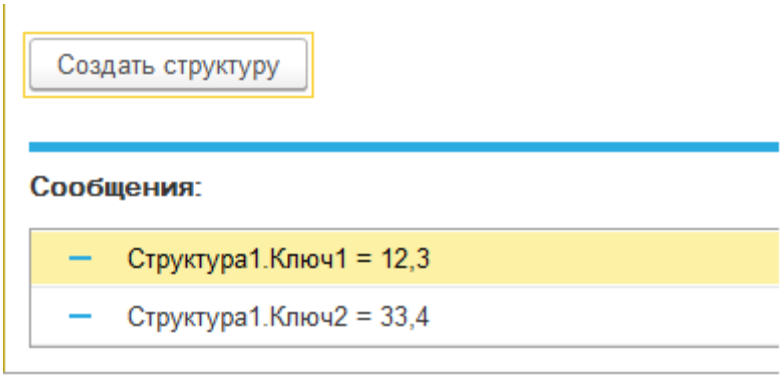


Рис. 7.2.2

Вы видите, в окно сообщений вышли те значения, которые мы привязали к ключам с помощью метода *Вставить*.

Не всегда обязательно использовать метод *Вставить*, чтобы добавить пару *КлючИЗначение* в структуру, иногда это можно сделать в конструкторе. Тогда конструктор будет иметь следующий вид:

Структура1 = Новый Структура(Ключ, Значение);

Переделаем предыдущий пример:

```
Структура1 = Новый Структура("Ключ1, Ключ2",12.3,33.4);
Сообщить("Структура1.Ключ1 = " + Строка(Структура1.Ключ1));
Сообщить("Структура1.Ключ2 = " + Строка(Структура1.Ключ2));
```

Листинг 7.2.3

В этом варианте конструктора все ключи должна задаваться в одной строкой, а значения этих ключей должны быть перечислены через запятую.

Изменить значение ключа

Если нам необходимо изменить значение какого-нибудь ключа, то мы, используя метод *Вставить*, указываем в качестве первого параметра ключ, значение которого хотим поменять, а в качестве второго параметра новое значение для данного ключа.

Переделаем обработку.

```
&НаКлиенте
Процедура СоздатьСтруктуру(Команда)
    Структура1 = Новый Структура("Ключ1, Ключ2",12.3,33.4);
    Структура1.Вставить("Ключ1",1200);
    Сообщить("Структура1.Ключ1 = " + Строка(Структура1.Ключ1));
    Сообщить("Структура1.Ключ2 = " + Строка(Структура1.Ключ2));
КонецПроцедуры
```

Листинг 7.2.4

Сохраните ее и смотрите на результат работы.

Количество элементов

Для того, чтобы узнать количество пар *«Ключ и значение»*, существует метод *Количество*.

Допишите в Вашу обработку следующий код, и посмотрите, какое количество элементов у созданной Вами структуры.

```
КоличествоЭлементов = Структура1.Количество();
Сообщить("В структуре1 количество элементов = " + КоличествоЭлементов);
```

Листинг 7.2.5

Обход коллекции

Обход структуры осуществляется с помощью уже знакомого нам оператора цикла *Для каждого…Цикл*.

Доработаем текущий пример: обойдем уже созданную структуру.

```
Для Каждого ЭлСтруктуры из Структура1 цикл
    Сообщить(Строка(ЭлСтруктуры.Ключ) + " = " + ЭлСтруктуры.Значение);
КонецЦикла;
```

Листинг 7.2.6

В этом коде переменная *ЭлСтруктуры* принимает значение каждого элемента структуры в зависимости от итерации. Данная переменная является парой *«Ключ и значение»*, поэтому мы можем прочитать у данной переменной ключ и значение, которое соответствует данному ключу.

Тип значений

В структуру можно записывать не только примитивные типы, но также любые другие объекты «1С:Предприятия», вплоть до других структур. Причем значения типов разных ключей структуры могут быть разными. Продемонстрируем это – запишем в структуру массив, ссылку на объект справочника и сам объект справочника. Для этого в Вашей обработке создадим новую команду, в которой и проделаем все вышеперечисленные манипуляции. Поскольку мы не можем в клиентском контексте получить доступ к объекту метаданных, то создайте для команды обработчик на клиенте и на сервере без контекста. В этих обработчиках напишем следующий код:

```
&НаСервереБезКонтекста
Процедура СоздатьСтруктуруСРазнымиЗначениямиНаСервере()

    Структура1 = Новый Структура;

    МассивДат = Новый Массив;
    Для н = 0 по 4 цикл
        МассивДат.Добавить(ТекущаяДата() + 3600*24*н);
    КонецЦикла;

    МаркаFordCсылка =
Справочники.МаркиАвтомобилей.НайтиПоНаименованию("Ford",Истина);
    Если НЕ МаркаFordCсылка.Пустая()  Тогда
        МаркаFordОбъект = МаркаFordCсылка.ПолучитьОбъект();
    иначе
        МаркаFordОбъект = Неопределено;
    КонецЕсли;

    Структура1.Вставить("МассивДат",МассивДат);
    Структура1.Вставить("МаркаФордОбъект",МаркаFordОбъект);
    Структура1.Вставить("МаркаФордСсылка",МаркаFordCсылка);


КонецПроцедуры

&НаКлиенте
Процедура СоздатьСтруктуруСРазнымиЗначениями(Команда)
    СоздатьСтруктуруСРазнымиЗначениямиНаСервере();
КонецПроцедуры
```

Листинг 7.2.7

Как Вы видите, мы вставили абсолютно разные объекты в значения структуры. Причем обращаясь к значению через ключ, мы получаем доступ непосредственно к самому объекту. Допишите следующий код в Вашей обработке:

```
Для Каждого СтрокаЭл из Структура1.МассивДат цикл
    Сообщить(Формат(СтрокаЭл,"ДЛФ=DD"));
КонецЦикла;
Сообщить(Структура1.МаркаГазСсылка.Наименование);
```

Листинг 7.2.8

Сохраните и запустите Вашу обработку. Посмотрим, как она работает (см. рис. 7.2.3).

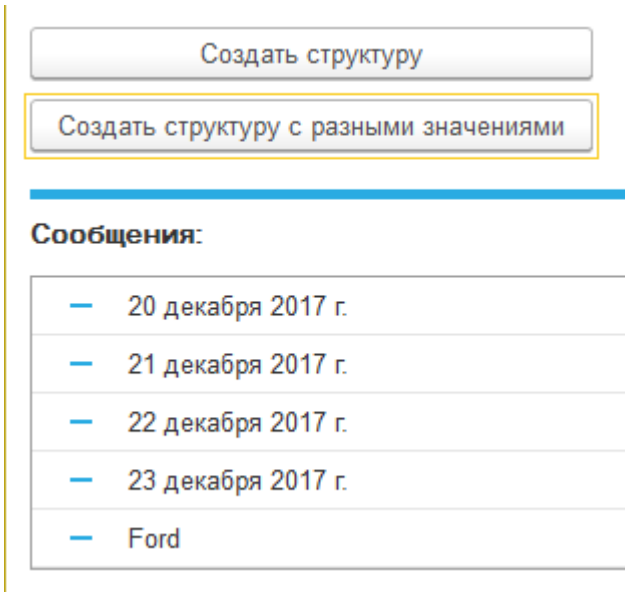


Рис. 7.2.3

Вы видите, что обращаясь к свойствам структуры посредством ключа, мы обращаемся непосредственно к объекту и можем проделать с ним любые манипуляции.

Обращаю Ваше внимание: если Вы проведете какие-нибудь манипуляции со значением структуры, то данные манипуляции отобразятся на той переменной, которую Вы записали в это значение.

Для демонстрации вышеприведенного утверждения, изменим предыдущий пример: у массива внутри структуры добавим новый элемент и посмотрим, как это отобразится на самом массиве, который мы записали в структуру.

```
Структура1.МассивДат.Добавить(ТекущаяДата() + 3600*24*5);
Для Каждого СтрокаЭл из МассивДат цикл
    Сообщить(Формат(СтрокаЭл,"ДЛФ=DD"));
КонецЦикла;
```

Листинг 7.2.9

Сохраните и перезапустите обработку. Как Вы увидите, элемент, который мы добавили в значение структуры, появился у изначального массива, поскольку в данной структуре содержится не копия переданного массива, а *указатель или ссылка* на сам массив, к которому можно обратиться посредством структуры и ключа.

Безошибочное получение значения элемента

Если при обращении к ключу структуры Вы укажете название несуществующего в данной структуре ключа, то программа выдаст ошибку. Но есть метод объекта *Структура*, с помощью которого можно обратиться к значению ключа, не боясь вызвать ошибку в случае промаха.

Этот метод – *Свойство*.

Данный метод является функцией и возвращает *Истину*, если указанный ключ есть, и *Ложь,* если указанного ключа нет.

Рассмотрим синтаксис метода:

Свойство(<ИмяКлюча>,<ЗаписываемоеЗначение>);

«ИмяКлюча» - название ключа для поиска свойства.

В параметр *«ЗаписываемоеЗначение»* будет возвращено найденное значение. В том случае, если ключа нет в структуре, то данному параметру присвоится значение *Неопределено*. Обращаю Ваше внимание, что параметр «*ЗаписываемоеЗначение»* где-то должен быть определен. Причем данный параметр также будет являться указателем на первоначальный объект.

Рассмотрим работу данного свойства.

Создайте новую команду формы и обработчик этой команды на клиенте. В обработчике создадим структуру и запишем в неё массив дат из предыдущих примеров. Также в начале кода в обработчике объявим новую переменную.

```
&НаКлиенте
Процедура РаботаСоСвойствомСтруктуры(Команда)
    Перем МассивДатНайденный;

    МассивДат = Новый Массив;
    Для н = 0 по 4 цикл
        МассивДат.Добавить(ТекущаяДата() + 3600*24*н);
    КонецЦикла;
    Структура1 = Новый Структура;
    Структура1.Вставить("МассивДат",МассивДат);

КонецПроцедуры
```

Листинг 7.2.10

Запишем в переменную *МассивДатНайденный* при помощи метода *Свойство* структуры массив *МассивДат* из структуры, а после изменим *эту* переменную, для того чтобы убедиться, что изначальный массив, который в структуре, тоже изменится.

Допишем код:

```
ЕстьМассив = Структура1.Свойство("МассивДат",МассивДатНайденный);

Если ЕстьМассив тогда
    МассивДатНайденный.Добавить(ТекущаяДата() + 3600*24*7);
КонецЕсли;

Для Каждого СтрокаЭл из МассивДат цикл
    Сообщить(СтрокаЭл);
КонецЦикла;
```

Листинг 7.2.11

Самостоятельно посмотрите, как работает данный код. Вы увидите, что изначальный массив, который в структуре, тоже изменился. Т.е. переменная *МассивДатНайденный* содержит указатель на переменную *МассивДат*.

Очистка структуры и удаление элементов

Выясним, каким образом можно удалить все пары *Ключ и значение*, или какую-то одну пару.

Для удаления всех пар используется метод *Очистить*.

Выглядит это так:

```
Структура1.Очистить();
```

После данной процедуры никаких пар *Ключ и значение* в данной структуре не будет.

Замечу, что метод «Очистить» применим для всех универсальных коллекций, поэтому в дальнейшем, при изучении других объектов коллекции, мы не будем на нем особо останавливаться.

Если необходимо удалить конкретную пару *Ключ и значение,* используется метод *Удалить*, с единственным параметром – ключ удаляемого значения.

```
Структура1.Удалить("МаркаГаз");
```

Если Вы введете неправильное название ключа, то ничего не произойдет, и ошибку отладчик не выдаст.

Поэкспериментируйте самостоятельно с данными методами структуры.

Резюме

В этой части урока мы научились работать со структурами, это очень важный объект базы, поскольку многие методы принимают в качестве параметра структуру с нужными ключами. Это не очень сложный объект для понимания, и если у Вас на начальных этапах возникли трудности, то рекомендую открывать данный объект в отладке, чтобы посмотреть, как он работает, как выглядят ключи и какие значения к этим ключам привязываются.

Часть 3. Соответствие

Соответствие в языке программирования 1С чем-то похоже на *Структуру*, это тоже коллекция пар *Ключ и значение*. Но в отличие от *Структуры*, в *Соответствие*, чтобы получить значение по определенному ключу, нужно обращаться не как к свойству объекта, а через квадратные скобки, и в качестве ключа можно использовать значение **любого типа**.

Структура	Соответствие
Структура1.Ключ1	Соответствие1["Ключ1"]

Соответствие можно создать на всех видах клиентов и в любом контексте. Сериализируется, т.е. можно передать переменную с типом структура с клиентского контекста на серверный.

Хоть и в ключ и значение соответствия можно записать переменные любого типа, использование типов в этом случае очень сильно зависит от контекста и от вида клиента: мы не можем в клиентском задать в ключ или значение, тип которого работает только в серверном контексте (например, ДокументОбъект.<>).

Работа с соответствием

Работа с *Соответствием* чем-то похожа на работу со *Структурой*. Мы рассмотрим основные методы объекта *Соответствие*, разберем, каким образом осуществляется обход *Соответствия*, и чем кардинально отличается *Соответствие* от *Структуры*.

Создайте новую обработку и форму обработки, в которой Вы будете делать все приведенные ниже примеры.

Метод «Вставить»

Метод *Вставить* для *Соответствия* работает точно так же, как для *Структуры*.

Он имеет следующий синтаксис:

Вставить(<Ключ>, <Значение>)

*Внешне данный метод не отличается от метода структуры, с той лишь разницей, что «Ключ» может иметь **любой** тип! Это очень важное отличие, с помощью него Вы сможете в процессе работы устанавливать соответствие любых объектов с любыми объектами.*

Параметр *Значение* тоже может принимать любой тип.

Сделайте следующий пример: установите соответствие с ключами трех типов: строка, число и дата. И выведите значения этих ключей на экран.

```
&НаКлиенте
Процедура СоответствиеМетодВставить(Команда)
    Соответствие1 = Новый Соответствие;
    Соответствие1.Вставить("Ключ1","Ключ тип строка");
    Соответствие1.Вставить(1,"Ключ тип число");
    Соответствие1.Вставить(Дата(2017,12,10),"Ключ тип дата");
    Сообщить(Соответствие1["Ключ1"]);
    Сообщить(Соответствие1[1]);
    Сообщить(Соответствие1[Дата(2017,12,10)]);
КонецПроцедуры
```

Листинг 7.3.1

Сохраните и запустите обработку.

Вы видите, мы можем использовать в качестве ключа значение любого типа, что, повторюсь, в некоторых случаях бывает удобно. Для того чтобы поменять значение какого-нибудь элемента, который соответствует определенному ключу, достаточно для этого ключа присвоить новое значение.

Допишем предыдущий пример, чтобы понять, как это работает.

```
Соответствие1.Вставить(1,"Ключ тип ""число""");
Сообщить(Соответствие1[1]);
```

Листинг 7.3.2

Посмотрим, как выполнится наша команда.

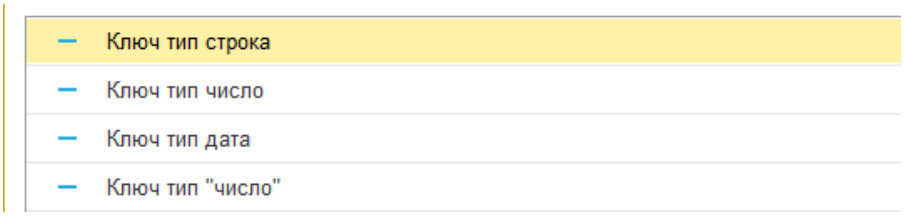


Рис. 7.3.1

Поменять имеющееся значение можно, также используя квадратные скобки. Код в листинге 7.3.2 можно переделать.

```
Соответствие1[1] = "Ключ тип ""число""";
Сообщить(Соответствие1[1]);
```

Листинг 7.3.3

Результат будет точно такой же.

Обход

Научимся обходить наше соответствие с помощью цикла *Для каждого…Цикл*.

Создадим новую команду, обработчик команды на клиенте, скопируем в этот обработчик код создания соответствия и выполним обход:

```
&НаКлиенте
Процедура СоответствиеОбход(Команда)
    Соответствие1 = Новый Соответствие;
    Соответствие1.Вставить("Ключ1","Ключ тип строка");
    Соответствие1.Вставить(1,"Ключ тип число");
    Соответствие1.Вставить(Дата(2017,12,10),"Ключ тип дата");

    Для Каждого ЭлементСоответствия из Соответствие1 цикл
        Сообщить("Соответствие1 [" +
                 Строка(ЭлементСоответствия.Ключ) + "] = " +
                 ЭлементСоответствия.Значение);
    КонецЦикла;

КонецПроцедуры
```

Листинг 7.3.4

Сохраним обработку, и посмотрите на результат.

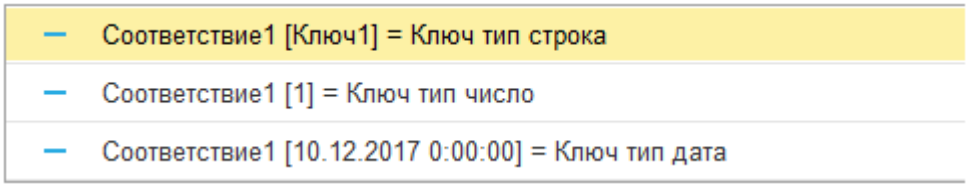


Рис. 7.3.2

Как видите, при обходе коллекции, мы получаем доступ к элементу коллекции, у данного элемента мы можем, как свойства объекта, получить и значение ключа, и само значение, которое соответствует данному ключу.

Очистка соответствия

Очистка соответствий осуществляется с помощью метода *Очистить*, который удаляет все элементы соответствия, и метода *Удалить*, параметром которого является ключ на конкретный элемент.

Синтаксис и работа методов похожи на синтаксис и работу методов структуры.

Соответствие1.Очистить();

Данный метод очищает полностью соответствие.

Синтаксис метода *Удалить* следующий:

Удалить(<Ключ>);

Рассмотрим пример с удалением элемента по ключу. Создайте новую команду, в этой команде создадим соответствие, в которое добавим несколько элементов с ключами типа дата. Потом удалим одну пару «КлючИЗначение».

```
&НаКлиенте
Процедура СоответствиеУдаление(Команда)
    Соответствие1 = Новый Соответствие;
    Соответствие1.Вставить(Дата(2017,12,1), "Число 2");
    Соответствие1.Вставить(Дата(2017,12,2), "Число 3");
    Соответствие1.Вставить(Дата(2017,12,3), "Число 1");
    Соответствие1.Вставить(Дата(2017,12,4), "Число 6");

    Соответствие1.Удалить(Дата(2017,12,3));

    Для Каждого ЭлементСоответствия из Соответствие1 цикл

        Сообщить("Соответствие1["+
                 Формат(ЭлементСоответствия.Ключ,"ДЛФ=DD") + "] = " +
                 ЭлементСоответствия.Значение);

    КонецЦикла;
КонецПроцедуры
```

Листинг 7.3.5

Проверьте работу этого кода самостоятельно.

Метод Получить

Как Вы уже поняли, для того чтобы получить значение элемента соответствия, мы используем квадратные скобки. Но иногда возможны ситуации, когда применение квадратных скобок неосуществимо. В таких случаях мы можем использовать метод *Получить*, который возвращает любое значение, соответствующее ключу. Если такого ключа нет, то вернется значение *Неопределено*.

Рассмотрим работу данного метода на уже известном нам соответствии датами в качестве ключей. Создадим новую команду, обработчик команды и скопируем в него код, где мы создаем соответствие, в котором в качестве ключей выступают даты. А потом с помощью метода *Получить* получим один из элементов этого соответствия.

```
&НаКлиенте
Процедура СоответствиеПолучить(Команда)
    Соответствие1 = Новый Соответствие;
    Соответствие1.Вставить(Дата(2017,12,1), "Число 2");
    Соответствие1.Вставить(Дата(2017,12,2), "Число 3");
    Соответствие1.Вставить(Дата(2017,12,3), "Число 1");
    Соответствие1.Вставить(Дата(2017,12,4), "Число 6");

    ЭлСоответствия1 = Соответствие1.Получить(Дата(2017,12,4));
    ЭлСоответствия2 = Соответствие1.Получить(Дата(2017,12,5));


Сообщить(?(ЗначениеЗаполнено(ЭлСоответствия1),ЭлСоответствия1,"Неопределено"));

Сообщить(?(ЗначениеЗаполнено(ЭлСоответствия2),ЭлСоответствия2,"Неопределено"));
КонецПроцедуры
```

Листинг 7.3.6

Посмотрим, как работает этот метод.

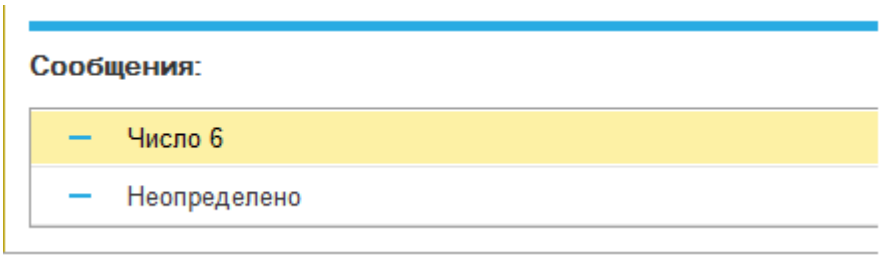


Рис. 7.3.3

Резюме

В этой части главы мы научились работать с соответствиями. Поначалу соответствия Вам чем-то будут напоминать структуры, но это абсолютно разные коллекции значений, и мы в этом убедились. В своих прикладных задачах Вы можете использовать как соответствия, так и структуры, в зависимости от тех целей, которые перед Вами стоят.

Часть 4. Список Значений

Список значений - по сути, это одномерный динамический массив, созданный в рамках платформы 1С для решения некоторых интерфейсных задач. В отличие от простого массива, элементы которого могут быть произвольных типов, у списка значений каждый элемент имеет специальный тип *Элемент списка значений*. Элемент списка значений может хранить в себе следующие данные: само значение (любого типа), представление значения (тип строка), пометка (тип булево) и картинка (тип картинка).

Список значений можно создать на всех видах клиентов и в любом контексте. Сериализируется, т.е. можно передать переменную с типом структура с клиентского контекста на серверный.

Хоть и в значение элемента списка значений можно записать переменные любого типа, использование типов в этом случае очень сильно зависит от контекста и от вида клиента: мы не можем в клиентском задать в значение тип, который работает только в серверном контексте (например, ДокументОбъект.<>).

В основном списки значений используются в интерфейсных задачах, это, как правило, организация выборки какого-нибудь одного объекта из списка возможных. В основном для решения таких задач используется элемент формы *Таблица*.

Таблицы формы и работу с таблицами будем изучать в следующей главе.

Приведем общий пример, чтобы Вам было понятно, что такое список значений и для чего: создайте новую обработку, форму обработки, а на форме создайте реквизит с типом «СписокЗначений» (см. рис. 7.4.1).

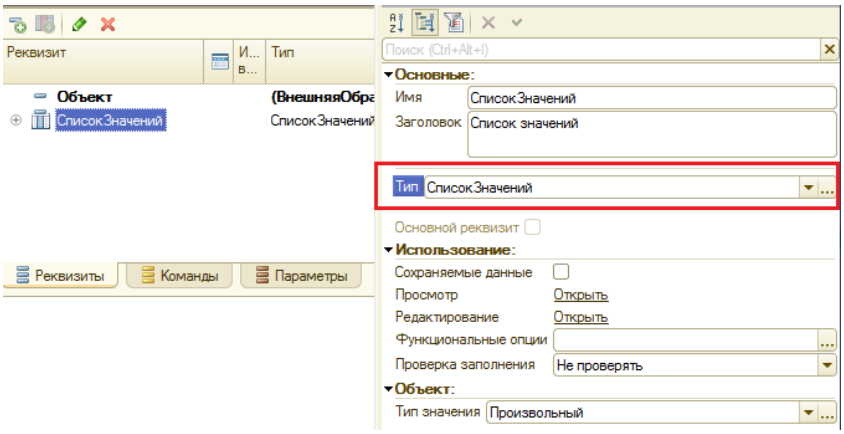


Рис. 7.4.1

Хоть и каждый элемент списка значений содержит значения типа *Элемент списка значений*, но можно задать тип данных, которые будут содержаться в этом элементе. Мы зададим следующий тип: «СправочникСсылка.МаркиАвтомилей». Тип данных в списке значений задается в свойстве реквизита *Тип значения*.

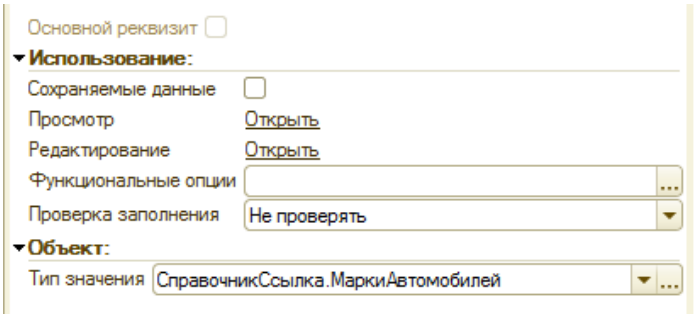


Рис. 7.4.2

Поместим этот реквизит на форму в виде таблицы:

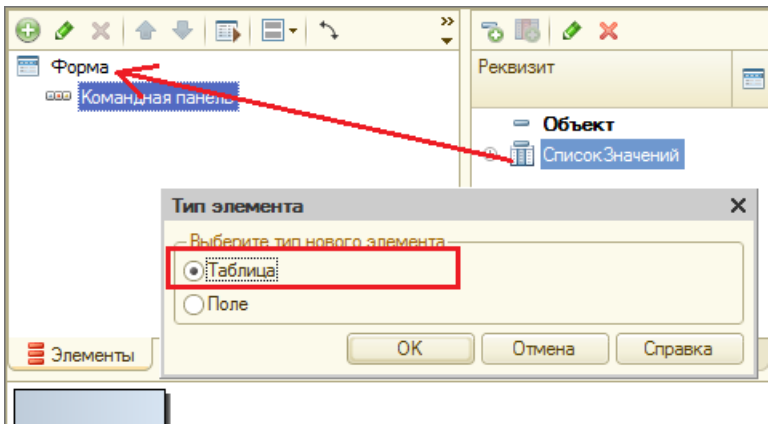


Рис. 7.4.3

На вопрос, добавлять колонки или нет, отвечаем – нет. Мы их добавим самостоятельно. После того, как Вы «перетащите» список значений на форму, на самой форме ничего не появится. Так произошло потому, что мы не добавили колонки. «Перетащим» самостоятельно две колонки: «Значение» и «Пометка». Нужно «тащить» колонки именно в сам элемент «СписокЗначений» (см. рис. 7.4.4 и 7.4.5).

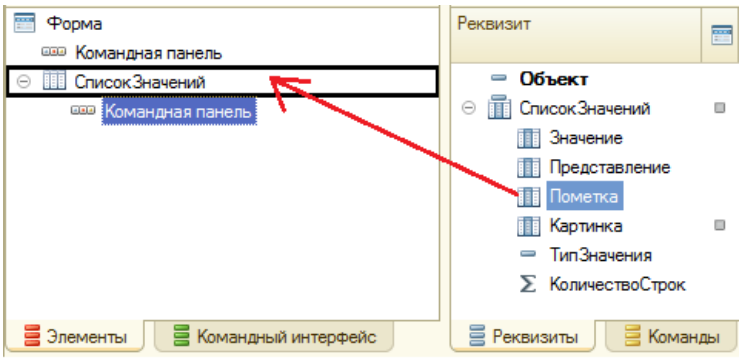


Рис. 7.4.4

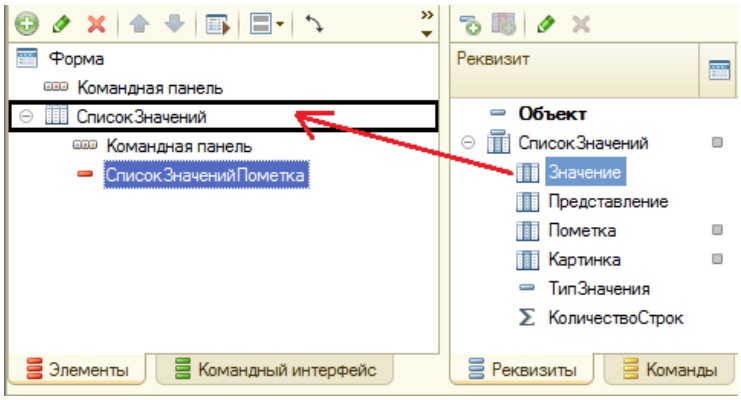


Рис. 7.4.5

Напишем в обработчике формы *ПриСозданииНаСервере* следующий код:

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

    Выборка = Справочники.МаркиАвтомобилей.Выбрать();
    Пока Выборка.Следующий() Цикл
        СписокЗначений.Добавить(Выборка.Ссылка,Выборка.Наименование);
    КонецЦикла;

КонецПроцедуры
```

Листинг 7.4.1

Сохраним и запустим обработку.

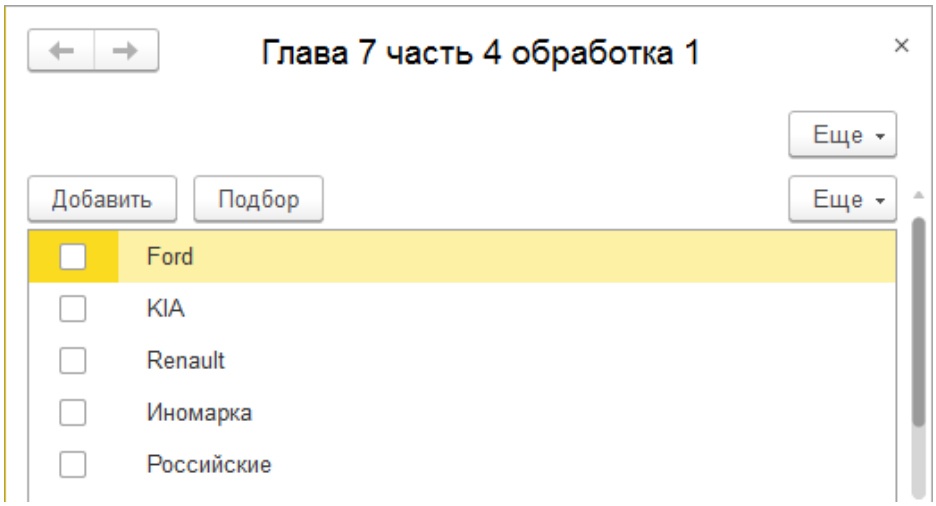


Рис. 7.4.6

Как видите, в поле списка вышли все элементы справочника *Марки автомобилей*.

Для этого мы использовали выборку из справочника (более подробно о них мы будем проходить в восьмой главе) и реквизит с типом *Список значений*, который заполнили с помощью метода *Добавить* элементами со значением *ссылка на элемент справочника* и представлением в виде наименования элемента.

Данный пример наглядно показывает, в каких целях можно использовать список значений.

Но все по порядку. Сначала мы научимся манипулировать списком значения: добавлять, вставлять, удалять, редактировать и сортировать элементы, а потом мы научимся работать со списком значений на форме.

Добавление элементов

Для того чтобы нам начать работать со списком значений, необходимо заполнить его элементами.

Сделать это можно тремя способами: добавить новый элемент (добавится в конец списка), вставить новый элемент в нужное место и загрузить элементы из массива.

Рассмотрим каждый способ по отдельности.

Метод «Добавить»

Метод *Добавить* создает новый элемент списка значений и добавляет его в конец списка.

Рассмотрим синтаксис данного метода.

Добавить(<Значение>, <Представление>, <Пометка>, <Картинка>)

Первый параметр *Значение* - это может быть объект любого типа, который будет храниться в списке значений.

Второй параметр - это непосредственно то, как данный объект будет представлен пользователю в списке. Имеет тип *строка*. Это необязательный параметр, в случае отсутствия представления объект будет показан в обычном виде.

Третий параметр задает, будет ли у данного объекта в списке пометка выбора или нет. Имеет тип *Булево*. Тоже необязательный параметр.

Четвертый параметр - это картинка, мы не будем разбирать картинки в данной книге, поэтому пропустим его, он тоже является необязательным.

Как Вы поняли из описания параметров, обязательным является только значение.

Рассмотрим пример: сделаем новую обработку, форму у этой обработки, команду на форме, и в обработчике команды создадим список значений, в который добавим элементы типа «Число». И укажем представления этих чисел.

```
&НаКлиенте
Процедура СписокЗначенийДобавить(Команда)
    Список = Новый СписокЗначений;
    Список.Добавить(100 , "Число 100");
```

```
    Список.Добавить(0.25, "Число: 1/4");
    Список.Добавить(-100, "Число -100");
    Список.Добавить(0.5 , "Число 1/2");
КонецПроцедуры
```

Листинг 7.4.2

Сохраним обработку и посмотрим в отладке, как записался наш список.

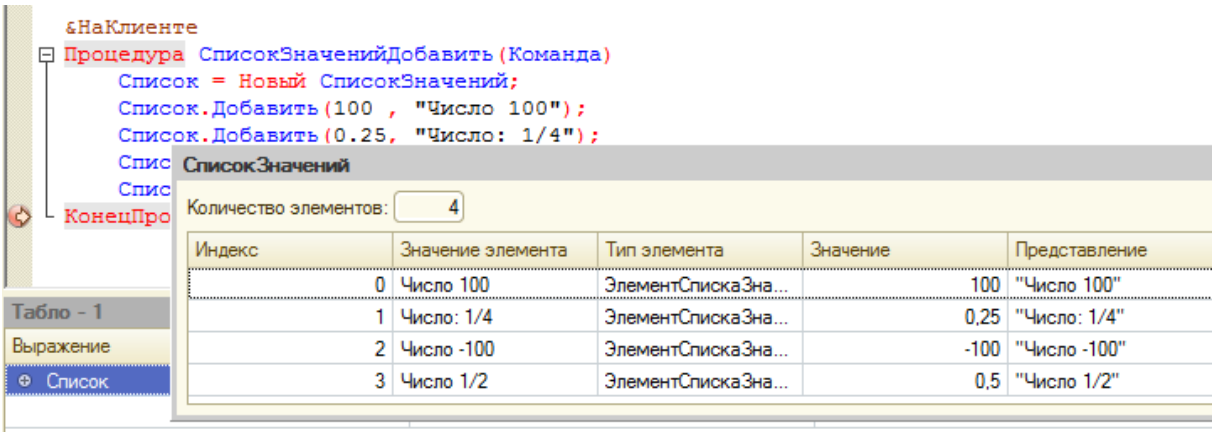


Рис. 7.4.7

Как видно из табло, значение и представление отличаются друг от друга. Но при этом, какое бы ни было представление, оно необходимо только для удобства восприятия пользователя. Сами данные хранятся непосредственно в значении. Продемонстрируем это: по аналогии с предыдущей обработкой на форме создадим реквизит *СписокНаФорме* с типом *СписокЗначения* (свойство *ТипЗначения* - число), поместим его на форму в виде таблицы и поместим в эту таблицу колонки «Значение» и «Представление» (см. рис. 7.4.8).

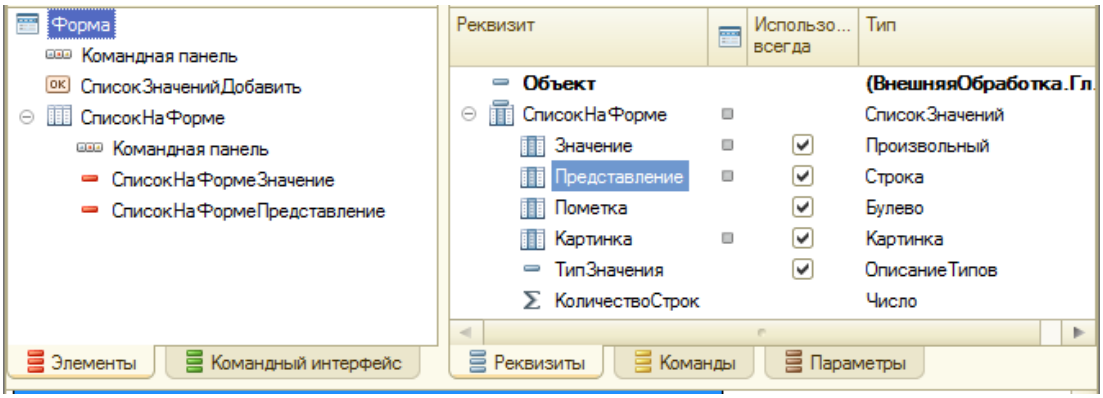


Рис. 7.4.8

А в процедуре обработчике команды этому реквизиту присвоим вновь созданный объект *Список значений*.

```
    СписокНаФорме = Список;
```

Перезапустите обработку, выполните команду и посмотрите, что выйдет в поле списка.

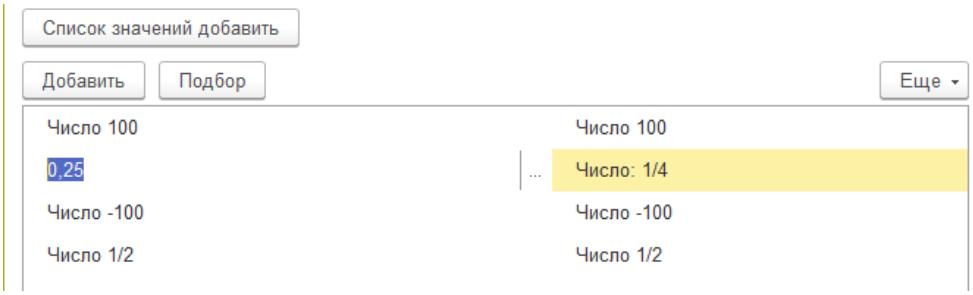


Рис. 7.4.9

Заметьте, в колонке значение данные отображаются в виде представлений, но если непосредственно зайти в ячейку для нужного значения, то увидим то значение, которое храниться в списке.

Рассмотрим следующий способ добавления новых элементов в список значение - с помощью метода *Вставить.*

Метод «Вставить»

Метод *Вставить* имеет следующий синтаксис.

Вставить(<Индекс>, <Значение>, <Представление>, <Пометка>, <Картинка>)

Все параметры, кроме *Индекса,* точно такие же, как у метода *Добавить*.

Параметр *Индекс* указывает непосредственно номер позиции, на который устанавливается элемент. Элемент, который был на этом индексе, сдвигается на плюс один, и все последующие элементы тоже.

Продемонстрируем это, вставим в уже созданный нами список в обработчике нажатия кнопки *«Выполнить»* новый элемент на второй индекс или на третье место по порядку.

```
Список.Добавить(0.5 , "Число 1/2");
Список.Вставить(2,777, "Число 777");//добавили в код
СписокНаФорме = Список;
```

Листинг 7.4.3

Сохраните обработку (я скрыл колонку *Представление*) и перезапустите.

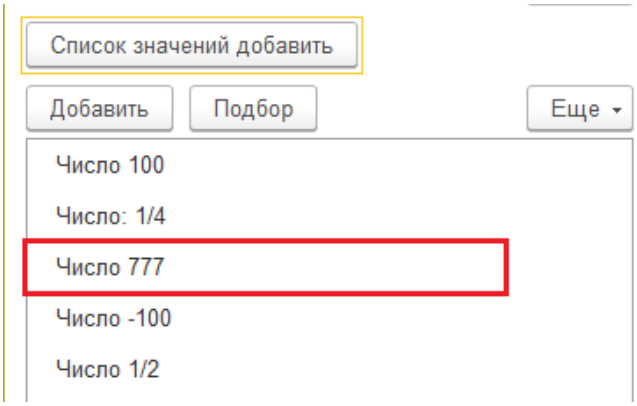


Рис. 7.4.10

Как видите, на третьем месте появился новый элемент. Напомню, что индексы в объектах 1С начинаются с нуля. Имейте всегда это в виду, когда будете делать обход по индексам или получать какое-нибудь значение по индексу.

Метод «ЗагрузитьЗначения»

Рассмотрим еще один метод, каким образом можно добавить элементы в список значений. Это метод *ЗагрузитьЗначения*. Данный метод позволяет загрузить значения из одномерного массива в список значений. Все предыдущие элементы списка значений при этом удалятся. Об этом надо всегда помнить.

Рассмотрим пример. В уже имеющейся обработке создайте новую команду *«Загрузить из массива»*, разместите её на форме в виде кнопки. А также создайте обработчик этой команды на клиенте.

В процедуре - обработчике команды напишем код, в котором будет заполняться массив некоторыми данными. А потом загрузим этот массив во вновь созданный список значений и выведем уже знакомым способом на форму в поле списка.

```
&НаКлиенте
Процедура ЗагрузитьИзМассива(Команда)

    Массив = Новый Массив;
    Массив.Добавить("Иванов И.И");
    Массив.Добавить("Петров А.Н");
    Массив.Добавить("Кузьмин А.Г.");
    Массив.Добавить("Сидоров Н.Н");

    Список = Новый СписокЗначений;
    Список.ЗагрузитьЗначения(Массив);
    СписокНаФорме = Список;

КонецПроцедуры
```

Листинг 7.4.4

Сохраните и перезапустите обработку, нажмите на вновь созданную кнопку *«Загрузить из массива»*. Должен выйти список как на рис. 7.4.11.

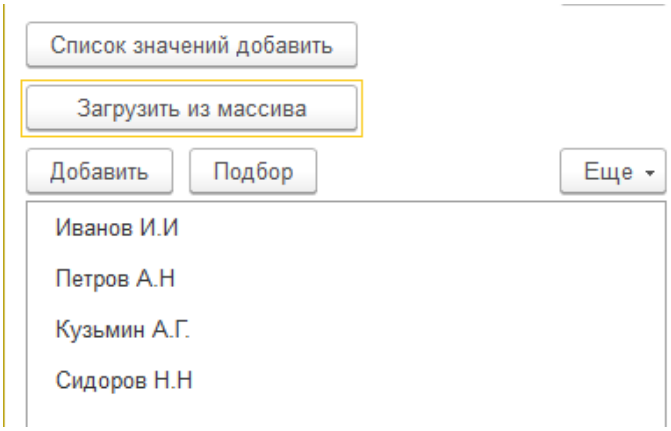


Рис. 7.4.11

Данный метод очень удобен, и его можно использовать, чтобы не писать лишний код по заполнению списка значений.

Имейте в виду, что этот метод заполняет только значения элементов списка, все остальные параметры (представление, картинка и т.п.) остаются пустыми.

Метод «ВыгрузитьЗначения»

Обратным методу *ЗагрузитьЗначения*, который заполняет список значений из массива, является метод *ВыгрузитьЗначения*. Данный метод является функцией и возвращает одномерный массив, формируя его значениями из списка.

Поэкспериментируйте с этим методом самостоятельно.

Элемент списка значений

После того, как мы научились добавлять, вставлять и загружать элементы в список значений, рассмотрим, что вообще такое *Элемент списка значений*.

Мы уже знаем, что элементы массива это и есть те значения, которые хранятся в массиве, мы знаем, что элементы структуры и соответствия это есть пара *Ключ и значение*. А что такое *Элемент списка значений*?

По сути, *Элементом списка значений* является отдельный объект, который имеет свойства и методы. Все параметры метода *Добавить* списка значений являются свойствами элемента списка значений, который будет *создан* при помощи этого метода. Любой список значений состоит из объектов *Элемент списка значений*. Мы можем обратиться к любому элементу списка

значений и работать с ним самостоятельно как с объектом. Но объект *Элемент списка значений* не может существовать без своего родителя - *Списка значений*.

Обратиться непосредственно к элементу списка значений можно двумя способами: с помощью оператора квадратные скобки и с помощью метода *Получить*.

Обратимся к первому элементу списка значений с помощью оператора квадратные скобки (метод *Получить* предлагаю Вам рассмотреть самостоятельно), который мы заполнили из массива. Данный элемент будет идти под индексом *0*.

В обработчике команды *Загрузить из массива* доработаем наш код:

```
//...
ПервыйЭлемент = Список[0];
Сообщить("ПервыйЭлемент.Значение = " + ПервыйЭлемент.Значение);
Сообщить("ПервыйЭлемент.Представление = " + ПервыйЭлемент.Представление);

СписокНаФорме = Список;
```

Листинг 7.4.5

Перезапустим обработку и выполним команду *Загрузить из массива.*

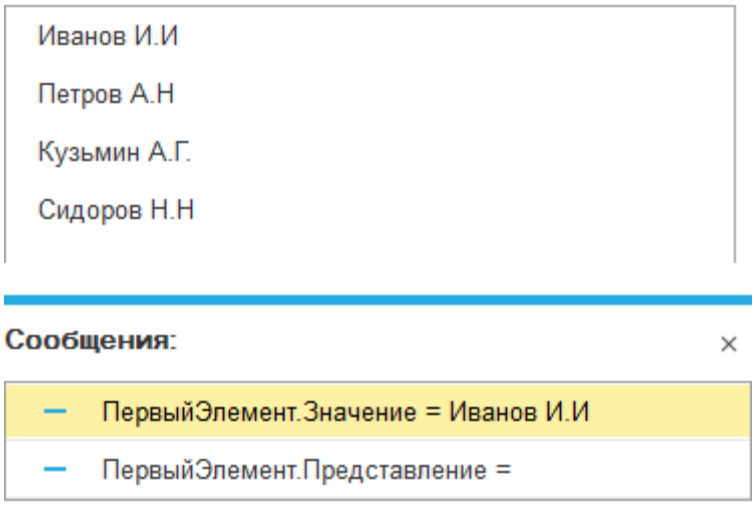


Рис. 7.4.12

Как видите, в первый элемент входит информация про *Иванова*, которую мы загрузили из массива. Представление не заполнено, поскольку при загрузке из массива заполняются только значения.

Теперь рассмотрим более внимательно код на листинге 7.4.5 .

Сначала мы получили элемент под индексом *0* в списке значений. Поскольку данный элемент является объектом, нам доступны его свойства. Их у элементов списка значений, как Вы уже поняли, четыре: *Значение*, *Представление*, *Пометка* и *Картинка*. Нас интересуют только первые два. Мы получаем эти свойства, как обычно, через точку.

Мы также можем самостоятельно изменять свойства элементов.

Доработайте Ваш пример, заполнив представление элемента.

```
//...
ПервыйЭлемент = Список[0];
ПервыйЭлемент.Представление = "Иванов Иван Иванович";
Сообщить("ПервыйЭлемент.Значение = " + ПервыйЭлемент.Значение);
Сообщить("ПервыйЭлемент.Представление = " + ПервыйЭлемент.Представление);

СписокНаФорме = Список;
```

Листинг 7.4.6

Сохраните и перезапустите обработку. Только что мы заполнили представление элемента и видим, что изменилось отображение первого элемента в списке.

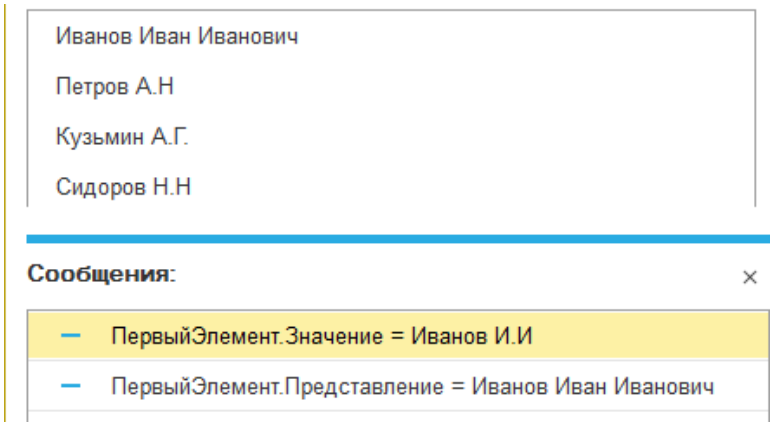


Рис. 7.4.13

Обращаю Ваше внимание, что отдельно (от списка значений) элемент списка значений как объект существовать не может! Он может существовать только в контексте данного списка значений.

Мы не можем его передавать из одного списка значений в другой.

Если уничтожился список значений, то уничтожился и элемент.

Обход коллекции

Обход коллекции списка значений можно выполнять двумя способами: с помощью операторов цикла *Для каждого…Цикл* или *Для…Цикл*.

Оба обхода равнозначны, но я рекомендую вам применять оператор *Для каждого…Цикл*, поскольку в таком случае Вы получаете сразу элемент коллекции. А в простом операторе его придется получать либо через квадратные скобки, либо через метод *Получить*.

Рассмотрим и тот, и другой способ обхода. Допишем код в обработчике команды «*Загрузить из массива*».

```
Для Каждого ЭлСписок из Список цикл
    Сообщить(ЭлСписок.Значение);
КонецЦикла;


Для н = 0 по Список.Количество() - 1 цикл
    Сообщить(Список[н].Значение);
КонецЦикла;
```

Листинг 7.4.7

Как видите, оба метода одинаковы, но, с другой стороны, способ с помощью цикла *Для…Цикл* является более быстрым, примерно процентов на 20. Поэтому при обходе больших коллекций значений всегда используйте цикл *Для…Цикл*.

Работа с элементом списка значений

Рассмотрим основные методы списка значений, предназначенные для работы с элементом.

И первый метод, который мы рассмотрим, будет *ВыбратьЭлемент*.

Метод «ВыбратьЭлемент»

Данный метод открывает окно для выбора определенного элемента из списка, и возвращает элемент, который выбрал пользователь, или *Неопределено*, если пользователь отказался от выбора.

Рассмотрим работу данного метода на примере: в последней обработке выберем любой элемент из списка *СписокНаФорме* и сообщим об этом.

Для этого создадим отдельную команду *Выбор элемента* и напишем в обработчике команды следующий код:

```
&НаКлиенте
Процедура ВыборЭлемента(Команда)

    ВыбрЭлемент = СписокНаФорме.ВыбратьЭлемент();
    Если ВыбрЭлемент <> Неопределено тогда
        Сообщить("Выбран " + ВыбрЭлемент.Значение);
    КонецЕсли;

КонецПроцедуры
```

Листинг 7.4.8

Сохраните, перезапустите обработку, заполните сначала список на форме, а потом нажмите кнопку *Выбор элемента.* Сразу откроется форма выбора элементов.

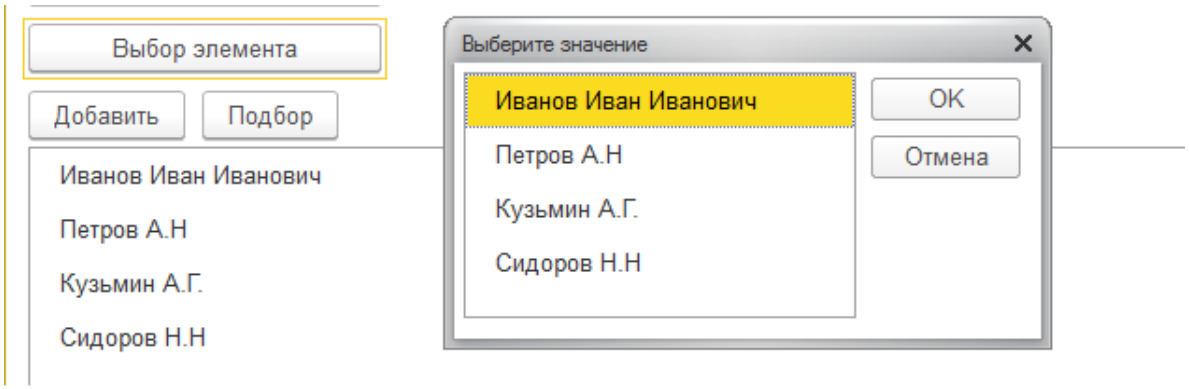


Рис. 7.4.14

Выберете любой из них двойным щелчком мышки, или просто указав на него и нажав кнопку *ОК*.

Выйдет соответствующее сообщение. Если мы ничего не выберем и нажмем кнопку *Отмена* или клавишу *Esc* клавиатуры, то никакое сообщение не выйдет – это значит, что метод вернул значение *Неопределено*.

Метод «Индекс»

Данный метод возвращает индекс передаваемого в него элемента.

Доработайте вышеприведенный пример: если выбран элемент, то сообщим его номер по порядку, а это номер индекса плюс один, если Вы помните.

```
&НаКлиенте
Процедура ВыборЭлемента(Команда)

    ВыбрЭлемент = СписокНаФорме.ВыбратьЭлемент();
    Если ВыбрЭлемент <> Неопределено тогда
        Номер = СписокНаФорме.Индекс(ВыбрЭлемент) + 1;
        Сообщить("Выбран " + ВыбрЭлемент.Значение + ", номер: " + Номер);
    КонецЕсли;

КонецПроцедуры
```

Листинг 7.4.9

Сохраните и перезапустите обработку.

Выберете какой-нибудь элемент, и Вы увидите, что в сообщении вышел номер по порядку данного элемента.

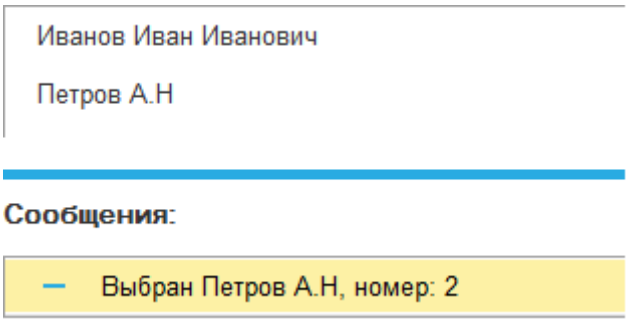


Рис. 7.4.15

Метод «Найти по значению»

У объекта *Список значений* есть один полезный метод, который позволяет искать элемент по его значению. Этот метод называется *НайтиПоЗначению*. Данный метод является функцией с одним параметром - это непосредственно значение, по которому ведется поиск элемента. Возвращает данная функция либо элемент списка значений, если он найден, либо значение *Неопределено*, в противном случае.

Покажем работу данного метода: для этого сделаем копию обработки, которую сделали в начале этой части (с марками автомобилей), и на форму копии поместим реквизит с типом значения *ссылка на справочник МаркиАвтомобилей* (см. рис. 7.4.16).

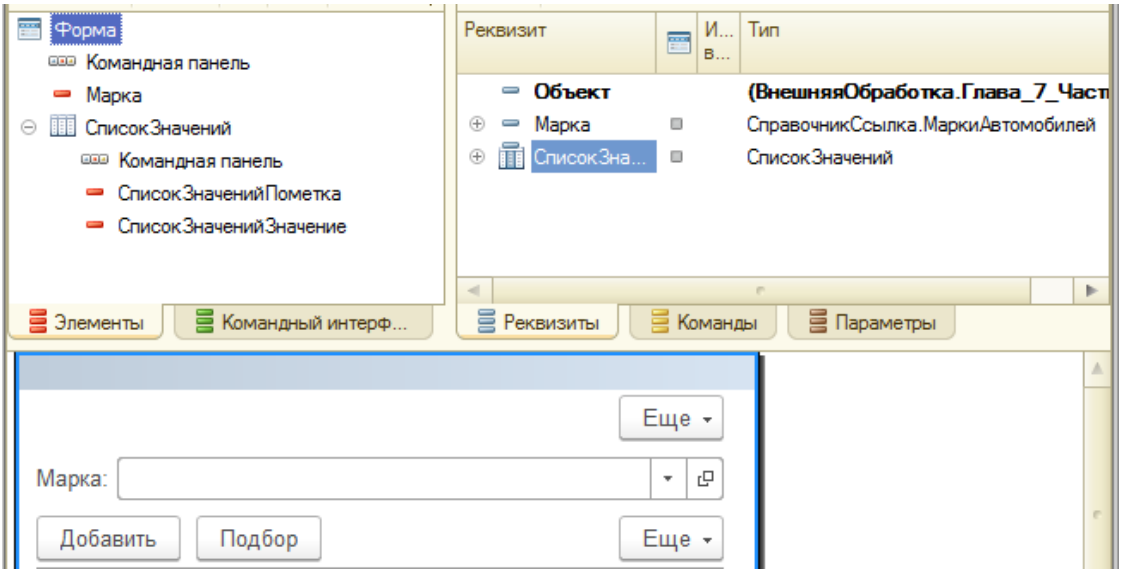


Рис. 7.4.16

При изменении значения в поле «Марка» будем искать в списке *СписокЗначений* это значение и отмечать его флажком. Перед этим у всех элементов мы уберем флажки с помощью метода *ЗаполнитьПометки*. Данная процедура имеет один параметр типа *булево*, и все флажки принимают значение этого параметра.

Создайте процедуру обработчик события *ПриИзменении* элемента форма «Марка», и в этой процедуре напишите следующий код:

```
&НаКлиенте
Процедура МаркаПриИзменении(Элемент)

    СписокЗначений.ЗаполнитьПометки(Ложь);
    ЭлНайденный = СписокЗначений.НайтиПоЗначению(Марка);
    Если ЭлНайденный <> Неопределено тогда
        ЭлНайденный.Пометка = Истина;
    КонецЕсли;

КонецПроцедуры
```

Листинг 7.4.10

Разберем данный код. В первой строке мы установили все пометки списка *Марка* в положение *Ложь*. В следующей строке мы ищем с помощью изучаемого метода элемент списка *Марка*. В параметр передается значение данных поля ввода *МаркаАвтомобиля.* И если элемент найден, то ставим его пометку равной *Истине*.

Сохраним обработку, запустите и посмотрите, как она работает.

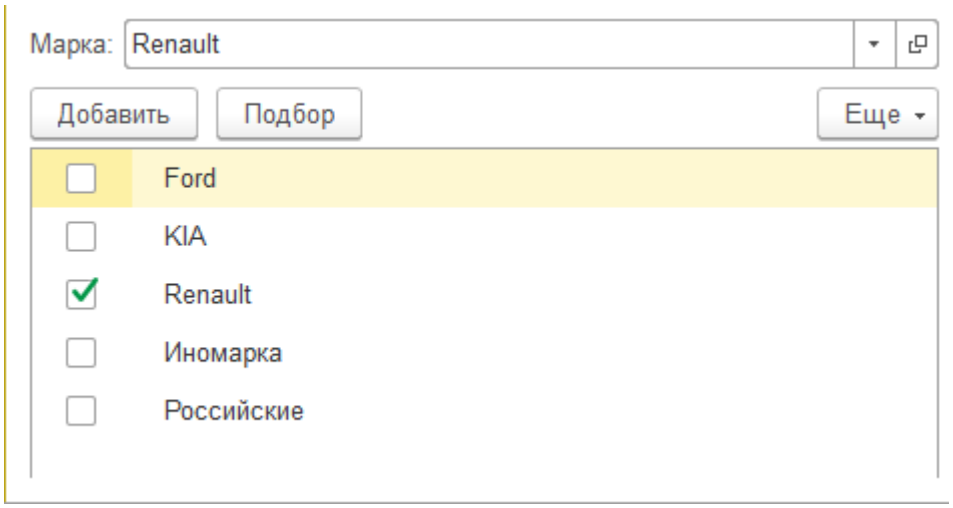


Рис. 7.4.17

Метод «Сдвинуть»

Рассмотрим последний из интересующих нас методов по работе с элементами - это метод *Сдвинуть*. Данный метод позволяет сместить элемент на определенное количество позиций.

Рассмотрим синтаксис этого метода.

Сдвинуть(<Элемент>, <Смещение>)

Параметр *Элемент* - это либо элемент списка значений, либо индекс данного элемента. *Смещение* – количество позиций, на которое сдвигается элемент. Если число положительное, то элемент сдвигается к концу списка, если отрицательное, то к началу списка.

Доработаем предыдущий пример: создадим новый реквизит на форме с типом значения *число*. И при выборе элемента будем сдвигать его на то число, которое в нем указано.

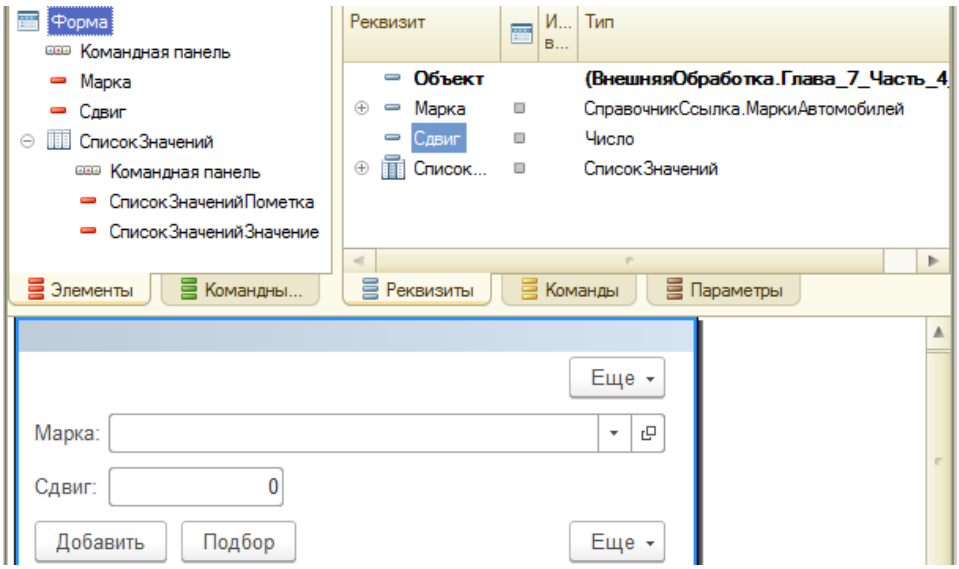


Рис. 7.4.18

Допишите в процедуре - обработчике события *ПриИзменении* поля ввода *Марка* код.

```
//...
ЭлНайденный.Пометка = Истина;
Если (СписокЗначений.Индекс(ЭлНайденный) + 1 + Сдвиг <=
СписокЗначений.Количество()) и
    (СписокЗначений.Индекс(ЭлНайденный) + 1 + Сдвиг > 0) тогда
    СписокЗначений.Сдвинуть(ЭлНайденный,Сдвиг);
КонецЕсли;
```

Листинг 7.4.11

В этом коде, в качестве условий, мы поставили ограничения, чтобы наш *Сдвиг* не выходил за границы списка значений. И сдвигаем найденный элемент.

Посмотрите, что получилось.

Сортировки

Разберем одну очень интересную возможность списка значений - это возможность сортировать элементы.

Для списков значений есть два метода, позволяющих отсортировать элементы внутри данного списка. Это *сортировка по значению*, когда сортируются непосредственно сами элементы в порядке возрастания или убывания. И второй метод - это *сортировка по представлению*, в этом случае элементы сортируются в алфавитном порядке представлений.

Первый способ сортировки лучше применять, когда в вашем списке находятся примитивные типы (строка, число и дата). В том случае, если в списке находятся элементы справочника, то сортировка по значению не всегда будет правильно выполняться, так как платформа будет сортировать все элементы по внутреннему идентификатору. Хотя очень часто внутренний идентификатор платформы может совпадать с названием элемента, и сортировка будет выполняться правильно.

Разберем пример: в обработке с предыдущего примера отсортируем список значения *СписокЗначения* по значению и по представлению. Для этого разместим на форме тумблер *Сортировка* с переключателями *По значению* и *По представлению* (реквизит и тумблер создайте самостоятельно, см. рис. 7.4.19). А при изменении этого тумблера будет сортировать *СписокЗначения* формы, для этого у поля *Сортировка* создадим обработчик команды *ПриИзменении*

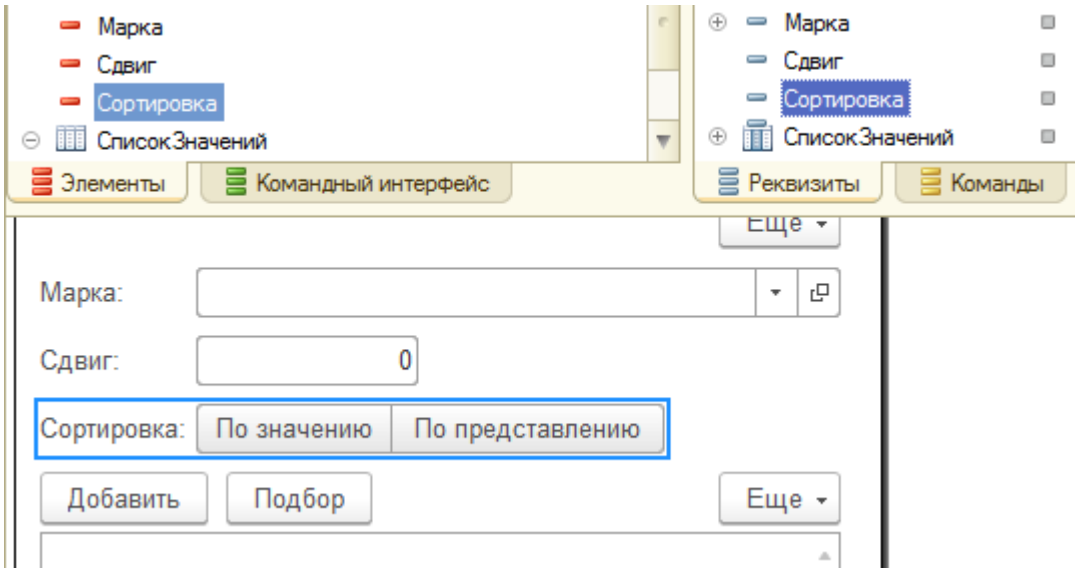


Рис. 7.4.19

В том случае, если нажат тумблер *По представлению*, элементы в списке будут сортироваться по представлению, а если *По значению* - по значению.

Для того, чтобы можно было наглядно убедиться, как работает данный метод, мы изменим представления списка *Марка* с марками автомобилей, так, чтобы последний в списке элемент начинался на *01*, предпоследний на *02* и т.д.

Допишите в обработчике события формы *ПриСозданииНаСервере* код следующим образом:

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)
    Выборка = Справочники.МаркиАвтомобилей.Выбрать();
    Пока Выборка.Следующий() Цикл
        СписокЗначений.Добавить(Выборка.Ссылка,Выборка.Наименование);
    КонецЦикла;

    ПоследнийИндекс = СписокЗначений.Количество() - 1;
    Счетчик = 1;
    Пока ПоследнийИндекс >= 0 цикл
        СписокЗначений[ПоследнийИндекс].Представление = Строка(Счетчик) + " "
+ Строка(СписокЗначений[ПоследнийИндекс].Значение);
        ПоследнийИндекс = ПоследнийИндекс - 1;
        Счетчик = Счетчик + 1;
    КонецЦикла;
КонецПроцедуры
```

Листинг 7.4.12

В этом коде мы изменяем представление списка *СписокЗначений* таким образом, чтобы был обратный порядок цифр. Сохраните и перезапустите обработку.

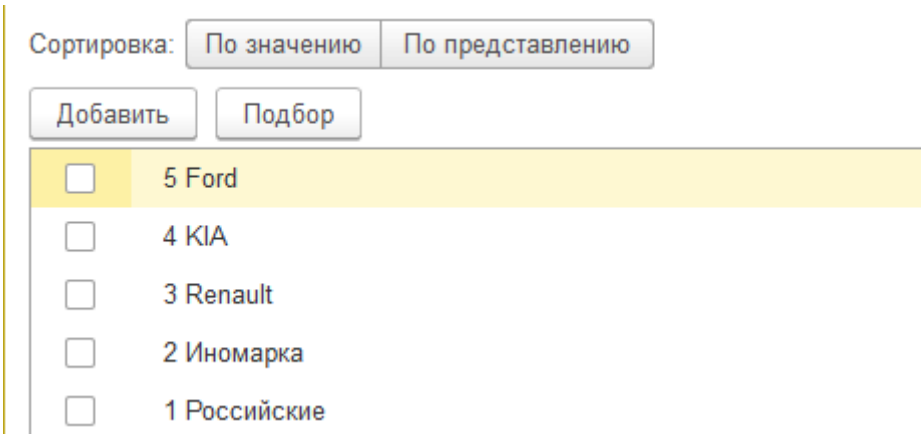


Рис. 7.4.20

Вы видите, что к названиям марок добавились цифры, которые идут в обратном порядке.

Теперь осуществим непосредственно саму сортировку. Внутри обработчика события *ПриИзменении* поля ввода «Сортировка» (тумблер) напишите следующий код:

```
&НаКлиенте
Процедура СортировкаПриИзменении(Элемент)
    Если Сортировка = 1 Тогда
        СписокЗначений.СортироватьПоЗначению();
    иначе
        СписокЗначений.СортироватьПоПредставлению();
    КонецЕсли;
КонецПроцедуры
```

Листинг 7.4.13

Сохраните, перезапустите обработку и посмотрите, как она работает.

Как Вы уже поняли, методы списков значений *СортироватьПоПредставлению* и *СортироватьПоЗначению* осуществляют сортировку элементов внутри списка.

У этих процедур есть параметр, это способ сортировки: либо по возрастанию, либо по убыванию. Данный параметр задается в виде системного перечисления *Направление сортировки*.

Покажем, как это все работает: добавьте новый тумблер на форму с двумя переключателями: *По убыванию* и *По возрастанию* (реквизит «ПорядокСортировки», см. рис. 7.4.21). Если нажат тумблер *По убыванию*, то элементы будут сортироваться по убыванию, в противном случае - по возрастанию.

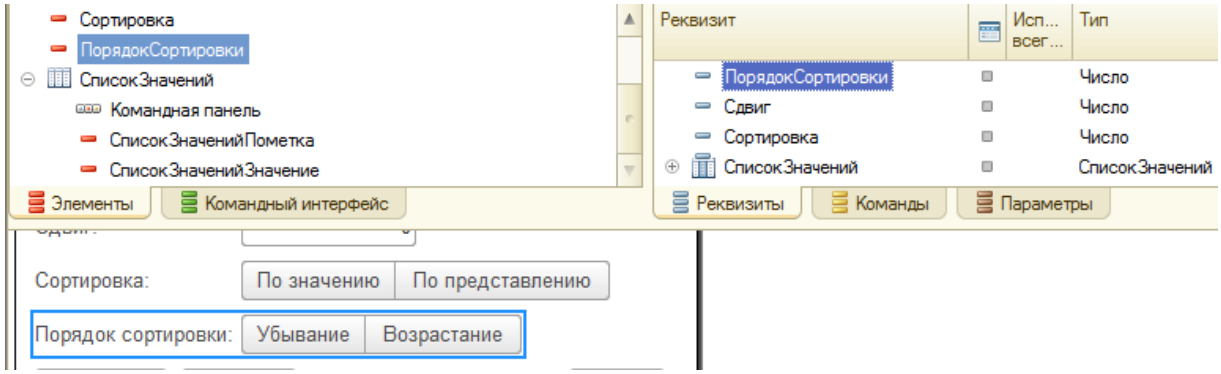


Рис. 7.4.21

Измените код в обработчике события кнопки *Сортировка*.

```
&НаКлиенте
Процедура СортировкаПриИзменении(Элемент)

    Если ПорядокСортировки = 1 Тогда
       Направление = НаправлениеСортировки.Убыв;
    ИначеЕсли ПорядокСортировки = 2 Тогда
       Направление = НаправлениеСортировки.Возр;
    КонецЕсли;

    Если Сортировка = 1 Тогда
         СписокЗначений.СортироватьПоЗначению(Направление);
    иначе
         СписокЗначений.СортироватьПоПредставлению(Направление);
    КонецЕсли;

КонецПроцедуры
```

Листинг 7.4.14

Сохраните, перезапустите обработку и посмотрите, как она работает.

Вот и все. Теперь мы с Вами научились сортировать различными способами списки значений, это совсем не сложно и требует очень небольших усилий.

Удаление элементов. Очистка

Напоследок рассмотрим такой важный вопрос, как удаление элементов и очистка списка значений в целом.

Очистка списка значений выполняется с помощью метода *Очистить*, который Вам уже знаком по предыдущим коллекциям, поэтому здесь его разбирать не будем.

Рассмотрим подробнее удаление элементов списка значений. Выполняется это с помощью метода *Удалить.*

Синтаксис метода следующий:

Удалить(<ЭлементСписка>)

Удалить(<Индекс>)

Данный метод содержит один параметр – *ЭлементСписка*. Это может быть либо число, которое будет являться индексом удаляемого элемента, либо конкретный элемент списка значений. Также можно указывать индекс элемента, который нужно удалить.

Посмотрим, как работает данный метод. Для этого разместите на форме кнопку *Удалить*, при нажатии на которую будет удаляться тот элемент списка на форме, который в данный момент выделен.

Создадим команду формы «Удалить», которую поместим на командную панель списка значений. К команде привяжем стандартную картинку «Удалить».

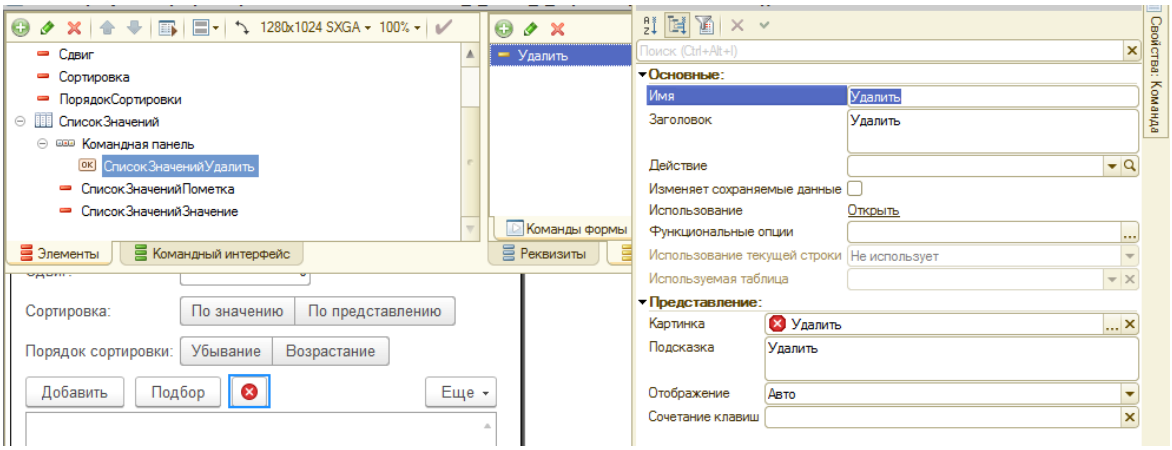


Рис. 7.4.22

Теперь в процедуре обработчике команды напишем код, в результате действия которого будет удаляться текущая строка (та, которая выделена курсором). Для этого будем использовать свойство *ТекущаяСтрока* элемента формы *Таблица*. Данное свойство возвращает индекс строки, на которой в данный момент установлен курсор, т.е. он выделен.

```
&НаКлиенте
Процедура Удалить(Команда)
    ИндексДляУдаления = Элементы.СписокЗначений.ТекущаяСтрока;

    Если ИндексДляУдаления <> Неопределено тогда
        СписокЗначений.Удалить(ИндексДляУдаления);
    КонецЕсли;
КонецПроцедуры
```

Листинг 7.4.15

Разберем этот код.

В первой строке мы получаем индекс текущего элемента, тот, который выделен в данный момент, с помощью свойства *ТекущаяСтрока* элемента формы *Таблица*.

Проверяем, действительно ли элемент выделен, потому что если элемент не выделен, то свойство вернет значение *Неопределено*.

И если переменная *ИндексДляУдаления* не равна значению *Неопределено*, то удаляем ее с помощью метода *Удалить*.

Сохраните обработку и посмотрите, как будет работать данная кнопка.

Резюме

Итак, в этой части мы научились работать со списками значений. Вы поняли, что использовать список значений гораздо удобнее, чем массив, поскольку данный объект обладает рядом несравненных преимуществ перед массивами. К тому же, есть специальный элемент формы, предназначенный для вывода данного объекта. Хотя в следующей главе мы с Вами будем рассматривать таблицы значений, которые более удобны и универсальны в работе с данными, все же знать принципы работы со списками необходимо.

Глава 8. Таблицы

В этой главе Вы научитесь работать с таблицами в 1С. При разработке в основном используется два вида таблиц: это таблицы значений, которые являются объектами универсальной коллекции значений, и непосредственно табличные части документов и справочников. В 8-й главе Вы научитесь работать с таблицами значений и с табличными частями документов и справочников.

Часть 1. Таблицы значений

Прежде всего определим, что такое *Таблица значений*.

Таблица значений представляет собой динамический набор значений, имеющих колонки и столбцы. Информация в таблице значений не хранится в базе данных и доступна только при работе контекста, в котором она была создана. Пользователи могут получать доступ к данным, хранящимся в таблице значений, посредством элемента формы *Таблица.*

Как мы уже определили, вся информация в таблице значений разбита на колонки, причем каждая колонка таблицы должна иметь свое уникальное название.

В том случае, если таблица выводится на форму посредством элемента *Таблица*, то у данного элемента могут быть все колонки, которые имеются у таблицы значений с соответствующими именами. Причем количество колонок в *Таблице* может быть меньше, чем непосредственно в таблице значений, то есть не все колонки из таблицы значений можно отображать в табличном поле. Нет никаких ограничений на количество колонок. Количество колонок, как правило, всегда фиксировано и задается разработчиком на этапе создания нужного функционала, а количество строк переменно. Строки в таблицу значений можно добавлять посредством *Таблицы* на форме интерактивно, а также используя методы объекта *Таблица значений,* программным способом. Количество строк может быть неограниченно.

Необязательно выводить таблицу значений на форму, программист может использовать ее и просто внутри модуля для осуществления различных алгоритмов.

У таблиц значений есть ряд полезных методов, которые позволяют сортировать таблицу по определенным колонкам, сворачивать, получать итог по конкретной колонке, осуществлять выборку части строк по определенным параметрам и многое другое. Все эти интересные вещи мы будем изучать в этой главе.

Обратите внимание, что таблица значений может существовать только либо в серверном контексте, либо в клиентском контексте, который выполняется ***под толстым клиентом****. Под тонким клиентом нельзя ни создать, ни использовать таблицу значений.*

В том случае, если Ваше приложение будет работать или под тонким клиентом, или под веб-клиентом, всегда работайте с таблицей значения в серверном контексте (под директивой

&НаСервере). Это утверждение справедливо для любого случая использования таблиц значений: и когда они созданы с помощью конструктора в модуле, и когда они размещены на форме в качестве реквизита.

Создать таблицу значений в программном модуле несложно, делается это с помощью оператора *Новый*, и она существует в рамках того контекста, в котором был написан конструктор.

Выглядит это так:

Таблица1 = Новый ТаблицаЗначений;

Колонки таблицы значений

Таблицу значений мы создали, но это только малая часть работы, поскольку сама по себе таблица значений нам абсолютно не нужна. Чтобы начать с ней работать, необходимо ее заполнить колонками.

Для этого мы будем использовать одно из основополагающих свойств таблиц значений - это свойство *Колонки*. Данное свойство представляет собой коллекцию значений, в которой хранятся все колонки конкретной таблицы.

Каждая колонка таблицы значений является отдельным объектом с типом *Колонка таблицы значений*, этот объект привязан к таблице значений и не может существовать без нее. У колонок нет методов, но зато существуют четыре свойства - это: *имя колонки*, *заголовок* (данный заголовок выводится в шапке *Таблицы*), *тип значения* (какие данные хранятся в колонке) и *ширина* (данное свойство содержит ширину колонки в символах).

Разберем, как мы будем создавать колонки.

Свойство *Колонки* таблицы значений является объектом *Коллекции колонок таблицы значений* (который привязан к этой таблице значений). У этой коллекции есть соответствующие методы, с помощью которых можно создавать и удалять элементы данной коллекции, которые являются колонками таблицы значений.

Коллекцию колонок, так же, как и любую другую коллекцию, можно обойти с помощью оператора цикла *Для каждого…Цикл*.

Итак, как же создавать сами колонки? У объекта *Коллекция колонок таблицы значений* существует два метода по созданию новых колонок. Это *Добавить* и *Вставить*.

Метод *Добавить* – добавляет новую колонку в конец коллекции, а метод *Вставить* – вставляет колонку в нужное место по соответствующему индексу.

Для примера, создадим простую таблицу значений, которую назовем *ФИО,* имеющую три колонки - *Фамилия, Имя* и *Отчество*. Создайте самостоятельно обработку и её форму, какую-нибудь команду. Создадим обработчики команды на сервере и на клиенте. Получится следующий код:

```
&НаСервере
Процедура СоздатьКолонкиТаблицыЗначенияНаСервере()
    ФИО = Новый ТаблицаЗначений;
    ФИО.Колонки.Добавить("Фамилия");
    ФИО.Колонки.Добавить("Имя");
    ФИО.Колонки.Добавить("Отчество");
КонецПроцедуры


&НаКлиенте
Процедура СоздатьКолонкиТаблицыЗначения(Команда)
    СоздатьКолонкиТаблицыЗначенияНаСервере();
КонецПроцедуры
```

Листинг 8.1.1

В данном простом примере Вы создали таблицу с тремя колонками. Теперь сохраните Вашу обработку и запустите ее в отладчике, чтобы посмотреть, что представляет собой коллекция колонок таблицы значений, о которой мы перед этим так долго говорили.

Выражение	Значение	Тип
⊖ ФИО	ТаблицаЗначений	ТаблицаЗначений
Индексы	ИндексыКоллекции	ИндексыКоллекции
⊖ Колонки	КоллекцияКолонокТаблицыЗначений	КоллекцияКолонокТаблицыЗначений
⊕ Имя	КолонкаТаблицыЗначений	КолонкаТаблицыЗначений
⊕ Отчество	КолонкаТаблицыЗначений	КолонкаТаблицыЗначений
⊕ Фамилия	КолонкаТаблицыЗначений	КолонкаТаблицыЗначений

Рис. 8.1.1

В данном табло Вы видите Вашу вновь созданную таблицу значений, у этой таблицы всего три колонки, которые мы создали. Раскроем одну из них.

⊖ Имя	КолонкаТаблицыЗначений	КолонкаТаблицыЗначений
Заголовок	""	Строка
Имя	"Имя"	Строка
⊕ ТипЗначения		ОписаниеТипов
Ширина	0	Число

Рис. 8.1.2

У этой колонки всего четыре свойства: *Заголовок, Имя, Тип Значения* и *Ширина*. Заполнено только одно свойство - *Имя*. Так произошло потому, что мы использовали только один параметр в методе *Добавить*.

Настало время разобрать синтаксис метода *Добавить* коллекции колонок таблицы значений.

Добавить(<Имя>, <Тип>, <Заголовок>, <Ширина>)

Как видно, у данного метода всего четыре параметра, по аналогии с четырьмя свойствами объекта *Колонка таблицы значений*. Параметры *Имя, Заголовок* и *Ширина* Вам понятны, тип параметров *Имя* и *Заголовок* это строка, а тип параметра *Ширина* – число. А тип второго параметра называется *Описание типов*, его мы еще не проходили, и поэтому сейчас разберем.

Описание типов

Как Вы должны помнить из главы, посвященной конфигурированию, когда мы задаем реквизит некоторого объекта (будь то справочник, документ, обработка или отчет), мы задаем тип для данного реквизита. То же самое справедливо и для реквизитов формы. Причем обращаю внимание, что один реквизит может принимать значения разных типов, если поставить флажок *Составной.*

Так мы задаем тип реквизита на этапе конфигурирования.

Но создавая таблицу значений в программном модуле, мы не можем задать тип той или иной колонки интерактивно как для элементов конфигурации, нам необходимо его задать программно. Делается это с помощью переменных, имеющих тип *Описание типов*. Переменная этого типа содержит все типы, значения которых может принимать соответствующая колонка таблицы значений.

Рассмотрим синтаксис конструктора данного типа:

Новый ОписаниеТипов(<Типы>, <КвалификаторыЧисла>, <КвалификаторыСтроки>, <КвалификаторыДаты>, <КвалификаторыДвоичныхДанных>)

Разберем данный конструктор. Как Вы уже поняли из конструктора, переменные данного типа задаются с помощью оператора *Новый*.

Первый параметр *Типы* - обязательный, и должен быть представлен либо в виде массива, элементы которого являются значениями типа *Тип*, либо это строка, в которой все типы перечислены через запятую.

Квалификаторы числа, строки, даты и *двоичных данных* - это объекты, которые задают параметры для соответствующих типов, это может быть: длина строки, длина и точность числа, и части даты.

Создадим описание типа, которое будет содержать в себе тип число, дата и строка. Причем тип будет задаваться двумя способами: с помощью массива и с помощью строки. Сделаем это в той же процедуре, где создавали таблицу значений, только выше.

```
&НаСервере
Процедура СоздатьКолонкиТаблицыЗначенияНаСервере()

    КвЧисл = Новый КвалификаторыЧисла(10,2);
    КвСтр  = Новый КвалификаторыСтроки(100);
    КвДаты = Новый КвалификаторыДаты(ЧастиДаты.ДатаВремя);

    МассивТипов = Новый Массив(3);
    МассивТипов[0] = Тип("Строка");
    МассивТипов[1] = Тип("Число");
    МассивТипов[2] = Тип("Дата");

    ОписаниеТипов1 = Новый ОписаниеТипов(МассивТипов,КвЧисл,КвСтр,КвДаты);
    //...
```

Листинг 8.1.2

В данном коде мы получили описание типов с помощью массива.

Рассмотрим вариант задания типов с помощью строки.

```
ОписаниеТипов2 = Новый
ОписаниеТипов("Строка,Число,Дата",КвЧисл,КвСтр,КвДаты);
```

Листинг 8.1.3

И вариант, когда типом будет ссылка на элемент справочника или на документ.

```
ОписаниеТипов3 = Новый ОписаниеТипов("СправочникСсылка.МаркиАвтомобилей");
ОписаниеТипов4 = Новый ОписаниеТипов("ДокументСсылка.ПрибытиеВГараж");
```

Листинг 8.1.4

Разберем вышеприведенный код. В листинге 8.1.2 в первых трех строках мы задаем квалификаторы числа, строк и даты. Это простые объекты, и мы не будем останавливаться на них подробно, поясню только, что задаются они с помощью оператора *Новый.* Квалификатор числа имеет два параметра, это: длина числа и точность. Квалификатор строки имеет также два параметра: длина строки и допустимая длина, а квалификатор даты имеет только один параметр – части даты.

Потом мы создаем массив из трех элементов и заполняем его переменными, имеющими тип *Тип* с помощью функции *Тип*.

И в последней строке листинга 8.1.2, а также в листингах 8.1.3, 8.1.4 создаем переменные с типом *Описание типов*. Причем если тип должен быть ссылочным, то необходимо писать либо *СправочникСсылка*, либо *ДокументСсылка*.

Если мы укажем переменную *ОписаниеТипов1* или *ОписаниеТипов2* при создании новой колонки таблицы, эта колонка сможет содержать в себе только три типа данных.

Пытливый читатель спросит: «А зачем задавать колонки и задавать в них переменные определенных типов, когда можно просто оставить данный параметр пустым, и значения в колонках будут любых типов?»

Да, так можно делать, но есть несколько «но». Во-первых, если Вы задаете колонки без типа, то данная таблица будет работать примерно в полтора раза медленнее, чем таблица с колонками, имеющими тип.

Во-вторых, если Вы будете передавать данную таблицу в запрос, то, в случае колонок без типа, данный запрос не будет работать.

Поэтому если таблица содержит небольшой объем данных и не передается в дальнейшем в запрос, то смело можете создавать нетипизированные колонки. В противном случае всегда задавайте тип колонок с помощью объекта *Описание типов*.

Итак, мы научились создавать описания типов, теперь сконструируйте таблицу со следующими колонками: *Фамилия* (тип *строка* с длиной 50), *Имя* (тип *строка*), *Отчество*, (тип *строка*) и *Дата рождения* (тип *дата*). Зададим все параметры метода *Добавить*.

Для этого создадим новую команду *«Создать таблицу значений с типами»* и обработчики к ней.

```
&НаСервере
Процедура СоздатьТЗСТипамиНаСервере()
    КвСтр  = Новый КвалификаторыСтроки(50);
    КвДаты = Новый КвалификаторыДаты(ЧастиДаты.Дата);

    ОписаниеТиповСтрока = Новый ОписаниеТипов("Строка",,КвСтр,);

    ФИО = Новый ТаблицаЗначений;
    ФИО.Колонки.Добавить("Фамилия",ОписаниеТиповСтрока,"Фамилия",50);
    ФИО.Колонки.Добавить("Имя",ОписаниеТиповСтрока,"Имя",50);
    ФИО.Колонки.Добавить("Отчество",ОписаниеТиповСтрока,"Отчество",50);
    ФИО.Колонки.Добавить("ДатаРождения",Новый
ОписаниеТипов("Дата",,,КвДаты),"Дата рождения",10);
КонецПроцедуры

&НаКлиенте
Процедура СоздатьТЗСТипами(Команда)
   СоздатьТЗСТипамиНаСервере();
КонецПроцедуры
```

Листинг 8.1.5

Разберем этот код. Первым делом мы создали квалификаторы строки и даты, для того чтобы потом их использовать в описании типов.

Следующим шагом создаем описание типа для строки. Вы заметили, что мы не стали создавать отдельную переменную, для описания типа *Дата*.

Потом мы создаем таблицу значений и добавляем колонки, используя все параметры метода *Добавить*. Надеюсь, Вы обратили внимание на последнюю строчку (листинг 8.1.5), в которой внутри параметра мы использовали ключевое слово *«Новый»*, так допустимо писать, если у Вас нет необходимости использовать созданное значение где-нибудь еще. Например, мы можем убрать переменные *КвСтр* и *КвДаты* и напрямую создавать новые квалификаторы.

Тогда наш код будет выглядеть так:

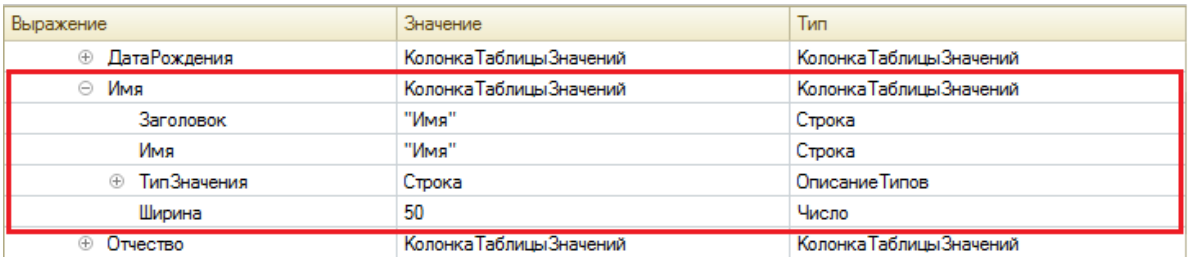
```
ОписаниеТиповСтрока = Новый ОписаниеТипов("Строка",,Новый
КвалификаторыСтроки(50),);

ФИО = Новый ТаблицаЗначений;
ФИО.Колонки.Добавить("Фамилия",ОписаниеТиповСтрока,"Фамилия",50);
ФИО.Колонки.Добавить("Имя",ОписаниеТиповСтрока,"Имя",50);
ФИО.Колонки.Добавить("Отчество",ОписаниеТиповСтрока,"Отчество",50);
ФИО.Колонки.Добавить("ДатаРождения",Новый ОписаниеТипов("Дата",,,Новый
КвалификаторыДаты(ЧастиДаты.Дата)),"Дата рождения",10);
```

Листинг 8.1.6

Как видите, использование оператора *Новый* внутри параметра вполне может оптимизировать код программы. Но не слишком увлекайтесь этим, поскольку при большом

количестве параметров внутри какой-нибудь процедуры или функции код будет трудно читаемым.

Посмотрим в отладке на колонку *Имя* нашей таблицы значений (см. рис. 8.1.3).



Выражение	Значение	Тип
ДатаРождения	КолонкаТаблицыЗначений	КолонкаТаблицыЗначений
Имя	КолонкаТаблицыЗначений	КолонкаТаблицыЗначений
Заголовок	"Имя"	Строка
Имя	"Имя"	Строка
ТипЗначения	Строка	ОписаниеТипов
Ширина	50	Число
Отчество	КолонкаТаблицыЗначений	КолонкаТаблицыЗначений

Рис. 8.1.3

Как видите, у колонки *Имя* заполнены все свойств.

Итак, мы научились добавлять новые колонки в таблицу значений, осуществляется это с помощью объекта *Коллекция колонок таблицы значений.* Поскольку данный объект является коллекцией, то каждая колонка имеет свой индекс. Вы можете осуществить обход коллекции с помощью оператора цикла *Для каждого…Цикл*, или получить доступ к свойствам колонки с помощью квадратных скобок.

Например, так:

ФИО.Колонки[0]

Обращаю Ваше внимание, что с помощью объекта *Колонки* Вы получите доступ только к свойствам колонки (название, заголовок и т.д.), а не к тем данным, которые хранятся в соответствующей колонке.

Метод «Вставить»

Сейчас, используя полученные ранее знания, разберем метод *Вставить* объекта *Коллекция колонок таблицы значений*. Данный метод имеет следующий синтаксис:

Вставить(<Индекс>, <Имя>, <Тип>, <Заголовок>, <Ширина>)

Метод *Вставить* имеет синтаксис, как и у метода *Добавить*, но только появился первый параметр *Индекс*, он указывает тот индекс, на место которого будет вставляться новая колонка.

Используем уже созданную нами таблицу, вставив колонку между *Отчеством* и *Датой рождения*, которая будет называться *ФИО*, где будет храниться фамилия с инициалами.

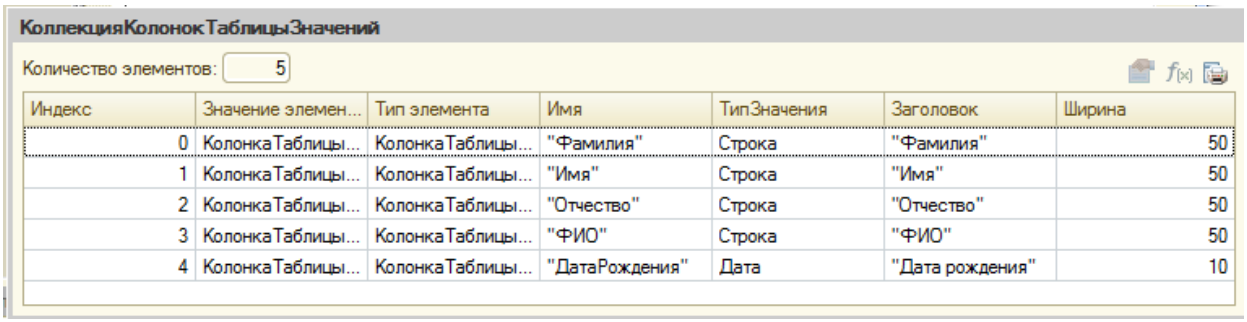
Доработаем код из листинга 8.1.6.

```
ФИО = Новый ТаблицаЗначений;
ФИО.Колонки.Добавить("Фамилия",ОписаниеТиповСтрока,"Фамилия",50);
ФИО.Колонки.Добавить("Имя",ОписаниеТиповСтрока,"Имя",50);
ФИО.Колонки.Добавить("Отчество",ОписаниеТиповСтрока,"Отчество",50);
ФИО.Колонки.Добавить("ДатаРождения",Новый ОписаниеТипов("Дата",,,Новый
КвалификаторыДаты(ЧастиДаты.Дата)),"Дата рождения",10);
```

```
//вставляем колонку
ФИО.Колонки.Вставить(3,"ФИО",ОписаниеТиповСтрока,"ФИО",50);
```

Листинг 8.1.7

Для того чтобы посмотреть, в какое место вставилась Ваша колонка, поставьте точку останова в конце процедуры и запустите отладку (см. рис. 8.1.4).


КоллекцияКолонокТаблицыЗначений

Количество элементов: 5

Индекс	Значение элемен...	Тип элемента	Имя	ТипЗначения	Заголовок	Ширина
0	КолонкаТаблицы...	КолонкаТаблицы...	"Фамилия"	Строка	"Фамилия"	50
1	КолонкаТаблицы...	КолонкаТаблицы...	"Имя"	Строка	"Имя"	50
2	КолонкаТаблицы...	КолонкаТаблицы...	"Отчество"	Строка	"Отчество"	50
3	КолонкаТаблицы...	КолонкаТаблицы...	"ФИО"	Строка	"ФИО"	50
4	КолонкаТаблицы...	КолонкаТаблицы...	"ДатаРождения"	Дата	"Дата рождения"	10

Рис. 8.1.4

Как видите, колонка *ФИО* встала точно между *Отчеством* и *Датой рождения*.

На этом мы закончим изучать работу с колонками таблицы значений, так как двух данных методов Вам будет вполне достаточно, чтобы успешно работать с таблицами значений. С остальными методами (они все похожи на методы, которые мы изучали для списка значений) Вы можете ознакомиться самостоятельно, работая с синтаксис-помощником. У Вас это не вызовет больших затруднений.

Строки таблицы значений

Только что мы научились создавать колонки таблицы значений. Как Вы поняли, на этапе создания колонок можно задать жестко, какие данные будут храниться в данной колонке. Это надо всегда помнить, поскольку если Вы попробуете записать значение типа, отличного от указанного, то значение не будет записано.

Теперь мы научимся заполнять таблицу программным способом: то есть создавать строки и записывать в них нужные данные.

Метод «Добавить»

Для того чтобы создать новую строку таблицы значений, используется метод *Добавить*, данный метод не имеет параметров и является функцией.

Новая строка таблицы значений создается следующим образом:

НоваяСтрока = ФИО.Добавить();

Как видите, с помощью метода *Добавить* мы создали переменную *НоваяСтрока*, тип значения которой *Строка таблицы значений*. Но, просто создать строку мало, нам еще необходимо записать в нее определенные данные. Как получить доступ к колонкам данной строки?

Осуществить это можно двумя способами:

Первый способ:

НоваяСтрока.Фамилия = "Иванов";

Второй способ:

НоваяСтрока[1] = "Иван";

В первом способе мы получаем доступ к колонке как к свойству через точку, во втором - используя оператор *квадратные скобки* и указывая в них *номер индекса колонки*. Я предпочитаю первый способ, так как он делает код гораздо лучше читаемым и наглядным.

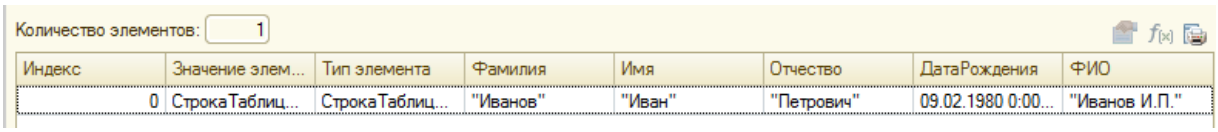
Продолжим работать с процедурой, где создали таблицу значений и заполнили колонками. Напишем в ней код, который создает новую строку таблицы значений и заполняет ее.

```
НоваяСтрока = ФИО.Добавить();
НоваяСтрока.Фамилия = "Иванов";
НоваяСтрока[1] = "Иван";
НоваяСтрока.Отчество = "Петрович";
НоваяСтрока.ДатаРождения = '19800209';
НоваяСтрока.ФИО = НоваяСтрока.Фамилия +" " +
                  Лев(НоваяСтрока.Имя,1) +"." +
                  Лев(НоваяСтрока.Отчество,1) + ".";
```

Листинг 8.1.8

Как Вы видите из кода, обращаясь к строке, мы можем как читать данные, так и записывать их.

А теперь посмотрите, что записалось в нашу таблицу значений. Для этого запустите опять отладку, поставьте точку останова. И откройте Вашу таблицу в табло (см. рис. 8.1.5).


Количество элементов: 1

Индекс	Значение элем...	Тип элемента	Фамилия	Имя	Отчество	ДатаРождения	ФИО
0	СтрокаТаблиц...	СтрокаТаблиц...	"Иванов"	"Иван"	"Петрович"	09.02.1980 0:00...	"Иванов И.П."

Рис. 8.1.5

Как Вы видите, в табло у данной таблицы значений уже есть одна строка, и она заполнена теми данными, которые мы только что в нее ввели.

Какие еще есть методы у строк таблицы значений? Всего их три: это метод *Владелец* – возвращает таблицу значений, которой принадлежит данная строка; метод *Установить*, который

записывает данные в нужную колонку строки; и метод *Получить*, который читает данные нужной колонки.

Методы *Установить* и *Получить* следует применять, когда Вы не можете использовать оператор *квадратные скобки* или обращаться напрямую через точку.

Разберем данные методы.

Метод *Установить* имеет следующий синтаксис

Установить(<Индекс>,<Значение>)

Первый параметр это индекс колонки, в которую будет записываться значение, второй параметр это непосредственно записываемое значение.

Добавим еще одну строку таблицы и занесем все данные в нее с помощью метода *Установить*.

```
НоваяСтрока = ФИО.Добавить();
НоваяСтрока.Установить(0,"Васильев");
НоваяСтрока.Установить(1,"Иван");
НоваяСтрока.Установить(2,"Андреевич");
НоваяСтрока.Установить(3,"Васильев И.А.");
НоваяСтрока.Установить(4,'19760910');
```

Листинг 8.1.9

Написали код, и теперь посмотрите, как записались данные в Вашу таблицу: поставьте точку останова, запустите отладку и посмотрите на Вашу таблицу в табло.

Количество элементов: 2

Индекс	Значение элем...	Тип элемента	Фамилия	Имя	Отчество	ДатаРождения	ФИО
0	СтрокаТаблиц...	СтрокаТаблиц...	"Иванов"	"Иван"	"Петрович"	09.02.1980 0:00...	"Иванов И.П."
1	СтрокаТаблиц...	СтрокаТаблиц...	"Васильев"	"Иван"	"Андреевич"	10.09.1976 0:00...	"Васильев И.А."

Рис. 8.1.6

Как Вы видите в табло, у нас появилась еще одна строка с данными, которые мы внесли с помощью метода *Установить*.

Теперь посмотрим, как с помощью метода *Получить* обратиться к значениям, которые хранятся в колонках: обойдем все колонки созданной только что строки и выведем в окно сообщений их данные.

```
Для н = 0 по ФИО.Колонки.Количество() - 1 цикл

    Сообщить(ФИО.Колонки[н].Заголовок + ": " +
              Строка(НоваяСтрока.Получить(н)));

КонецЦикла;
```

Листинг 8.1.10

Разберем данный код. По условиям задания необходимо обойти все колонки и получить значения, которые хранятся в них. В принципе, это можно было осуществить китайским методом, написав пять раз сообщение, но мы будем решать данную задачу используя цикл *Для…Цикл*. В данном цикле необходимо получить индексы всех колонок, для этого мы начинаем обход с нуля, и крайним значением будет количество колонок за минусом единицы.

В процедуре *Сообщить* сначала мы получаем заголовок, используя свойство *Заголовок* колонки таблицы значений, а потом с помощью метода *Получить* - значение, которое хранится в данной колонке.

Сохраните и перезапустите обработку. При нажатии кнопки *Выполнить* должны выйти данные, которые хранятся в последней созданной колонке.

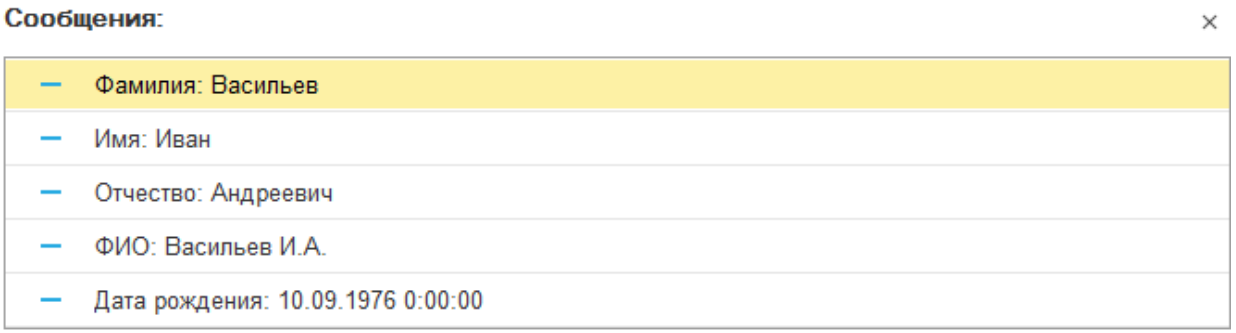


Рис. 8.1.7

Метод «Вставить»

С помощью метода *Добавить* мы создаем строку, которая вставляется в конец таблицы значений. Для того чтобы поместить строку в нужное место таблицы значений, необходимо использовать метод *Вставить*. Параметром данного метода является индекс, на который вставляется данная строка.

Добавим с помощью этого метода еще одну строку в нашу таблицу. И поставим ее на первое место.

```
НоваяСтрока = ФИО.Вставить(0);
НоваяСтрока.Фамилия = "Петров";
НоваяСтрока.Имя     = "Игорь";
НоваяСтрока.Отчество = "Андреевич";
НоваяСтрока.ФИО = НоваяСтрока.Фамилия +" " + Лев(НоваяСтрока.Имя,1) +"." +
Лев(НоваяСтрока.Отчество,1) + ".";
НоваяСтрока.ДатаРождения = '19850909';
```

Листинг 8.1.11

Поставьте точку останова в конце процедуры и посмотрите, в какое место таблицы встала данная строка.

Индекс	Значение элем...	Тип элемента	Фамилия	Имя	Отчество	ДатаРождения	ФИО
0	СтрокаТаблиц...	СтрокаТаблиц...	"Петров"	"Игорь"	"Андреевич"	09.09.1985 0:00...	"Петров И.А."
1	СтрокаТаблиц...	СтрокаТаблиц...	"Иванов"	"Иван"	"Петрович"	09.02.1980 0:00...	"Иванов И.П."
2	СтрокаТаблиц...	СтрокаТаблиц...	"Васильев"	"Иван"	"Андреевич"	10.09.1976 0:00...	"Васильев И.А."

Рис. 8.1.8

Как видите, новая строка встала на первое место таблицы значений.

Обход строк

Мы научились добавлять новые строки и читать в них значения. Теперь научимся обходить сроки таблицы. Это можно осуществлять с помощью операторов цикла *Для каждого...Цикл* и *Для…Цикл*. Осуществим обход уже заполненной таблицы и тем, и другим способом.

Начнем с оператора цикла *Для каждого...Цикл.*

```
Для Каждого Стр из ФИО цикл
    Сообщить(Стр.ФИО + ", дата рождения " +
             Формат(Стр.ДатаРождения,"ДЛФ = ДД"));
КонецЦикла;
```

Листинг 8.1.12

Как видите, с помощью этого способа мы обходим все строки таблицы значений. Переменная *Стр* принимает значения всех строк таблицы в порядке обхода. Тип данной переменной - *Строка таблицы значений*, поэтому мы можем спокойно обращаться к колонкам данной строки, используя точку или квадратные скобки.

Покажем, как осуществить обход, используя оператор *Для…Цикл*.

```
Для н = 0 по ФИО.Количество() - 1 цикл
    Сообщить(ФИО[н].ФИО + ", дата рождения " +
             Формат(ФИО[н].ДатаРождения,"ДЛФ = ДД"));
КонецЦикла;
```

Листинг 8.1.13

В этом цикле нам необходимо получить индексы всех строк, для этого мы осуществляем обход, начиная с нуля и заканчивая значением, которое возвращает метод *Количество* за минусом единицы.

Когда мы используем квадратные скобки применительно к таблице значений *(например, ФИО[н])*, то результатом данной операции является строка таблицы с соответствующим индексом. Это мы демонстрируем в только что приведенном примере.

Итак, мы научились обходить таблицы значений, используя два разных оператора цикла. Оба они дают одинаковые результаты. Но первый способ имеет преимущества: во-первых, код более читаемый, во-вторых, первый способ более быстрый по времени.

Поэтому рекомендую Вам для обхода таблиц значений использовать цикл *Для Каждого…Цикл*.

Поиск по таблице значений

Довольно часто ставится задача найти в таблице строку с каким-нибудь конкретным значением, или получить строки по определенному заранее заданному отбору. Осуществляется это с помощью методов *Найти* и *НайтиСтроки*.

Метод «Найти»

Метод *Найти* осуществляет поиск нужного значения в указанных колонках. Данный метод возвращает строку таблицы значений, если в ней присутствует искомое значение, или возвращает *Неопределено*, если значение не было найдено. Если имеется несколько строк с искомым значением, то будет возвращена только одна. Поэтому рекомендуется использовать данный метод для поиска *уникальных* значений.

Рассмотрим синтаксис функции *Найти*.

Найти(<Значение>, <Колонки>)

Первый параметр - непосредственно то значение, которое мы ищем, он является обязательным.

Второй параметр имеет тип значения *строка* и представляет собой имена колонок, разделенных запятыми. Он может быть не заполнен, тогда поиск осуществляется по всей таблице.

Рассмотрим пример: в уже созданной нами таблице *ФИО* будем искать нужную нам фамилию. Для этого на форме создадим реквизит «Фамилия» с типом «Строка» (длина 50) и разместим его на форме в виде поля ввода (см. рис. 8.1.9). И при выполнении команды *Создать таблицу значений с типами* будет осуществляться поиск значения из данного поля в колонке *Фамилия.* Если оно будет найдено, то выведем в окно сообщений соответствующую информацию.

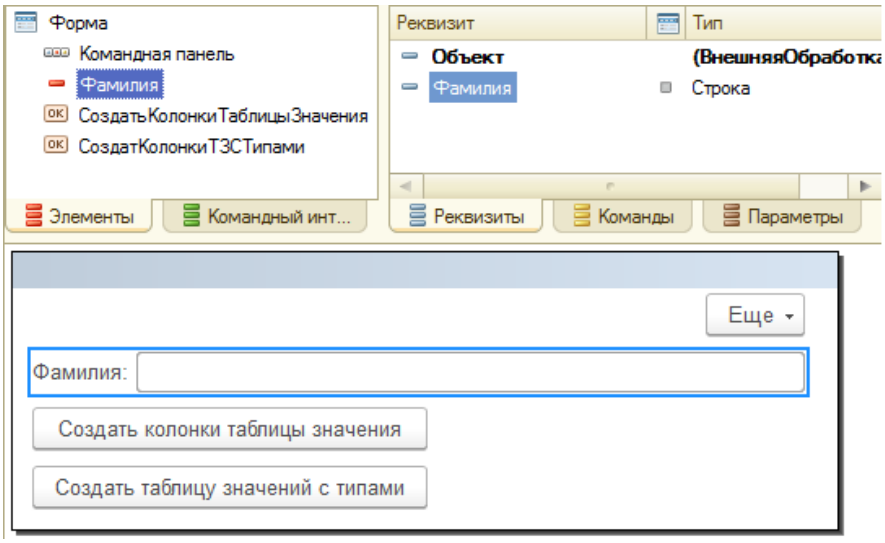


Рис. 8.1.9

Допишите в Вашей обработке следующий код:

```
Если Не ПустаяСтрока(Фамилия) тогда
    СтрокаСФамилией = ФИО.Найти(СокрЛП(Фамилия),"Фамилия");
    Если СтрокаСФамилией <> Неопределено тогда
        Для н = 0 по ФИО.Колонки.Количество() - 1 цикл
            Если ТипЗнч(СтрокаСФамилией[н]) = Тип("Дата") тогда
                Сообщить(ФИО.Колонки[н].Заголовок + ": " +
                         Формат(СтрокаСФамилией[н],"ДЛФ = ДД"));
            Иначе
                Сообщить(ФИО.Колонки[н].Заголовок + ": " +
                        Строка(СтрокаСФамилией[н]));
            КонецЕсли;
        КонецЦикла;
    иначе
        Сообщить("В таблице нет записей с фамилией " + Фамилия);
    КонецЕсли;
КонецЕсли;
```

Листинг 8.1.14

Разберем вышеприведенный код.

Первым делом мы проверяем, пустая у нас строка или нет, потому что иначе нет смысла искать. Для того чтобы найти строку с искомой фамилией, мы применяем метод *Найти*, в котором указываем непосредственно искомое значение и название колонки, по которой будет осуществляться поиск.

Метод *Найти* возвращает найденную строку или значение *Неопределено*, поэтому мы проверяем переменную на неравенство значению *Неопределено* и выводим сообщения о данных, которые хранятся в строке, обойдя все колонки по индексу. Конечно же, Вы обратили внимание, что мы проверяем тип данных в колонке, и если тип является типом *Дата*, то выводим ее в специальном формате.

Сохраните обработку и посмотрите, как данный код будет работать.

Только что мы научились осуществлять поиск по какому-то одному значению, но часто бывают ситуации, когда необходимо найти строку или строки по нескольким значениям.

Осуществляется данная операция с помощью метода *НайтиСтроки*.

НайтиСтроки

Рассмотрим синтаксис данного метода, который является функцией и возвращает массив строк таблицы значений соответствующим условиям поиска.

НайтиСтроки(<Отбор>)

У данного метода всего один обязательный параметр, это *Отбор*, который представляет собой структуру, её ключи соответствуют названиям колонок таблицы, а значения структуры – искомым значениям.

Повторюсь, что на выходе данной функции мы имеем массив, состоящий из строк, соответствующих данному отбору, причем в массиве хранятся ссылки на строки. Поэтому если Вы поменяете какое-нибудь значение в строке, то оно изменится и в таблице значений. И наоборот, если что-нибудь поменяете в таблице, то автоматически изменится и определенный элемент массива.

Доработаем наш пример: необходимо осуществлять поиск по *Фамилии* и по *Имени*. Если строки с такими данными найдены, то выведем информацию в окно сообщений. Для этого добавим еще один реквизит *Имя* на форму.

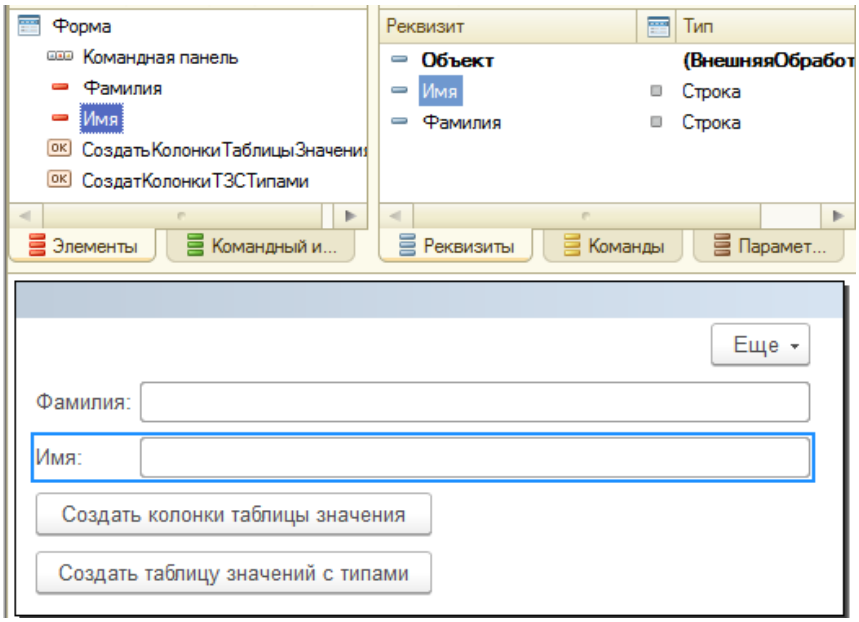


Рис. 8.1.10

Код поиска будет следующий:

```
Если Не ПустаяСтрока(Имя) и Не ПустаяСтрока(Фамилия) тогда
    СтруктураПоиска = Новый Структура;
    СтруктураПоиска.Вставить("Имя",СокрЛП(Имя));
    СтруктураПоиска.Вставить("Фамилия", СокрЛП(Фамилия));
    НайденныеСтроки = ФИО.НайтиСтроки(СтруктураПоиска);

    Если НайденныеСтроки.Количество() <> 0 тогда
        Для Каждого СтрокаФИО из НайденныеСтроки цикл
            Для н = 0 по ФИО.Колонки.Количество() - 1 цикл
                Если ТипЗнч(СтрокаФИО[н]) = Тип("Дата") тогда
                    Сообщить(ФИО.Колонки[н].Заголовок + ": " +
                             Формат(СтрокаФИО[н],"ДЛФ = ДД"));
                Иначе
                    Сообщить(ФИО.Колонки[н].Заголовок + ": " +
                             Строка(СтрокаФИО[н]));
                КонецЕсли;
            КонецЦикла;
        КонецЦикла;
    КонецЕсли;
КонецЕсли;
```

Листинг 8.1.15

Разберем данный код.

Сначала мы проверяем, есть ли данные в соответствующих реквизитах формы. Если есть, то мы создаем структуру, которую называем *СтруктураПоиска*, и добавляем в нее две пары *Ключ и значения*, причем, как Вы заметили, названия ключей соответствуют названиям колонок в таблице. После этого мы используем метод *НайтиСтроки*, который возвращает некоторый массив строк. Если он заполнен, то выводим информацию в окно сообщений, точно так же, как мы это делали в предыдущем примере.

Сохраните обработку, и посмотрите, как работает наш новый поиск.

Резюмируем. Теперь мы умеем осуществлять всевозможный поиск по таблицам значений. Потренируйтесь с поиском данных в Вашей созданной таблице, такие навыки Вам очень пригодятся в дальнейшей деятельности.

Работа с таблицей значения

Мы научились заполнять таблицу значений и осуществлять поиск по ней, научимся работать непосредственно с самой таблицей: сворачивать ее, копировать, выгружать данные из колонки, загружать данные в колонку, получать итоги по какой-нибудь одной колонке, заполнять всю таблицу одним значением и сортировать.

Свертка

Очень часто бывает необходимо *свернуть таблицу*. *Свернуть таблицу* - это значит сгруппировать значения по некоторым колонкам, причем так, чтобы все одинаковые строки в какой-нибудь колонке (или в нескольких колонках) должны слиться в одну.

Продемонстрируем это наглядно, например, у нас есть таблица следующего вида:

Дата	ФИО	Часы	Сумма
01.12.12	Иванов И.И.	10	1000
02.12.12	Иванов И.И.	8	800
03.12.12	Петров Ф.Ф.	5	500
04.12.12	Петров Ф.Ф.	7	700

Таблица 8.1.1

В ней имеются колонки с датами, фамилиями, часами и суммами.

Нам необходимо свернуть таблицу по колонке *ФИО*, причем *Часы* и *Сумма* должны быть просуммированы. И на выходе должна получиться следующая таблица:

ФИО	Часы	Сумма
Иванов И.И.	18	1800
Петров Ф.Ф.	12	1200

Таблица 8.1.2

Как выполнить такую операцию? Для этого используем метод *Свернуть* объекта *Таблица значений*.

Свернуть(<КолонкиГруппировок>, <КолонкиСуммирования>)

Первый параметр – это колонки, по которым значения в таблице будут группироваться.

Второй параметр - это колонки, по которым значения будут суммироваться.

То есть, касательно нашего примера, данная процедура должна выглядеть так:

Таблица.Свернуть("ФИО","Часы,Сумма")

Причем имейте в виду, что данный метод является процедурой, в результате выполнения которой Ваша таблица изменится: те колонки, которые не перечислены в колонках группировки или колонках суммирования, удалятся, и данные в них потеряются.

Также обращаю внимание, что списки колонок группировки и суммирования не должны пересекаться.

Реализуем приведенную нами выше задачу на практике. Создадим таблицу значений, заполним ее, а потом свернем. Для этих целей создадим новую обработку, поместим на неё

команду и создадим обработчики команды на сервере и на клиенте. В обработчиках напишем следующий код:

```
&НаСервере
Процедура СворачиваниеТаблицЗначенийНаСервере()
    КвЧисл = Новый КвалификаторыЧисла(10,2);
    ОписаниеТиповЧисло  = Новый ОписаниеТипов("Число",КвЧисл,,);

    ТаблицаОплат = Новый ТаблицаЗначений;
    ТаблицаОплат.Колонки.Добавить("Дата",Новый ОписаниеТипов("Дата",,,Новый
КвалификаторыДаты(ЧастиДаты.Дата)));
    ТаблицаОплат.Колонки.Добавить("ФИО",Новый ОписаниеТипов("Строка",Новый
КвалификаторыСтроки(100)));
    ТаблицаОплат.Колонки.Добавить("Часы", ОписаниеТиповЧисло);
    ТаблицаОплат.Колонки.Добавить("Сумма",ОписаниеТиповЧисло);

    НоваяСтрока = ТаблицаОплат.Добавить();
    НоваяСтрока.Дата = '20121201';
    НоваяСтрока.ФИО  = "Иванов И.И.";
    НоваяСтрока.Часы = 10;
    НоваяСтрока.Сумма = 1000;

    НоваяСтрока = ТаблицаОплат.Добавить();
    НоваяСтрока.Дата = '20121202';
    НоваяСтрока.ФИО  = "Иванов И.И.";
    НоваяСтрока.Часы = 8;
    НоваяСтрока.Сумма = 800;

    НоваяСтрока = ТаблицаОплат.Добавить();
    НоваяСтрока.Дата = '20121205';
    НоваяСтрока.ФИО  = "Петров Ф.Ф.";
    НоваяСтрока.Часы = 5;
    НоваяСтрока.Сумма = 500;

    НоваяСтрока = ТаблицаОплат.Добавить();
    НоваяСтрока.Дата = '20121205';
    НоваяСтрока.ФИО  = "Петров Ф.Ф.";
    НоваяСтрока.Часы = 7;
    НоваяСтрока.Сумма = 700;

    ТаблицаОплат.Свернуть("ФИО","Часы,Сумма");
КонецПроцедуры

&НаКлиенте
Процедура СворачиваниеТаблицЗначений(Команда)
    СворачиваниеТаблицЗначенийНаСервере();
КонецПроцедуры
```

Листинг 8.1.16

Не будем разбирать подробно код в данном примере, Вы без труда разберете его самостоятельно. Но для того чтобы понять, как работает данный метод, поставьте точку останова на строку свертки таблицы и посмотрите, какой была таблица до свертки (рис. 8.1.11) и что с ней стало после (8.1.12).

Индекс	Значение элемента	Тип элемента	Дата	ФИО	Часы	Сумма
0	СтрокаТаблицыЗ...	СтрокаТаблицыЗ...	01.12.2012 0:00:00	"Иванов И.И."	10	1 000
1	СтрокаТаблицыЗ...	СтрокаТаблицыЗ...	02.12.2012 0:00:00	"Иванов И.И."	8	800
2	СтрокаТаблицыЗ...	СтрокаТаблицыЗ...	05.12.2012 0:00:00	"Петров Ф.Ф."	5	500
3	СтрокаТаблицыЗ...	СтрокаТаблицыЗ...	05.12.2012 0:00:00	"Петров Ф.Ф."	7	700

Рис. 8.1.11

Индекс	Значение элемента	Тип элемента	ФИО	Часы	Сумма
0	СтрокаТаблицыЗначе...	СтрокаТаблицыЗначе...	"Иванов И.И."	18	1 800
1	СтрокаТаблицыЗначе...	СтрокаТаблицыЗначе...	"Петров Ф.Ф."	12	1 200

Рис. 8.1.12

Как Вы видите, таблица свернулась, и получилась точно такая же таблица, как в таблице 8.1.2.

Еще один интересный момент: второй параметр в процедуре *Свернуть* не является обязательным, то есть можно просто сгруппировать таблицу по определенным колонкам, притом вовсе не нужно суммировать значения где-либо. Поэкспериментируйте самостоятельно с этим методом таблиц значений.

Копирование таблиц и копирование структуры таблицы

Иногда бывает необходимо получить вторую точно такую же таблицу, из ранее созданной, то есть скопировать ее. Причем часто нужно, чтобы скопированная таблица содержала все данные, которые были в таблице-оригинале, а иногда - чтобы она была пустой, но имела точно такую же структуру, как и оригинал. В этом нам помогут два метода: *Скопировать* и *СкопироватьКолонки*.

Скопировать

Разберем метод *Скопировать*. Данный метод создает копию исходной таблицы.

Рассмотрим синтаксис данной функции. Их два.

Первый:

Скопировать(<Строки>, <Колонки>)

Этот синтаксис имеет два необязательных параметра. Первый параметр, *Строки,* - это массив строк таблицы значений, которые скопируются в новую строку, а второй параметр, *Колонки,* - это список колонок для копирования, которые перечислены через запятую.

Второй вариант синтаксиса следующий:

Скопировать(<ПараметрыОтбора>, <Колонки>)

Первый параметр является структурой, по которой будет осуществлен отбор строк в новую таблицу, по аналогии со структурой для метода *Отбор*. Второй параметр точно такой же, как и в предыдущем варианте синтаксиса.

Таким образом, если Вам необходимо перенести в таблицу только часть строк, то Вы должны решить, можно ли получить эти строки с отбором по каким-либо признакам, или нет. Если да, то используйте синтаксис с вариантом для параметра отбора. Если нет, то создайте массив, в который добавьте нужные Вам строки, и используйте синтаксис с вариантом для массива.

Создадим на форме новую команду, обработчики этой команды на сервере и на клиенте, и скопируем в серверный обработчик код создания таблицы *ТаблицаОплат* из предыдущего примера.

В коде, после создания и заполнения таблицы *ТаблицаОплат*, создадим копированием две новые таблицы.

В первую таблицу добавим первую и последнюю строку из таблицы *ТаблицаОплат* и колонки: *Дата*, *ФИО* и *Сумма*.

Во вторую вставим все строки, где колонка *ФИО* содержит «*Иванов И.И.*», и колонки *ФИО, Часы* и *Сумма*, и сразу свернем ее, просуммировав колонки по *Часам* и *Сумме*.

```
МассивДляТаблицы = Новый Массив;
МассивДляТаблицы.Добавить(ТаблицаОплат[0]);
МассивДляТаблицы.Добавить(ТаблицаОплат[ТаблицаОплат.Количество() - 1]);

ТаблицаОплат1 = ТаблицаОплат.Скопировать(МассивДляТаблицы,"Дата,ФИО,Сумма");

СтруктураОтбора = Новый Структура;
СтруктураОтбора.Вставить("ФИО","Иванов И.И.");

ТаблицаОплат2 = ТаблицаОплат.Скопировать(СтруктураОтбора,"ФИО, Часы, Сумма");
ТаблицаОплат2.Свернуть("ФИО","Часы,Сумма");
```

Листинг 8.1.17

Теперь сохраним обработку и поставим точку останова в конце процедуры, для того чтобы посмотреть, какими стали новые таблицы.

Индекс	Значение элемента	Тип элемента	Дата	ФИО	Сумма
0	СтрокаТаблицыЗначе...	СтрокаТаблицыЗначе...	01.12.2012 0:00:00	"Иванов И.И."	1 000
1	СтрокаТаблицыЗначе...	СтрокаТаблицыЗначе...	05.12.2012 0:00:00	"Петров Ф.Ф."	700

Рис. 8.1.13. «ТаблицаОплат1»

Индекс	Значение элемента	Тип элемента	ФИО	Часы	Сумма
0	СтрокаТаблицыЗначе...	СтрокаТаблицыЗначе...	"Иванов И.И."	18	1 800

Рис. 8.1.14. «ТаблицаОплат2»

Как Вы видите, в них именно те данные, которые мы хотели.

СкопироватьКолонки

Еще один метод, который создает новую таблицу, это метод *СкопироватьКолонки*. Данный метод создает таблицу с точно тем же набором колонок, как и у оригинала. Новая таблица абсолютно пустая и не содержит никакой информации из оригинала.

Синтаксис этого метода следующий:

СкопироватьКолонки(<Колонки>)

Параметр *Колонки* - это колонки, перечисленные через запятую. Если он пустой, то копируются все колонки.

Посмотрим простой пример:

НоваяТаблица = ТаблицаОплат.СкопироватьКолонки("ФИО,Часы,Сумма");

Самостоятельно поработайте с этим методом.

Выгрузка и загрузка данных в таблицу значений

Сейчас мы научимся с Вами выгружать данные из колонки и загружать данные в колонку.

Начнем с выгрузки. У таблиц значения существует метод, который позволяет выгрузить все значения из колонки в массив. Он называется *ВыгрузитьКолонку*. Рассмотрим синтаксис этого метода.

ВыгрузитьКолонку(<Колонка>)

Данный метод имеет один параметр - это колонка, значения которой мы будем выгружать. Колонку можно представить тремя способами: индекс, название колонки и как саму колонку.

Продемонстрируем, как это выглядит. Как и в предыдущий раз, создадим новую команду на форме, и в серверный обработчик скопируем код, в котором создаем таблицу оплат. Допишем этот код:

```
МассивФИО   = ТаблицаОплат.ВыгрузитьКолонку("ФИО");
МассивФИО_1 = ТаблицаОплат.ВыгрузитьКолонку(1);
МассивФИО_2 = ТаблицаОплат.ВыгрузитьКолонку(ТаблицаОплат.Колонки.ФИО);
```

Листинг 8.1.18

Посмотрите самостоятельно, какие данные выгрузятся в массив.

Теперь разберем, как загружать значения в таблицу из массива. Для этого будем использовать метод таблиц значений *ЗагрузитьКолонку*. Рассмотрим синтаксис данного метода.

ЗагрузитьКолонку(<Массив>, <Колонка>)

Первый параметр - это массив данных, которые будут загружаться в колонку, второй параметр - это колонка в одном из трех видов, которые мы применили в предыдущем методе.

Данные из массива будут загружаться по индексу, то есть первый элемент массива будет загружен в первую строку, второй - во вторую и так далее. Причем, обращаю внимание, если для колонки определены типы, то данные в массиве должны быть тех же типов, которые определены для колонки. Методу безразлично, больше элементов в массиве или меньше, если элементов массива меньше, то будет загружено ровно столько, сколько есть, остальные строки таблицы останутся незаполненными.

Рассмотрим пример: добавим в таблицу *ТаблицаОплат* колонку *Порядок*, создадим массив, который заполним цифрами от *1* до *5*, и загрузим этот массив в колонку *Порядок*.

```
Массив = Новый Массив;
Для н = 1 по 5 цикл
    Массив.Добавить(н);
КонецЦикла;

ТаблицаОплат.Колонки.Добавить("Порядок");
ТаблицаОплат.ЗагрузитьКолонку(Массив,"Порядок");
```

Листинг 8.1.19

Думаю, нет смысла комментировать данный код. Сохраним обработку и посмотрим, как изменилась наша таблица.

Индекс	Значение элем...	Тип элемента	Дата	ФИО	Часы	Сумма	Порядок
0	СтрокаТаблиц...	СтрокаТаблиц...	01.12.2012 0:00...	"Иванов И.И."	10	1 000	1
1	СтрокаТаблиц...	СтрокаТаблиц...	02.12.2012 0:00...	"Иванов И.И."	8	800	2
2	СтрокаТаблиц...	СтрокаТаблиц...	05.12.2012 0:00...	"Петров Ф.Ф."	5	500	3
3	СтрокаТаблиц...	СтрокаТаблиц...	05.12.2012 0:00...	"Петров Ф.Ф."	7	700	4

Рис. 8.1.15

Вы видите, что новая колонка добавилась и заполнилась нужными Вам значениями. Теперь самостоятельно уменьшите количество элементов массива до трех и посмотрите, как заполнится новая колонка.

Заполнить значение

Представим, что в таблице оплат мы забыли создать колонку *Валюта*, которую необходимо заполнить значением *Рубли*. Как добавить новую колонку мы знаем, но как заполнить всю колонку одним значением? Для этого используется метод *ЗаполнитьЗначения*.

Рассмотрим синтаксис данного метода:

ЗаполнитьЗначения(<Значение>, <Колонки>)

Первый параметр - это значение, которым мы будем заполнять колонки. Второй параметр - это список колонок через запятую. Второй параметр не обязательный, если он пустой, то значение заполнит собой всю таблицу. Если указано несколько колонок, то значение будет

заполнено во всех этих колонках. Если для колонок определены типы, то значение должно быть того же типа, который определен для колонок.

Доработаем наш предыдущий пример: создадим новую колонку *Валюта* и заполним значение в ней.

```
ТаблицаОплат.Колонки.Добавить("Валюта");
ТаблицаОплат.ЗаполнитьЗначения("Рубли","Валюта");
```

Листинг 8.1.20

Получение итогов

Очень часто бывает необходимо получить суммовой итог по какой-нибудь колонке. Например, по таблице *ТаблицаОплат* подсчитать, какова общая сумма. Для этого нам пригодится метод *Итог*.

Синтаксис данного метода простой:

Итог(<Колонка>)

где параметр *Колонка* – имя колонки в виде строки.

Посмотрим, как он работает:

```
Сообщить("Общая сумма: " + ТаблицаОплат.Итог("Сумма"));
```

Листинг 8.1.21

Сохраните обработку. И запустите ее заново. При выполнении команды должен выйти итог по всей колонке *Сумма*.

Сортировка

И напоследок, мы научимся сортировать таблицу значений. Делается это достаточно просто с помощью метода *Сортировать*.

Рассмотрим синтаксис данного метода:

Сортировать(<Колонки>)

У данного метода один обязательный параметр - это список колонок через запятую. После каждого названия колонки можно указывать направление сортировки. Всего два направления – по убыванию и по возрастанию.

Посмотрим, как задавать сортировку по убыванию и по возрастанию.

ТаблицаОплат.Сортировать("ФИО Убыв");

ТаблицаОплат.Сортировать("ФИО Возр");

В первом случае сортировка будет вестись по убыванию. Для этого после названия поля пишется слово *«Убыв»*.

Во втором случае сортировка будет вестись по возрастанию. Тогда соответственно пишется слово *«Возр»*.

Обращаю Ваше внимание, что сортировка ведется в порядке очередности колонок, которые указаны в параметре, то есть если Вы первым полем указали *ФИО*, а потом *Дата*, то сначала произойдет сортировка всех строк по *ФИО*, а потом внутри отсортированных строк произойдет сортировка по *Дате*.

Потренируйтесь с сортировкой таблиц самостоятельно.

Итак, мы научились работать с таблицами значений, изучили большинство нужных нам методов. Рекомендую Вам, используя синтаксис-помощник, ознакомиться с остальными методами таблиц (они несложные и вполне доступны для понимания) для полного представления всех возможностей данного объекта.

Работа с таблицей значения на форме

Мы научились работать с таблицей значений посредством встроенного языка 1С, т.е. программно, в модуле формы (все примеры справедливы также и для других модулей). В заключение части о таблице значений научимся работать с ней на форме, когда таблица значений представлена в виде реквизита формы и когда этот реквизит размещен на форме в виде *Таблицы*.

Создадим новую обработку, форму обработки, на форму добавим новый реквизит, тип которого будет *ТаблицаЗначений*.

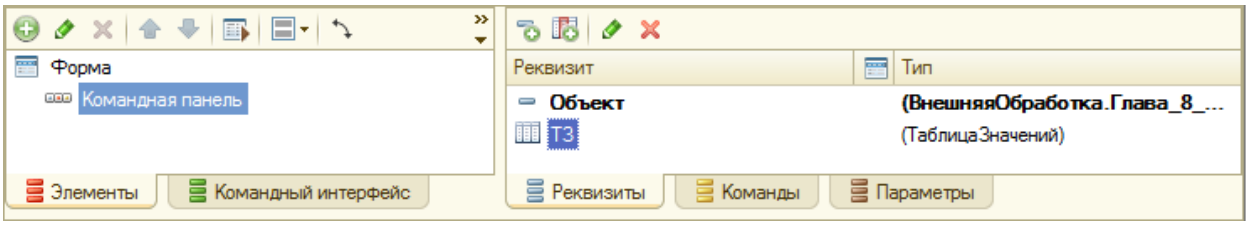


Рис. 8.1.16

Но сама по себе таблица значений с отсутствующими колонками бесполезна. Для того, чтобы добавить колонку в таблицу значений, необходимо выделить нужный реквизит (нашу таблицу значений) и нажать на кнопку «Добавить колонку реквизита» в командной панели окна реквизитов (см. рис. 8.1.17).

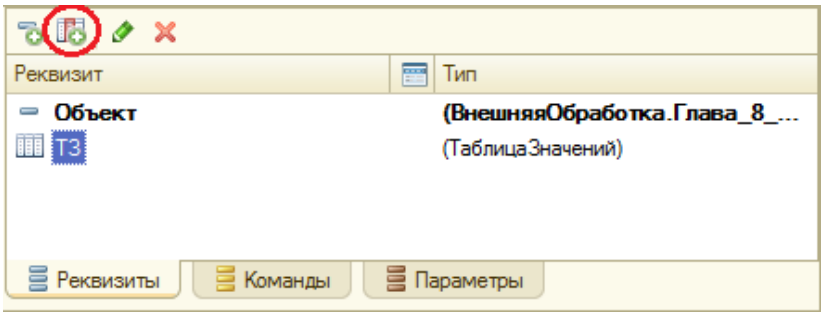


Рис. 8.1.17

После этих действий будет создана новая колонка таблицы значений, а справа откроется палитра свойств этой колонки, где можно установить название, заголовок, тип и т.д.

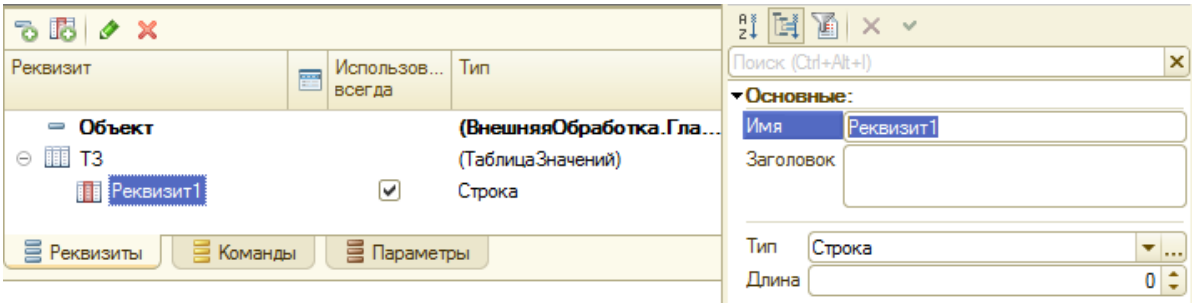


Рис. 8.1.18

Создадим следующие реквизиты:

- ДатаВыплаты (тип Дата);
- ФИО (тип Строка);
- Автомобили (тип СправочникСсылка.Автомобили);
- КоличествоЧасов (тип Число(10,1))
- ЦенаЧаса (тип Число(10,2))
- Сумма (тип Число (10,2))

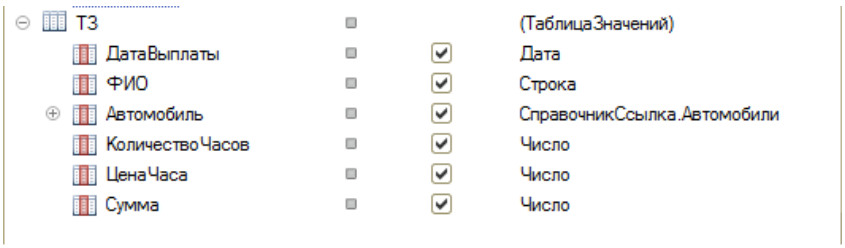


Рис. 8.1.19

Мы создали реквизит ТЗ с типом *ТаблицаЗначений,* с этим реквизитом мы спокойно можем работать в модуле формы (ниже мы научимся это делать).

Но нас сейчас интересует, как эта таблица значений будет отображаться на форме. Для отображения таблиц значений на форме используется элемент формы *Таблица*. Добавим этот элемент на форму (см. рис. 8.1.20).

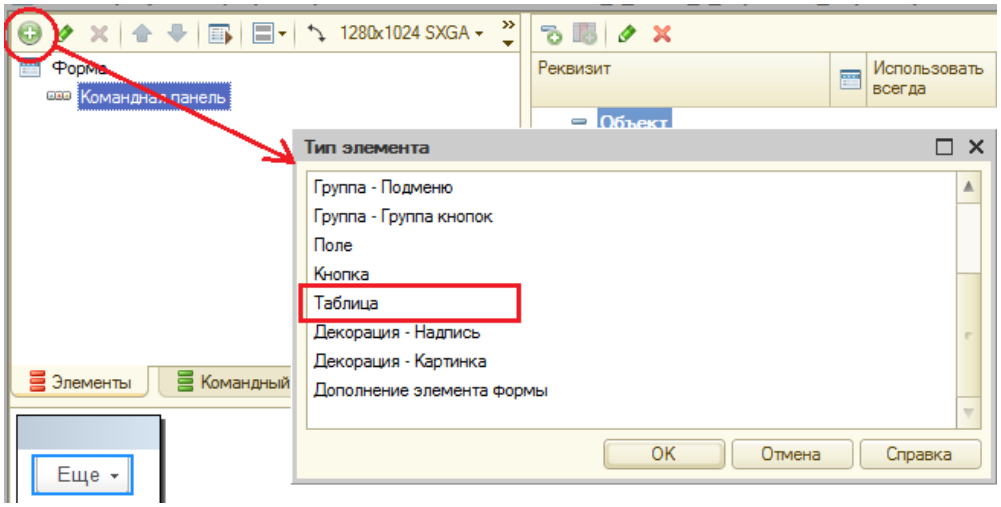


Рис. 8.1.20

После добавление элемента *Таблица*, на представлении формы ничего не изменится. Так произошло, потому что наша *Таблица* ещё не связана с реквизитом формы ТЗ и у неё не добавлены колонки. Привяжем элемент формы к реквизиту, выбрав нужную таблицу значений в свойстве *ПутьКДанным* элемента (см. рис. 8.1.21).

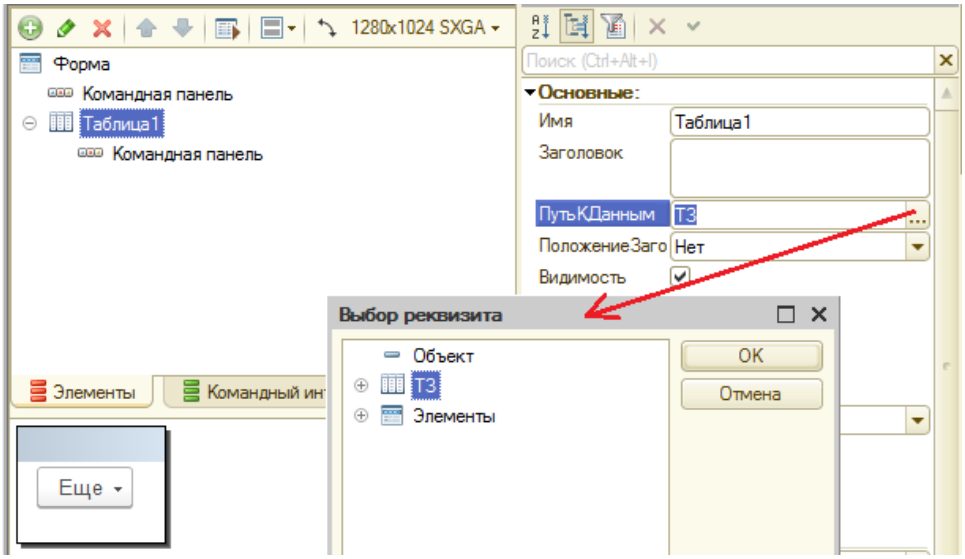


Рис. 8.1.21

После выбора будет предложено автоматически заполнить колонки, мы – откажемся, для того, чтобы научиться добавлять колонки самостоятельно. Добавить колонку в таблицу на форму можно двумя способами. Первый - просто «перетащить» нужную колонку мышкой из реквизита на форму (см. рис. 8.1.22). Тогда колонка сразу отобразится на представлении формы (см. рис. 8.1.23).

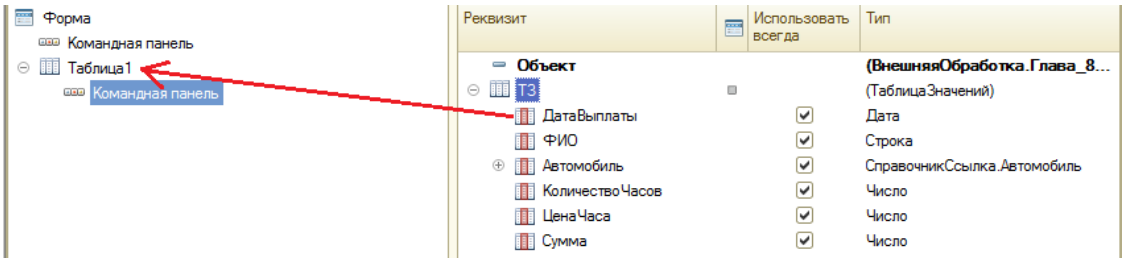


Рис. 8.1.22

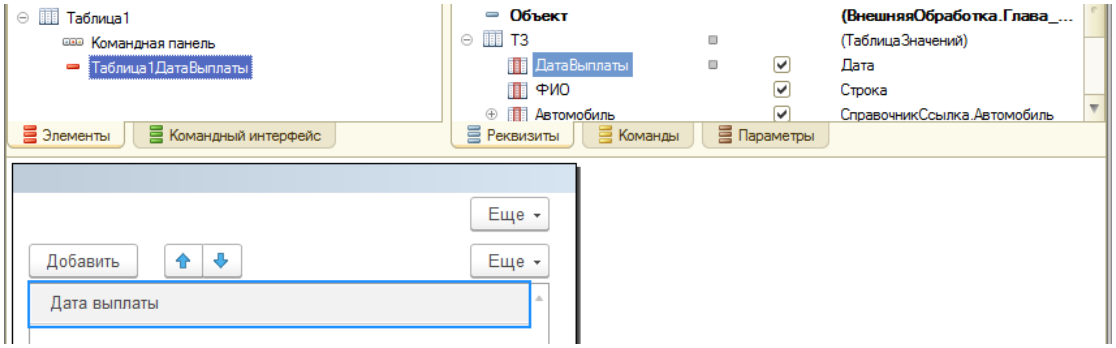


Рис. 8.1.23

Второй способ: выделить нужный элемент *Таблица1*, а после добавить новый элемент формы *Поле* (см. рис. 8.1.24). Будет добавлена новая колонку *элемента* формы, её необходимо связать с колонкой *реквизита* формы. Делается это в уже знакомом нам свойстве элемента формы *ПутьКДанным* (см. рис. 8.1.25).

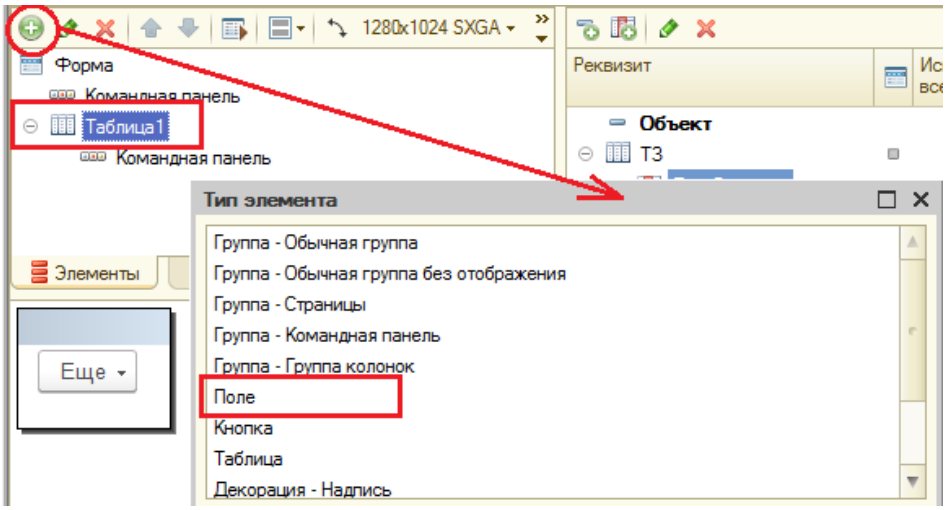


Рис. 8.1.24

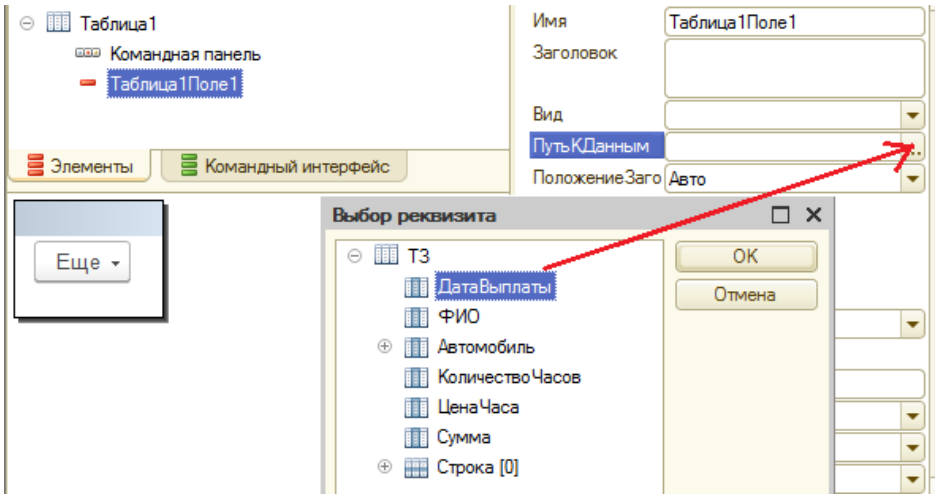


Рис. 8.1.25

После всех этих действий колонка отобразится на форме.

Добавьте самостоятельно все колонки (для тренировки сделайте сначала первым способом, а потом – вторым), чтобы получилась картинка как на рис. 8.1.26.

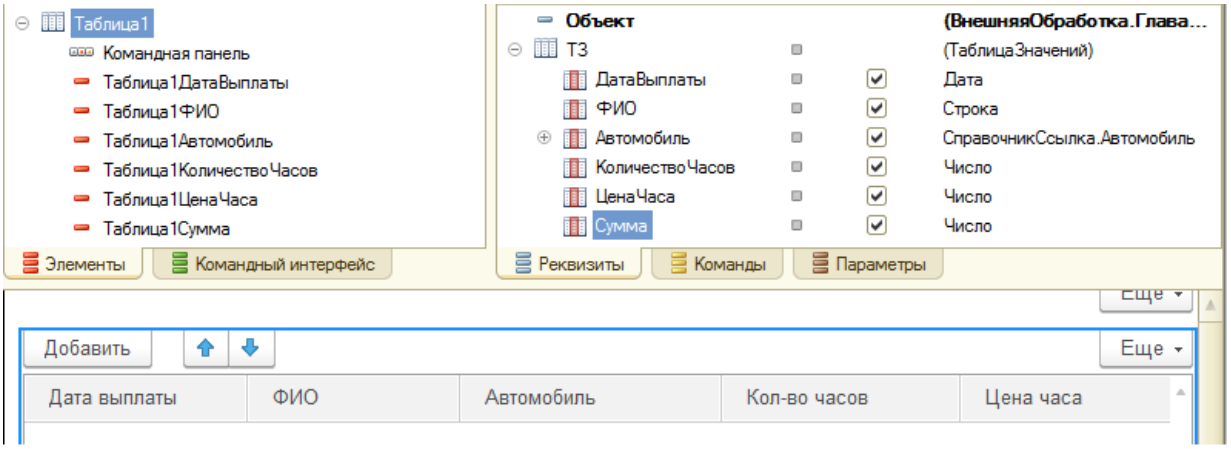


Рис. 8.1.26

Мы научились создавать таблицу на форме, связывать её с реквизитом типа *ТаблицаЗначений*, добавлять колонки к таблице на форме и связывать их с соответствующими колонками нужной таблицы значений. Теперь освоим несколько приемов оформления элемента *Таблица* на форме. И первый прием, который разберем, это группировка колонок.

Группировка колонок

При помощи группировки колонок можно сделать красивый вид нашей таблицы. Колонки можно разместить или друг над другом, или в одной ячейке. Для группировки колонок используется элемент формы *Группа* с видом *ГруппаКолонок*. К слову, это единственный возможный вид элемента *Группа* для таблиц.

Объединим две колонки: «ФИО» и «Автомобиль». Для того, чтобы добавить группу колонок, необходимо выделить элемент формы *Таблица* и добавить элемент формы «Группа – Группа колонок» (см. рис. 8.1.27).

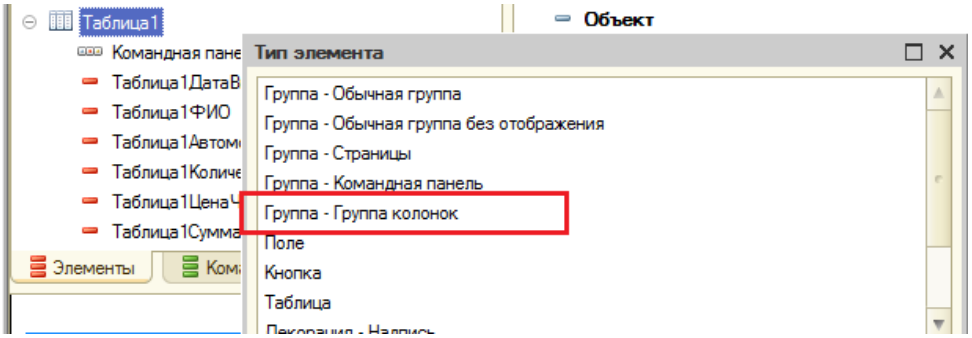


Рис. 8.1.27

После того, как группа колонок будет добавлена в таблицу, её можно двигать по таблице при помощи кнопок *Вверх* и *Вниз* командной панели окна *Элементы*, а также мышкой перетаскивать нужные поля-колонки в группу. В нашем случае должен получиться следующий вид:

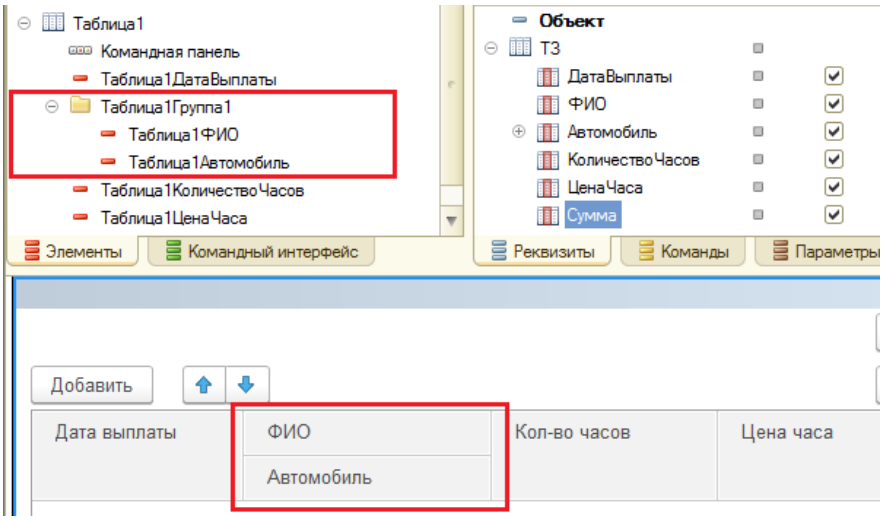


Рис. 8.1.28

Всего существует три варианта группировки: горизонтальная, вертикальная и в ячейке. Все эти варианты задаются в свойстве *Группировка* нужной группы (см. рис. 8.1.29).

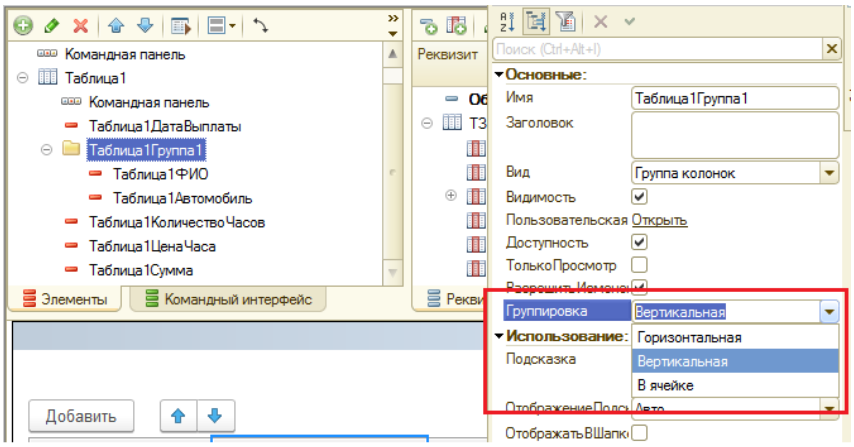


Рис. 8.1.29

О том, как группируются колонки в случае *горизонтальной* и *вертикальной* группировки, понятно из названий. Группировку «В ячейке» целесообразнее применять, когда в таблице размещены колонки, привязанные к связанным реквизитам. Например, у реквизита «Автомобиль» имеется связанный реквизит *Код*. Разместим реквизит *Код* на форме (см. рис. 8.1.30) и сгруппируем поля ввода для автомобиля и код так, чтобы вид группировки был «В ячейке» (см. рис. 8.1.31).

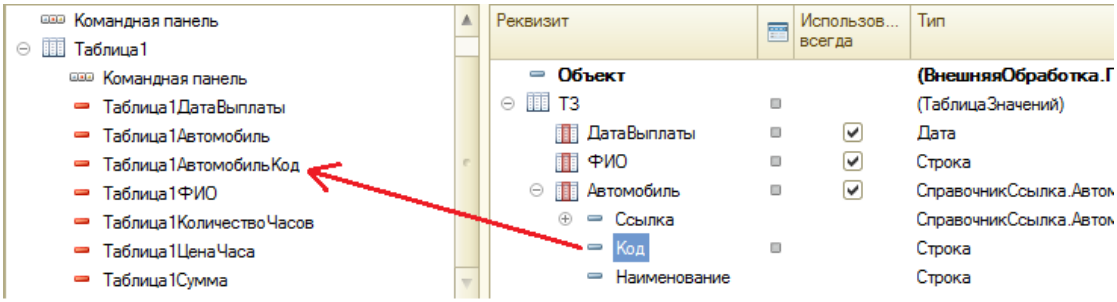


Рис. 8.1.30

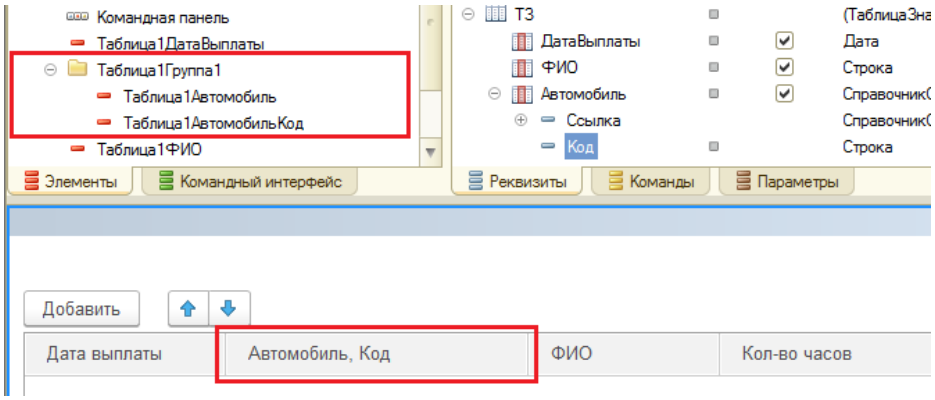


Рис. 8.1.31

Поле, связанное с основным реквизитом (в нашем случае *Автомобиль*) должно располагаться первым в группе. Теперь если мы выберем в поле таблицы какой-нибудь автомобиль, то также отобразится его код (см. рис. 8.1.32).

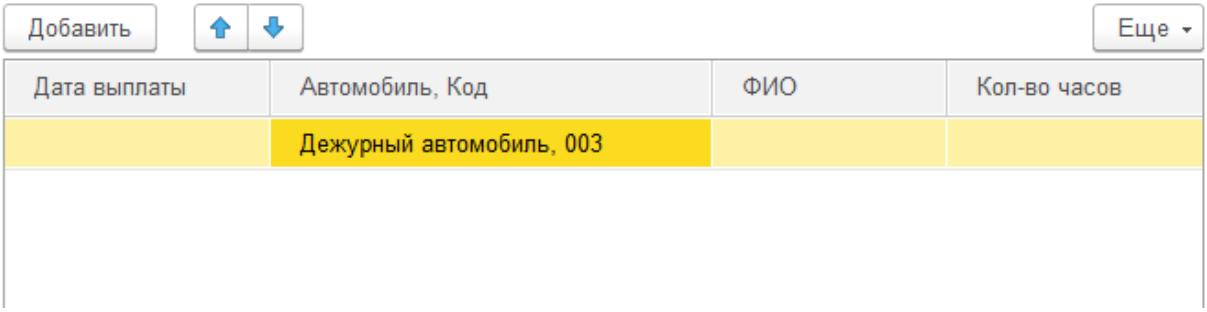


Рис. 8.1.32

Можно формировать разноуровневые группировки колонок, например, вот так:

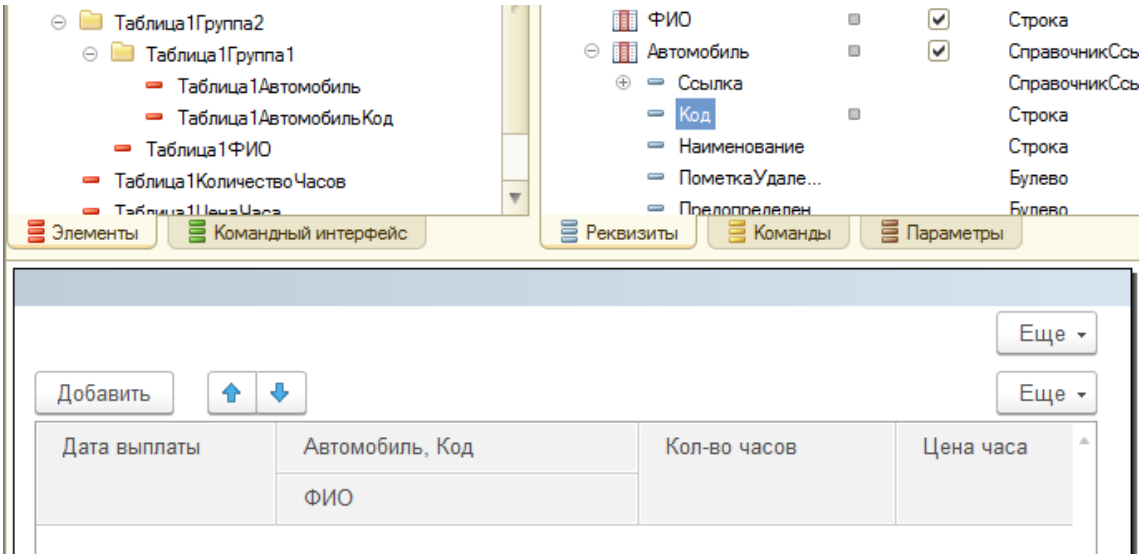


Рис. 8.1.33

Мы научились с Вами создавать таблицы, заполнять их колонками и даже как-то группировать эти колонки. Этих знаний Вам вполне хватит, чтобы начать работать с таблицами.

Программная работа с таблицей значения на форме

Мы научились создавать реквизиты с типом *ТаблицаЗначений*, размещать их на форме в виде элемента *Таблица* и конфигурировать колонки этой таблицы. Осталось научиться работать непосредственно с реквизитом типа *ТаблицаЗначений* в модуле формы (в клиентском и серверном контексте). Тут есть одна особенность. Поскольку с таблицами значений можно работать только в серверном контексте, то платформа автоматически преобразует объект *ТаблицаЗначений* в объект *ДанныеФормыКоллекция*. Этот объект предназначен для эмулирования коллекций значений при работе с ними в управляемой форме. И он доступен в серверном и клиентском контексте (включая тонкий клиент).

Когда мы задаем реквизиту формы какой-нибудь тип, который может работать только в серверном контексте (объект *ТаблицаЗначений* только один из них, все остальные мы не будем изучать в этой книге), то платформа *всегда* преобразует его в один из типов данных форм. Причем

при работе в любом контексте: и серверном, и клиентском. Ознакомится с этими объектами Вы сможете самостоятельно в синтаксис-помощнике (по пути «Интерфейс (управляемый) – Данные формы»).

Проверим это утверждение: создадим на форме команду «Записать в таблицу» и для этой команды создадим обработчики на клиенте и на сервере. Поставим точки останова в обеих процедурах – обработчиках и посмотрим в табло, что представляет собой реквизит ТЗ (с типом *ТаблицаЗначений*) в клиентском (см. рис. 8.1.34) и в серверном (см. рис. 8.1.35) контексте.

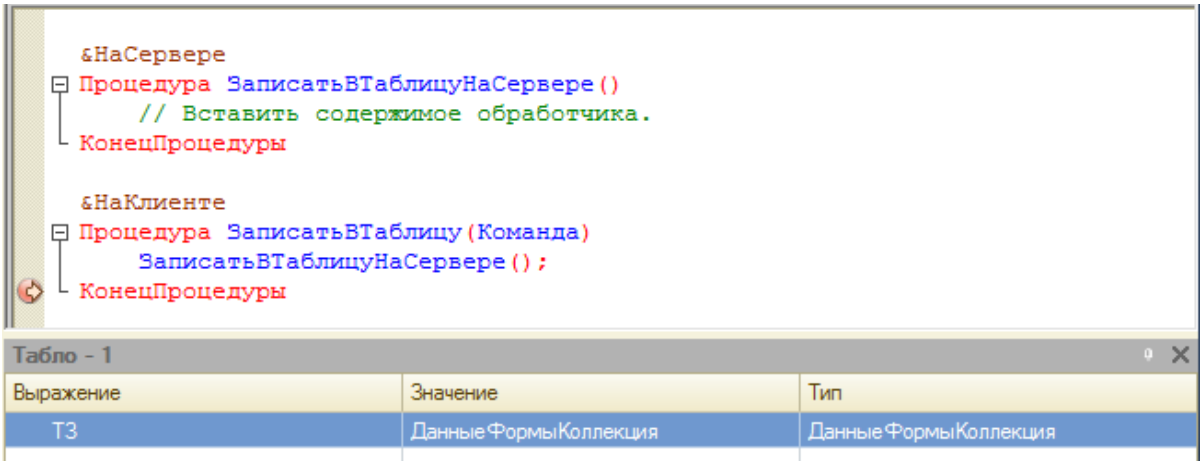


Рис. 8.1.34

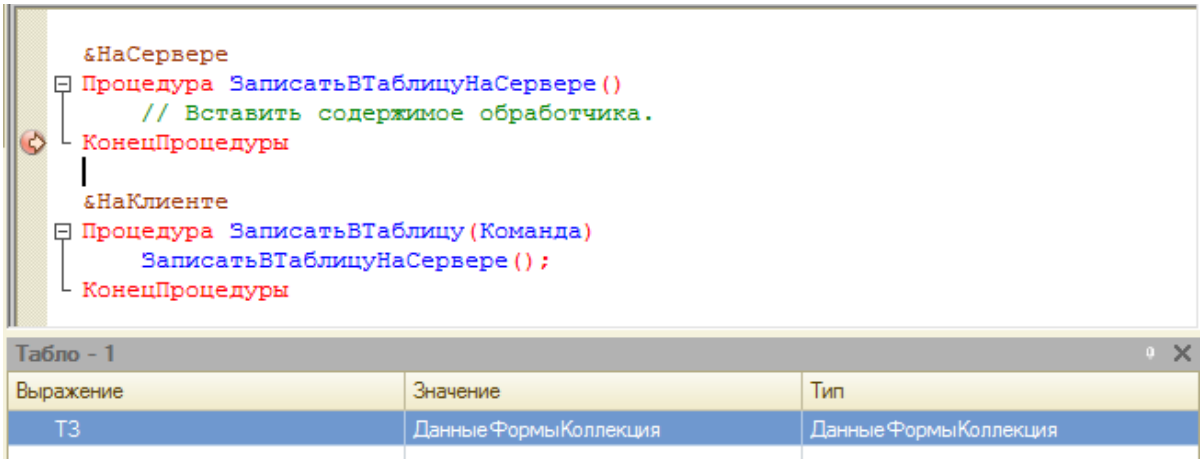


Рис. 8.1.35

Объект *ДанныеФормыКоллекция* - это коллекция некоторых значений, для которой доступен обход с помощью цикла «Для каждого …. Цикл». Она имеет несколько стандартных методов типа *Добавить, Вставить, Очистить, НайтиСтроки* и т.д. Мы не будем подробно разбирать все эти методы, со всеми методами этого объекта Вы сможете ознакомиться в синтаксис-помощнике.

Т.е. если Вы имеете цель с помощью встроенного языка добавить какую-нибудь строку в реквизит с типом *ТаблицаЗначениц*, или, наоборот, удалить, или отредактировать, то нет смысла

получать каким-то образом саму таблицу значений (о том, как это сделать, - ниже), можно напрямую работать с объектом *ДанныеФормыКоллекция*. Сделаем простой пример: в уже созданном нами обработчике добавим при помощи встроенного языка пару строк в реквизит *ТЗ* (см. листинг 8.1.22). Для этих целей нам не обязательно делать серверный вызов, мы все можем сделать на клиенте.

```
&НаКлиенте
Процедура ЗаписатьВТаблицу(Команда)
    НовСтр = ТЗ.Добавить();
    НовСтр.ДатаВыплаты = ТекущаяДата();
    НовСтр.ФИО = "Иванов И.И.";
    НовСтр.КоличествоЧасов = 10;
    НовСтр.ЦенаЧаса = 500;
    НовСтр.Сумма = НовСтр.КоличествоЧасов * НовСтр.ЦенаЧаса;

    НовСтр = ТЗ.Добавить();
    НовСтр.ДатаВыплаты = ТекущаяДата();
    НовСтр.ФИО = "Петров С.В.";
    НовСтр.КоличествоЧасов = 20;
    НовСтр.ЦенаЧаса = 1000;
    НовСтр.Сумма = НовСтр.КоличествоЧасов * НовСтр.ЦенаЧаса;
КонецПроцедуры
```

Листинг 8.1.22

Если мы Выполним эту команду, то произойдет заполнение таблицы на форме (см. рис. 8.1.36).

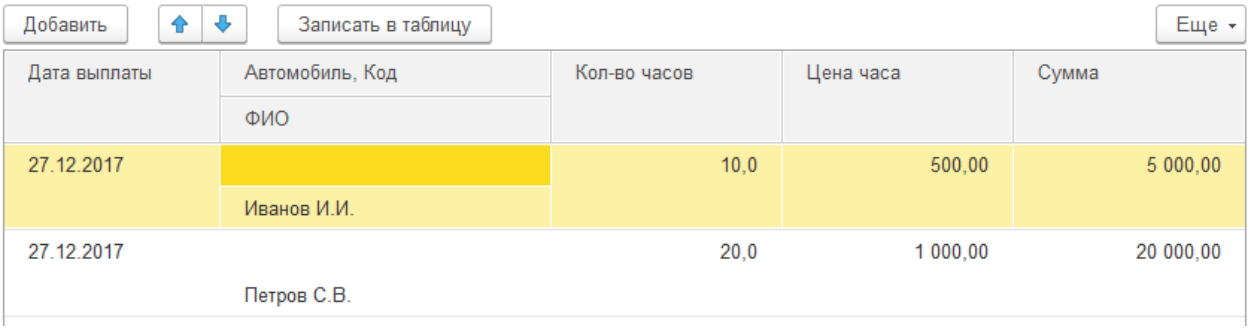


Рис. 8.1.36

Естественно, Вы заметили, что я не заполнил колонку «Автомобиль». Сделал я это намеренно, чтобы показать, как изменять реквизит формы с типом *ТаблицаЗначений*.

Для примера добавим на форму реквизит с типом *СправочникСсылка.Автомобили* и разместим его в виде поля ввода (см. рис. 8.1.37), а при изменении этого поля ввода будем перезаписывать колонку «Автомобиль» в реквизите ТЗ. Для этого создадим обработчик события *ПриИзменении* поля ввода (в клиентском контексте, см. рис. 8.1.38). А в обработчике напишем код, в котором будем заполнять колонку *Автомобиль* реквизита формы ТЗ (см. листинг 8.1.23).

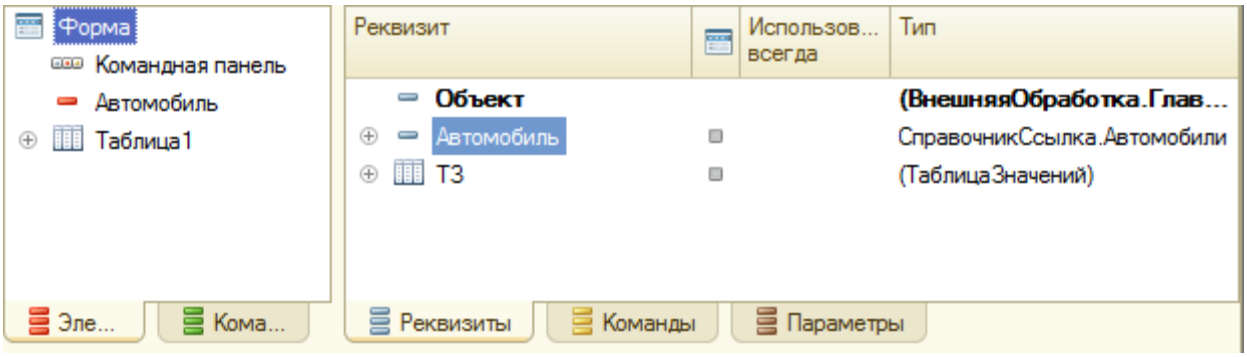


Рис. 8.1.37

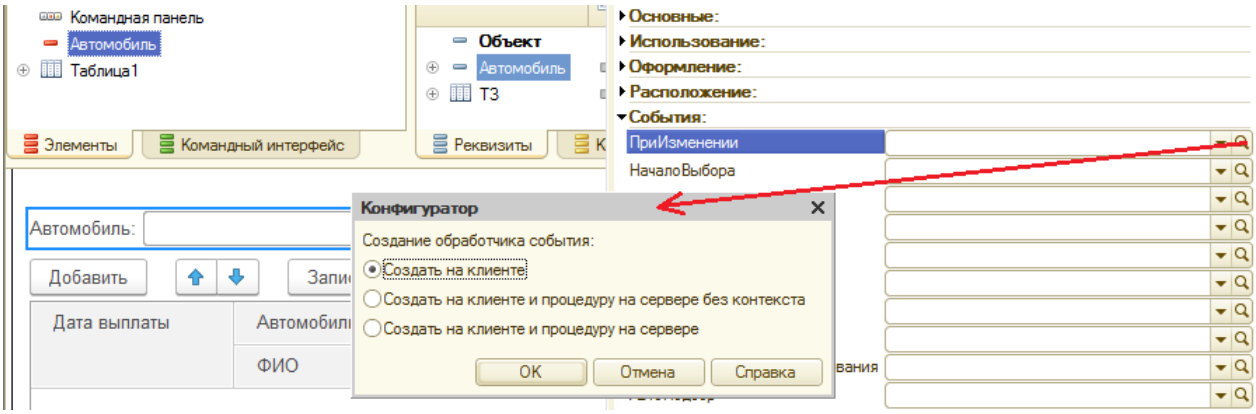


Рис. 8.1.38

```
&НаКлиенте
Процедура АвтомобильПриИзменении(Элемент)
    Если ТЗ.Количество() = 0 Тогда
        Возврат;
    КонецЕсли;
    Для Каждого стрТЗ из ТЗ  Цикл
        стрТЗ.Автомобиль = Автомобиль;
    КонецЦикла;
КонецПроцедуры
```

Листинг 8.1.23

Самостоятельно проверьте работу этого метода.

Научимся получать таблицу значений из объекта *ДанныеФормыКоллекция* и, наоборот, загружать таблицу значений в объект *ДанныеФормыКоллекция.* Сделать обе операции можно двумя способами. Первый способ: с помощью метода управляемой формы *РеквизитФормыВЗначение* получить из реквизита таблицу значения, а потом, обработав её, загрузить обратно в реквизит при помощи метода и *ЗначениеФормыВРеквизит*.

Второй способ: получить таблицу значения, используя метод коллекции *Выгрузить*, а потом, обработав её, загрузить обратно при помощи метода коллекции *Загрузить*.

Для того, чтобы понять, как работают оба способа, сделаем одну задачку: будем сворачивать таблицу значения, суммируя поля *КоличествоЧасов* и *Сумма*. Для этого создадим две команды: «Свернуть первым способом» и «Свернуть вторым способом» с обработчиками в

клиентском и серверном контексте. Для красивого расположения команд, в командной панели создадим группу «Подменю» (см. рис. 8.1.39).

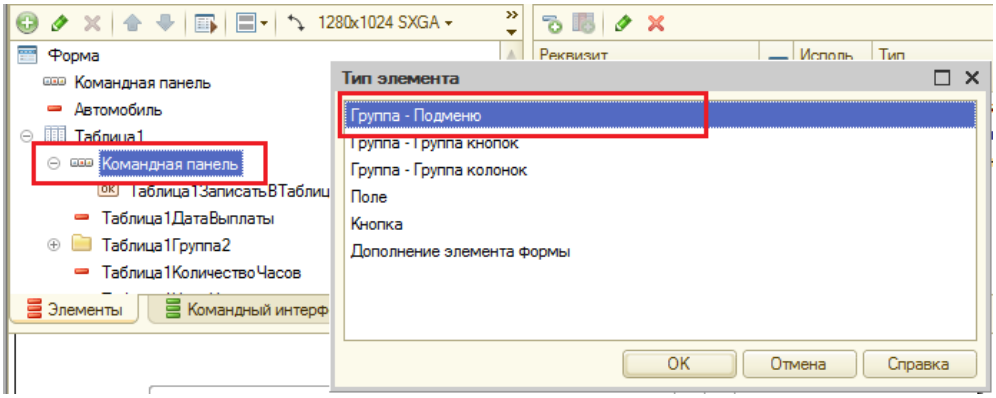


Рис. 8.1.39

И перетащим в эту группу обе команды (которые Вы сделали самостоятельно). Заголовок для группы будет «Сворачивать». И если все правильно сделали, то должна получиться такая же красивая картинка как на рис. 8.1.40.

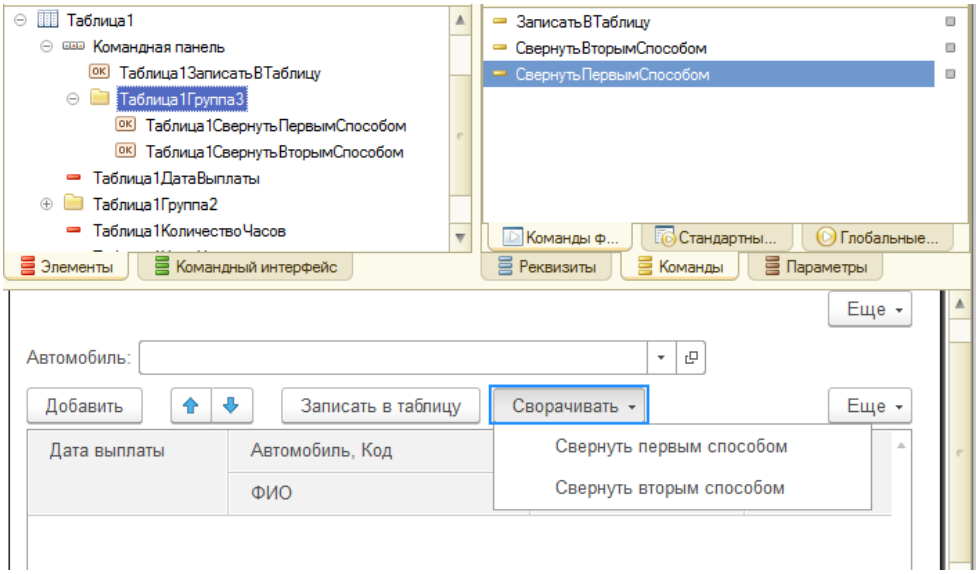


Рис. 8.1.40

В обработчике команды «Свернуть первым способом» напишем код, в котором будем сворачивать таблицу значений при помощи методов формы (см. листинг 8.1.24), а в другом обработчике напишем код, в котором свернем таблицу значений при помощи методов коллекции (см. листинг 8.1.25).

```
&НаСервере
Процедура СвернутьПервымСпособомНаСервере()
    ТаблицаДляСвертки = РеквизитФормыВЗначение("ТЗ");
    ТаблицаДляСвертки.Свернуть("ДатаВыплаты,ФИО,Автомобиль,ЦенаЧаса",
                               "КоличествоЧасов,Сумма");
    ЗначениеВРеквизитФормы(ТаблицаДляСвертки,"ТЗ");
КонецПроцедуры
```

```
&НаКлиенте
Процедура СвернутьПервымСпособом(Команда)
    СвернутьПервымСпособомНаСервере();
КонецПроцедуры
```

Листинг 8.1.24

```
&НаСервере
Процедура СвернутьВторымСпособомНаСервере()
    ТаблицаДляСвертки = ТЗ.Выгрузить();
    ТаблицаДляСвертки.Свернуть("ДатаВыплаты,ФИО,Автомобиль,ЦенаЧаса",
                               "КоличествоЧасов,Сумма");
    ТЗ.Загрузить(ТаблицаДляСвертки);
КонецПроцедуры

&НаКлиенте
Процедура СвернутьВторымСпособом(Команда)
    СвернутьВторымСпособомНаСервере();
КонецПроцедуры
```

Листинг 8.1.25

С методом формы *РевизитФормыВЗначение* мы знакомы, он создает переменную того типа, который указан в реквизите на форме. Таким образом, любую переменную из данных форм при помощи этого метода можно преобразовать в изначальный тип.

А метод *ЗагрузитьВРеквизитФормы* выполняет обратную функцию: загружает какую-либо переменную в реквизит формы. Причем переменная должна быть того же типа, который указан у реквизита.

С методами коллекции данных формы *Выгрузить* и *Загрузить* все понятно. Ознакомьтесь с ними самостоятельно в синтаксис-помощнике.

На этом наше изучение таблиц значений закончилось, Вы много усвоили и поняли в этой главе, и данная информация очень пригодится для дальнейшей работы.

А работать Вы с ними будете часто, поэтому не бойтесь применять таблицы значений. Если Вы раньше писали на другом языке программирования (например, Паскаль), то у Вас будет огромное желание использовать массивы, это делать нежелательно, поскольку таблицы значений суть динамический двумерный массив, который гораздо удобнее использовать в своей работе. Замечу, что точно такие же методы, как и методы таблиц значений, имеют многие коллекции значений. Поэтому если Вы хорошо научитесь работать с таблицами значений, то освоить остальные коллекции не будет составлять особого труда.

Часть 2. Табличные части справочников и документов

В предыдущей части этой главы мы научились работать с таблицами значений, в этой части мы разберем, как работать с табличными частями справочников и документов. Вы уже знаете из четвертой главы, что у любого справочника и документа можно сконструировать табличную часть.

Вспомним, как выглядят табличные части справочников и документов.

Для этого раскройте Вашу конфигурацию, созданную в 4 главе. И разверните справочник *Автомобили.* Внутри данного справочника Вы увидите элемент узла *Табличные части*. Разверните этот элемент.

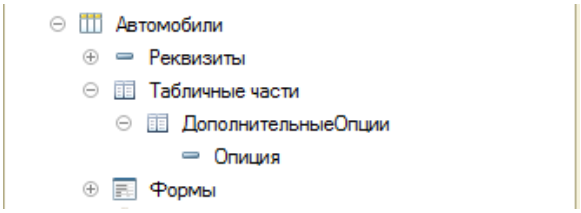


Рис. 8.2.1

И внутри этого элемента Вы увидите единственную табличную часть – *ДополнительныеОпции*, у которой только один реквизит - это *ДополнительныеОпции*.

Можно создавать несколько табличных частей, и неограниченное количество реквизитов в ней. Посмотрим, каким образом данная табличная часть выглядит на форме.

Откроем форму элемента справочника (если она создана; если нет, то создайте самостоятельно).

Как видите, на форме присутствует уже знакомый Вам элемент *Таблица* с двумя колонками. Откройте палитру свойств этого элемента и обратите внимание на свойство *ПутьКДанным* (см. рис. 8.2.3).

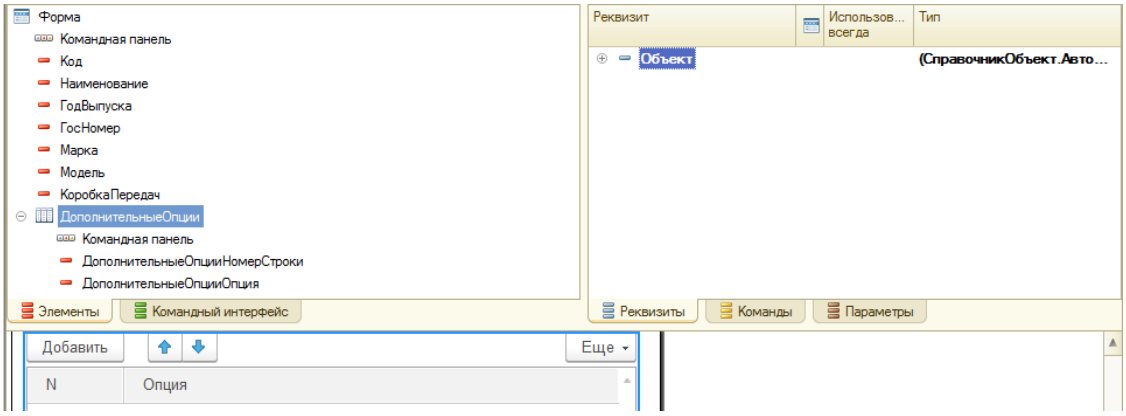


Рис. 8.2.2

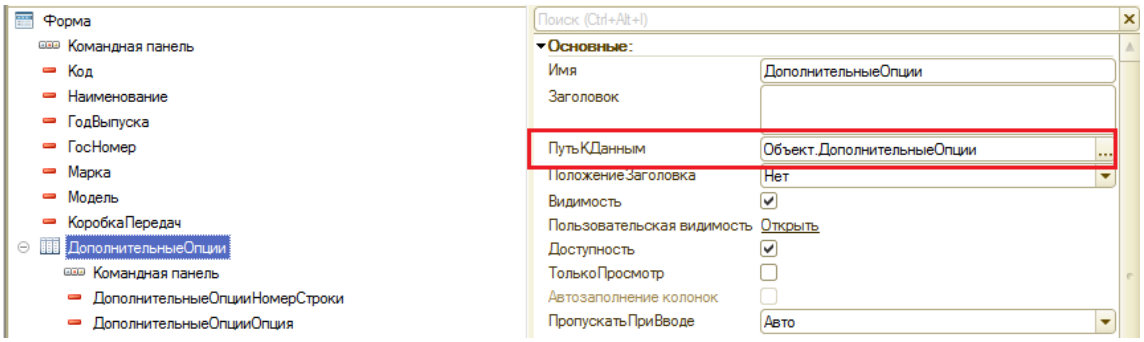


Рис. 8.2.3

Обратите внимание, что в свойстве *ПутьКДанным* указано *Объект.ДополнительныеОпции*, раскройте это свойство (см. рис. 8.2.4).

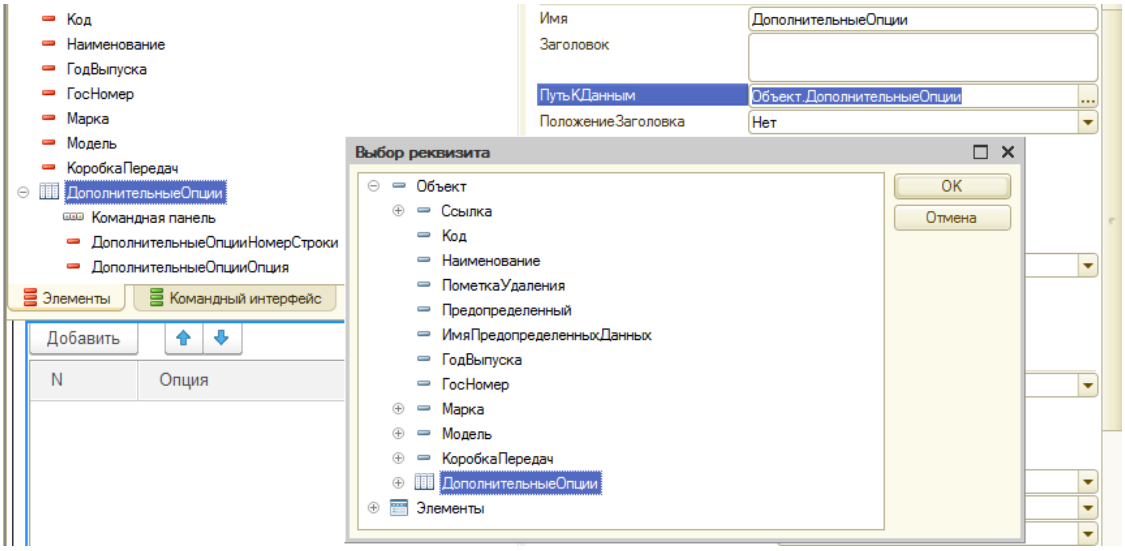


Рис. 8.2.4

Вы видите, что ДополнительныеОпции входят в состав основного реквизита формы *Объект*, тип которого *СправочникОбъект.Автомобили*. Если бы у нас была еще одна табличная часть, она бы тоже присутствовала.

В управляемом приложении принципы работы с табличной частью объекта (документа или справочника) посредством формы и напрямую отличаются. Поэтому мы сначала научимся работать с табличной частью напрямую, через объект, а потом используя управляемую форму.

Работа с табличной частью

Для того, чтобы показать, как работать с табличными частями объектов (мы будем тренироваться на документах), решим следующую задачу: оператор в обработке забивает в таблицу на форме данные о ценах на топливо разных поставщиков. После того, как все данные введены (они могут быть введены хаотично), оператор выполняет команду «Создать документы», в результате этих действий будут созданы документы «Установка цен на топливо». Причем их

будет создано столько, сколько в целом введено поставщиков. Например, в таблице 10 строк, в 3 из них информация по поставщику «ТопливоОйл», а в остальных по поставщику «НефтьСэйл». В этом случае должно быть создано два документа «Установка цен на топливо». Для каждого поставщика свой документ.

Для реализации задачи создадим обработку, форму обработки, на форме обработки создадим реквизит с типом *ТаблицаЗначений*, у которой будут следующие поля:

- Поставщик (тип *СправочникСсылка.ПоставщикиТоплива*),
- ТипТоплива (тип *СправочникСсылка.ТипыТоплива*) ,
- Цена (тип *Число(10,2)*).

На форме, также будет создана команда «Создать документы».

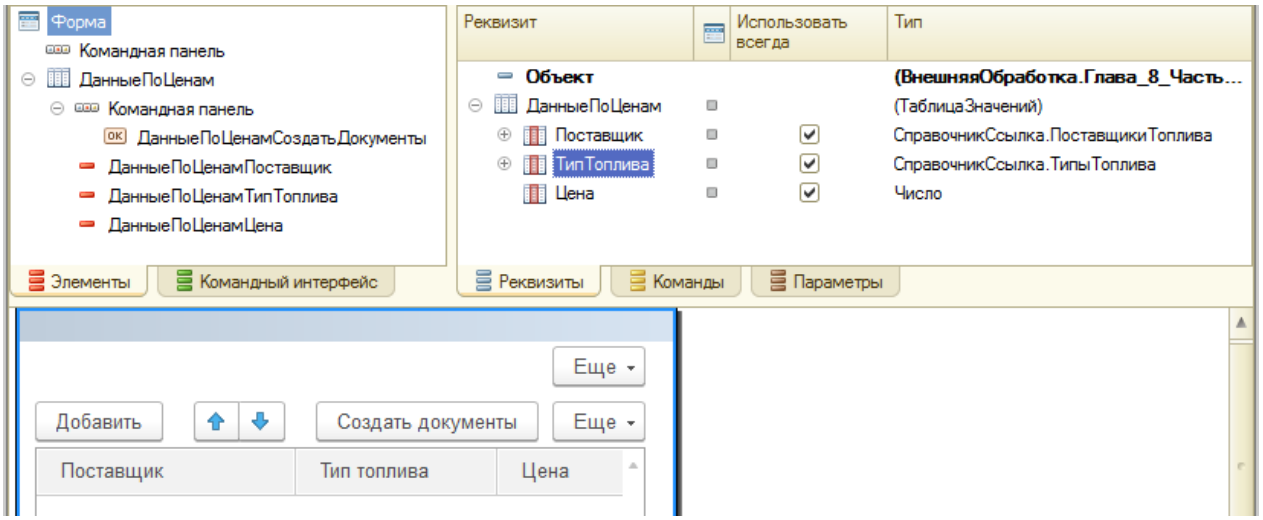


Рис. 8.2.5

Создадим обработчики команды в серверном и клиентском контексте. И в серверном контексте выполним следующий алгоритм:

1) Сгруппируем всех поставщиков в отдельную таблицу;
2) Для каждого поставщика из сгруппированной таблицы будем создавать отдельный документ «Установка цен на топливо»;
3) Заполним табличную часть созданного документа ценами на топливо для соответствующего поставщика.
4) Сохраним и проведем документ.

Создадим обработчики команды «Создать документы» в клиентском и серверном контексте и напишем в них следующий код. В листинге к этому коду я дал пояснения, поэтому в целом комментировать этот код не буду.

```
&НаСервере
Процедура СоздатьДокументыНаСервере(ДатаДокумента)
      //получаем таблицу значения, выгрузив её
      //из объекта ДанныеФормыКоллекция
      ТЗ_ДанныеПоЦенам = ДанныеПоЦенам.Выгрузить();
      //создаем новую таблиц значения при помощи метода
      //Скопировать, причем в этой таблице будет только поставщик
```

```
        ТЗ_Поставщиков = ТЗ_ДанныеПоЦенам.Скопировать(,"Поставщик");
        //сворачиваем
        ТЗ_Поставщиков.Свернуть("Поставщик");
        //обходим свернутую таблицу поставщиков
        Для каждого стрПоставщик Из ТЗ_Поставщиков  Цикл
            //создаем новый документ
            НовДокумент = Документы.УстановкаЦенНаТопливо.СоздатьДокумент();
            НовДокумент.Дата = ДатаДокумента;
            НовДокумент.Поставщик = стрПоставщик.Поставщик;
            //отбираем цены и типы для поставщика конкретного поставщика
            МассивЦен = ТЗ_ДанныеПоЦенам.НайтиСтроки(Новый
                            Структура("Поставщик",стрПоставщик.Поставщик));
            //обходим отобранные цены
            Для каждого стрЦена Из МассивЦен Цикл
                //добавляем новые строки в табличную часть документа
                НовСтрокаТабЧасти = НовДокумент.ЦеныТоплива.Добавить();
                ЗаполнитьЗначенияСвойств(НовСтрокаТабЧасти,СтрЦена);
            КонецЦикла;
            НовДокумент.Комментарий = "#Создан из обработки ввода данных";
            Попытка
                //проводим документ
                НовДокумент.Записать(РежимЗаписиДокумента.Проведение)
            Исключение
                Сообщить("Не удалось записать документ установки цен для
поставщика " +
                         Строка(стрПоставщик.Поставщик));
            КонецПопытки;
        КонецЦикла;
КонецПроцедуры

&НаКлиенте
Процедура СоздатьДокументы(Команда)
    СоздатьДокументыНаСервере(ТекущаяДата());
КонецПроцедуры
```

Листинг 8.2.1

Поясню некоторые моменты.

Получить доступ к табличной части документа или справочника достаточно просто: обращение к ней идет как к свойству (по имени) у переменной ссылочного или объектного типа (точка, а после название табличной части). После этого мы получаем объект, общее название которого будет *ТабличнаяЧасть*. Не будем вдаваться в особенности этого объекта, отмечу только, что он имеет почти все те же методы, которые мы изучали у таблиц значений: *Добавить*, *Вставить*, *Итог*, *НайтиСтроки* и т.д., с тем же принципом работы.

Когда мы применяем метод *Добавить* табличной части, то создается новая строка табличной части объекта, которую мы впоследствии заполняем при помощи метода глобального контекста *ЗаполнитьЗначенияСвойств*. Эта процедура заполняет данные в первом параметре по данным из второго параметра. Данные сопоставляются по названиям и по типам.

Проверьте самостоятельно работу этого кода, если Вам не понятен принцип работы с табличными частями, зайдите в отладку и посмотрите на значение переменной *НовДокумент.ЦеныТоплива*.

Если Вы будете хорошо тестировать этот код, то обнаружите баг – ошибку: когда мы вводим несколько одинаковых строк, то не можем провести документ.

Для устранения бага сделаем небольшую интерфейсную задачу. Для того, чтобы избежать ситуации, когда у одного поставщика на один вид топлива пользователь может ввести две разных цены, реализуем проверку после ввода значения в любое поле. Для этого будем использовать событие элемента формы *Таблица – ПередОкончаниемРедактирования*. Этот обработчик вызывается перед окончанием редактирования строки, и у него имеется параметр *Отказ*, и если этот параметр принимает значение «Истина», то редактирование не будет закончено.

Создайте самостоятельно этот обработчик элемента формы *ДанныеПоТаблицам*, и напишите следующий код:

```
&НаКлиенте
Процедура ДанныеПоЦенамПередОкончаниемРедактирования(Элемент, НоваяСтрока,
ОтменаРедактирования, Отказ)
    ТекущиеДанные = Элемент.ТекущиеДанные;
    Если ЗначениеЗаполнено(ТекущиеДанные.Поставщик) и
        ЗначениеЗаполнено(ТекущиеДанные.ТипТоплива) и
        ЗначениеЗаполнено(ТекущиеДанные.Цена) и
        Не ОтменаРедактирования тогда
            Отказ = ЕстьСтрокиДляПоставщикаИТоплива(ТекущиеДанные.Поставщик,
                                                    ТекущиеДанные.ТипТоплива);
            Если Отказ Тогда
                Сообщить("Нельзя создавать дубли строк!");
            КонецЕсли;
    КонецЕсли;
КонецПроцедуры

&НаКлиенте
Функция ЕстьСтрокиДляПоставщикаИТоплива(Поставщик, ТипТоплива)
     СтруктураОтбора = Новый Структура;
     СтруктураОтбора.Вставить("Поставщик",Поставщик);
     СтруктураОтбора.Вставить("ТипТоплива",ТипТоплива);

     МассивОтбора = ДанныеПоЦенам.НайтиСтроки(СтруктураОтбора);
     Возврат ?(МассивОтбора.Количество() >= 2, Истина, Ложь);
КонецФункции // ЕстьСтрокиДляПоставщикаИТоплива()
```

Листинг 8.2.2

Разберем этот код. В процедуре-обработчике *ДанныеПоЦенамПередОкончаниемРедактирования* мы получаем текущие данные строки таблицы формы. Делаем это с помощью параметра обработчика *Элемент*, в котором содержится ссылка на элемент формы, к которому привязан обработчик. *ТекущиеДанные* - это свойство таблицы, которое предоставляет доступ к текущим данным строки.

Следующим шагом мы проверяем, все ли колонки текущей строки заполнены, а также смотрим на значение параметра обработчика «ОтменаРедактирования», чтобы пользователь смог отменить ввод каких-то данных.

А в функции *ЕстьСтрокиДляПоставщикаИТоплива* проверяем, есть ли уже строки для текущей комбинации «Поставщик-Топливо». С помощью метода *НайтиСтроки* объекта

ДанныеФормыКоллекция мы узнаем, сколько уже есть строк с заданной комбинацией. И если их больше двух (т.е. уже ввели лишнюю), то отказываем в окончании редактирования.

Работа с табличной частью на управляемой форме

В начале этой части мы узнали, что табличные части справочников и документов присутствуют на форме в виде уже знакомого Вам элемента формы *Таблица*. У этого элемента в свойстве *ПутьКДанным* указан путь к табличной части через основной реквизит формы *Объект*. Посмотрим в отладке, какой тип имеет объект вид *Объект.<НазваниеТабличнойЧасти>*. Для этого мы возьмем форму документа *Установка цен на топливо*, на которой размещена *Таблица,* связанная с табличной частью документа *ЦеныТоплива* (см. рис. 8.2.6).

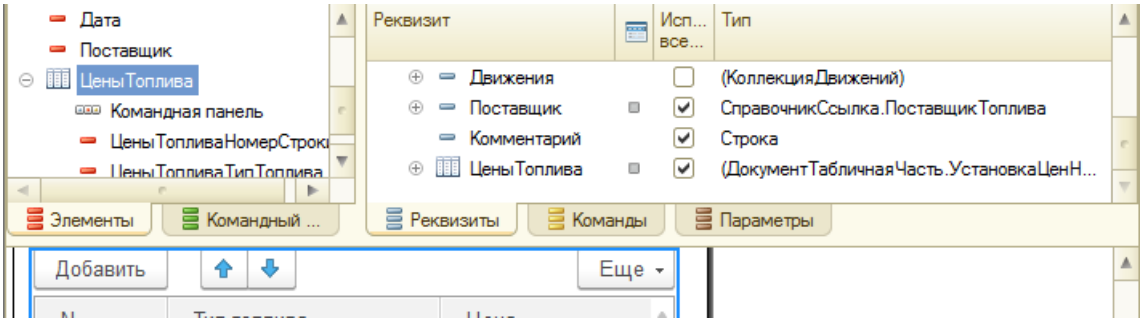


Рис. 8.2.6

Чтобы узнать, какой тип у выражения вида *Объект.ЦеныТоплива,* установим точку останова в конце события *ПриОткрытии* и посмотрим на значение этого выражения в отладке (см. рис. 8.2.7).

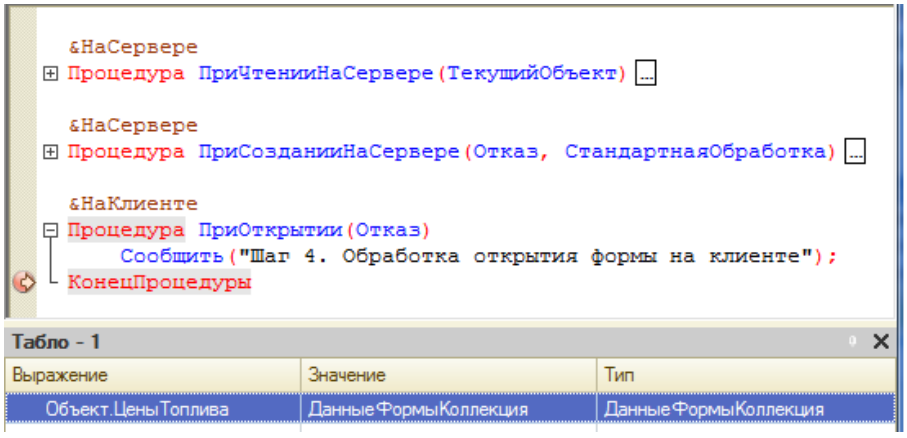


Рис. 8.2.7

Как видите, табличная часть нам доступна посредством уже знакомой нам коллекции *ДанныеФормыКоллекция*. Поэтому нет никаких особых сложностей по работе с табличной частью на управляемой форме: мы просто изменяем коллекцию, а после сохранения документа эти изменения запишутся непосредственно в табличную часть документа.

Выполним несколько простых заданий, для того чтобы понять, как с этим со всем работать. Первое задание будет следующим: создадим на форме команду, после выполнения этой команды по всей табличной части будет устанавливаться одинаковая цена. Цену эту пользователь будет вводить при помощи уже знакомого нам по первым главам метода «ВвестиЧисло». Самостоятельно сделайте команду «Установить одинаковую цену», подменю *Заполнить* в командной панели таблицы *ЦеныТоплива* и разместите команду «Установить одинаковую цену» в этом подменю. Должно получиться как на рис. 8.2.8:

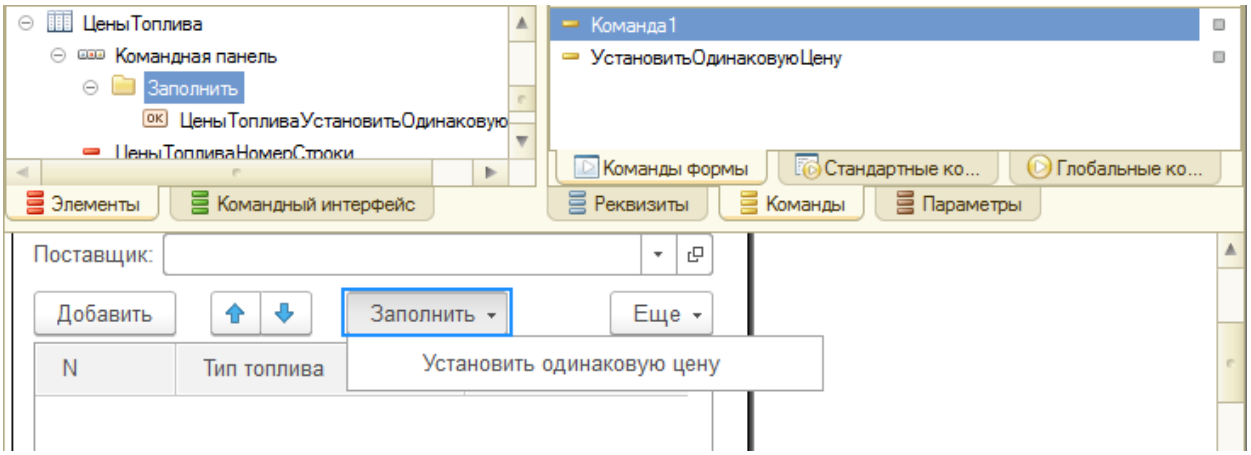


Рис. 8.2.8

Для команды «Установить одинаковую цену» создадим обработчик в клиентском контексте, в котором напишем следующий код:

```
&НаКлиенте
Процедура УстановитьОдинаковуюЦену(Команда)

    Перем НоваяЦена;

    Если ВвестиЧисло(НоваяЦена,"Новая цена",10,2) Тогда
        Для каждого стрЦен Из Объект.ЦеныТоплива  Цикл
            стрЦен.Цена = НоваяЦена;
        КонецЦикла;
        Модифицированность = Истина;
    КонецЕсли;

КонецПроцедуры
```

Листинг 8.2.3

В листинге 8.2.3, если пользователь ввел число, мы обходим коллекцию *Объект.ЦеныТоплива* и изменяем каждый элемент коллекции. Чтобы пользователь увидел, что данные на форме изменились, мы устанавливаем свойству формы *Модифицированность* значение *Истина*.

Самостоятельно сделайте команды, которые будут прибавлять к цене определенное число (вводит пользователь) и вычитать из цены определенное число (тоже вводит пользователь).

В коллекцию данных формы можно делать почти все что угодно: добавлять строки, удалять строки, очищать её. Сделаем небольшой пример, чтобы показать, как работает добавление строк: будем заполнять табличную часть всеми типами топлива. Для этого сделайте команду «Заполнить всеми типами цен», которую разместите в созданной ранее группе подменю «Заполнить». Для этой команды создайте обработчики на клиенте и на севере. В серверном обработчике напишите следующий код:

```
&НаСервере
Процедура ЗаполнитьВсемиТипамиЦенНаСервере()
    Объект.ЦеныТоплива.Очистить();
    Выборка = Справочники.ТипТоплива.Выбрать();
    Пока Выборка.Следующий() Цикл
        НовТипЦен = Объект.ЦеныТоплива.Добавить();
        НовТипЦен.ТипТоплива = Выборка.Ссылка;
    КонецЦикла;
    Модифицированность = Истина;
КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьВсемиТипамиЦен(Команда)
    ЗаполнитьВсемиТипамиЦенНаСервере();
КонецПроцедуры
```

Листинг 8.2.4

В процедуре *ЗаполнитьВсемиТипамиЦенНаСервере* мы получаем выборку цен (подробно выборки будем изучать в следующей главе). Обходим эту выборку и в каждой итерации создаем новую строку коллекции данных формы, у которой точно такие же колонки, как и у табличной части документа, и у реквизита «Объект.ЦеныТоплива».

Потренируйтесь с последней командой. Например, можете ввести цену на топливо при помощи уже знакомого Вам метода «ВвестиЧисло».

На этом мы закончим изучать работу с табличными частями справочников и документов. Кроме справочников и документов табличные части могут быть у планов видов характеристик, обработок и т.д. Работа с табличными частями других метаданных точно такая же, как и работа с табличными частями документов или справочников.

Часть 3. Элемент формы «Таблица»

Мы научились работать с таблицами значений и табличными частями справочников и документов. Все эти объекты на форму выводятся в виде элемента формы *Таблица*. В заключительной части текущей главы мы изучим некоторые особенности работ с этим элементом.

Работа с подвалом таблицы

У каждой таблицы формы есть возможность вывести *подвал* – область в нижней части таблицы. Для того, чтобы научиться работать с подвалом таблицы, откроем форму документа *Отчет о расходе топлива* (если её нет, то создайте самостоятельно). Откройте палитру свойств таблицы «Автомобили» и обратите внимание на свойство *Подвал* (см. рис. 8.3.1). По умолчанию оно всегда выключено.

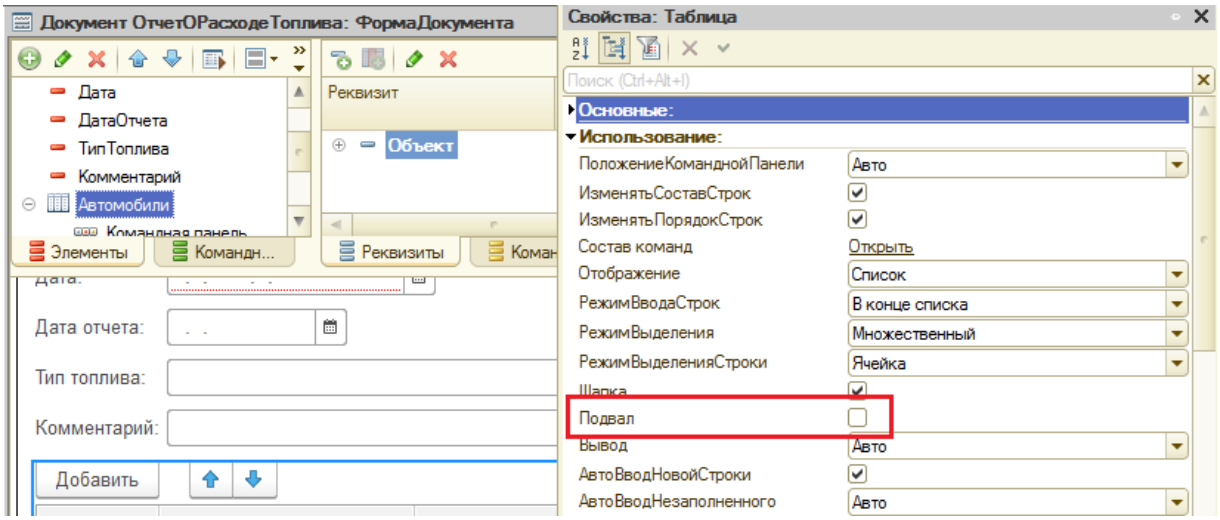


Рис. 8.3.1

Включите это свойство и обратите внимание: внизу таблицы появится область (см. рис. 8.3.2).

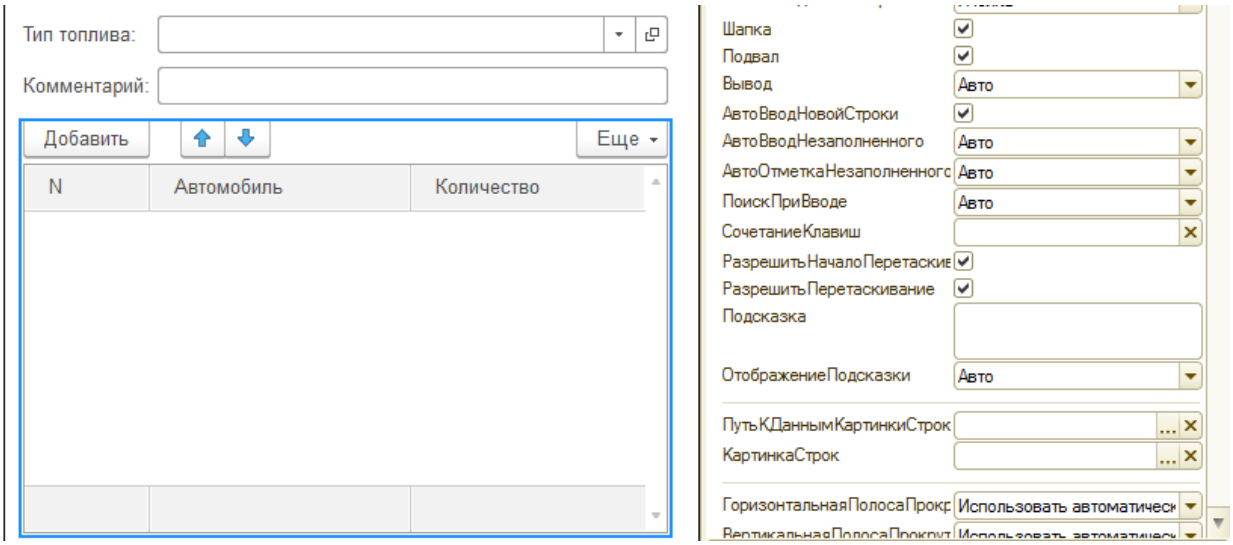


Рис. 8.3.2

Сам по себе подвал не очень интересен, но в нем можно размещать различную информацию. Зайдите в палитру свойств поля «Автомобиль» таблицы формы. В этой палитре нас интересуют два свойства - *ТекстПодвала* и *ПутьКДаннымПодвала* (см. рис. 8.3.3).

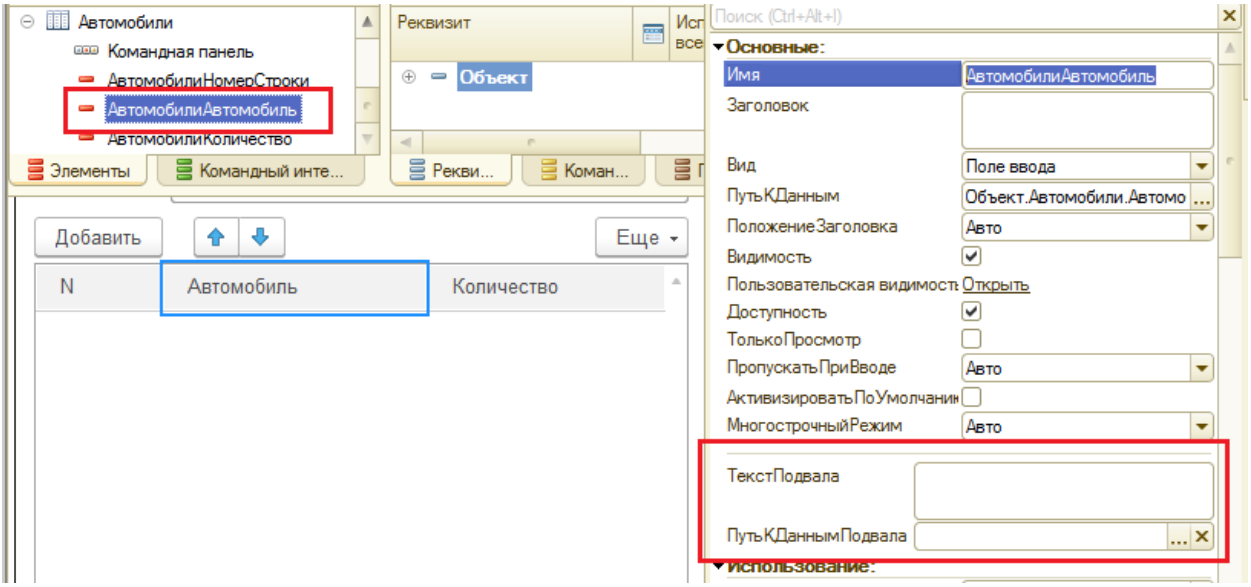


Рис. 8.3.3

В свойстве «ТекстПодвала» можно размещать какой-либо статический текст. А в свойстве *ПутьДаннымПодвала* можно указывать различные динамические данные (например, сумму по всем строкам). Для демонстрации этих возможностей сделаем небольшую задачу: на форме документа *Отчет о расходе топлива* в подвале колонки *Автомобиль* должен быть текст «Итог по всем автомобилям», а в подвале колонки *Количество* общая сумма по всем строкам.

С первым требованием проблем у Вас возникнуть не должно: просто напишем нужный текст в свойстве *ТекстПодвала* поля таблицы *Автомобиль* (см. рис. 8.3.4).

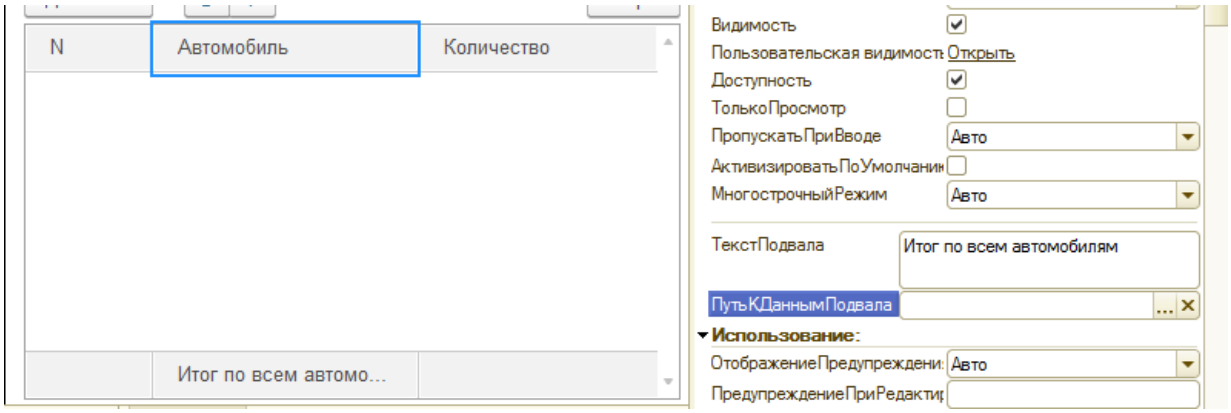


Рис. 8.3.4

Перейдем к колонке *Количество*. Зайдем в палитру свойств этой колонки и нажмем на кнопку «…» свойства *ПутьКДаннымПодвала*. Откроется окно выбора реквизита (см. рис. 8.3.5).

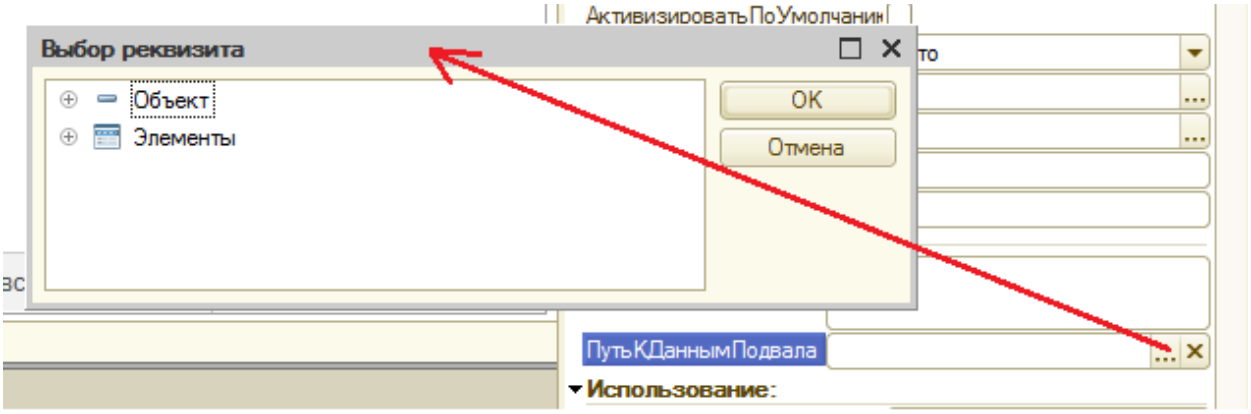


Рис. 8.3.5

В этом окне два узла – *Объект* и *Элементы*. Узел *Объект* - это основной реквизит формы, Вы можете выбрать любое свойство этого реквизита (например, «Дату» или «Комментарий», см. рис. 8.3.6).

В узле *Элементы* можно выбрать данные по текущей строке редактируемой таблицы (см. рис. 8.3.7). Например, можно сделать, чтобы в подвале всегда отображался автомобиль той строки, на которой в данный момент установлен курсор.

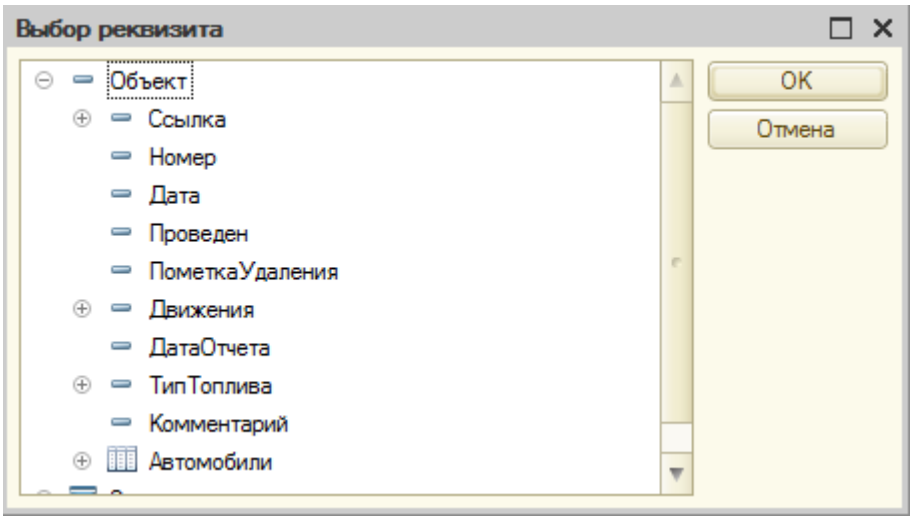


Рис. 8.3.6

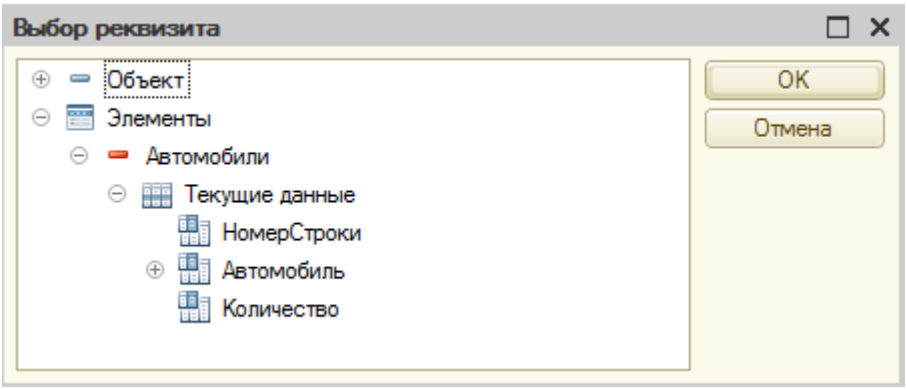


Рис. 8.3.7

Вернемся обратно к объекту и развернем узел *Автомобили*. В этом узле нас интересует пункт *ИтогКоличество*.

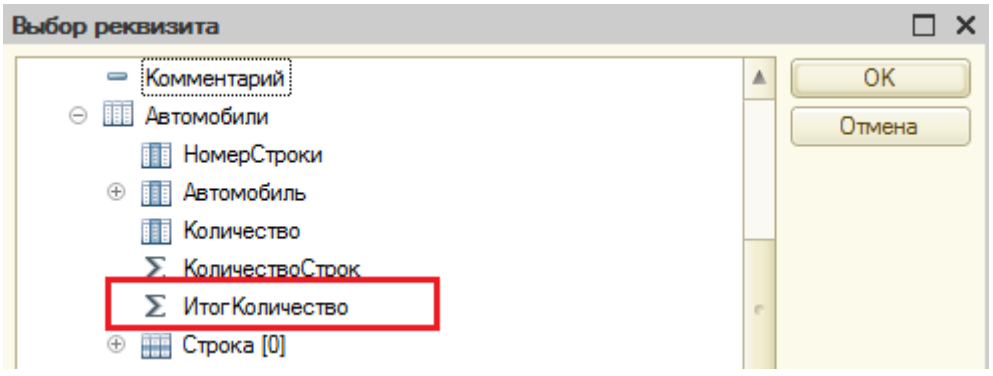


Рис. 8.3.8

У таблиц табличных частей справочников и документов у нас есть возможность получать итоги по всем реквизитом, у которых тип «Число».

Выберем этот пункт в свойство *ПутьКДаннымПодвала* (см. рис. 8.3.9).

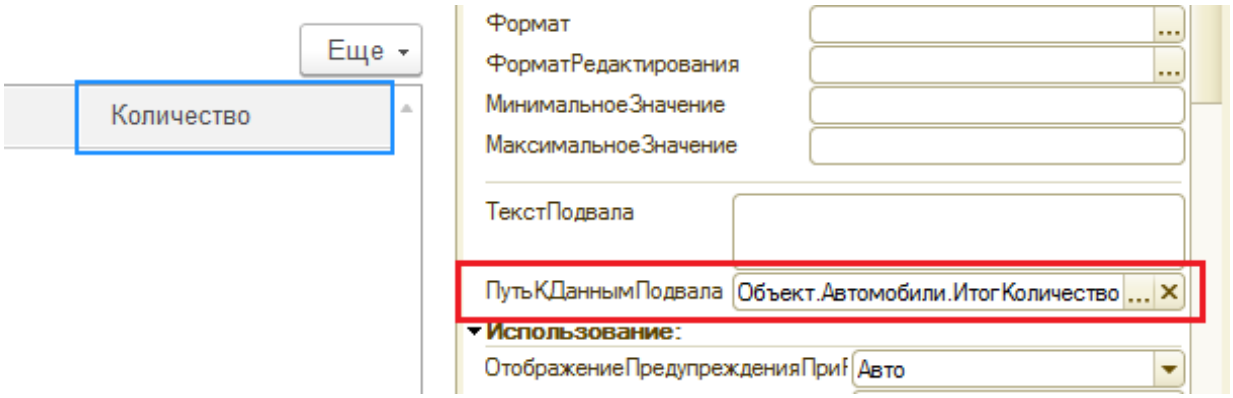


Рис. 8.3.9

Сохраним конфигурацию, обновим базу данных и откроем форму документа *Отчет о расходе топлива*, чтобы посмотреть, как заполняется подвал формы (см. рис. 8.3.10).

N	Автомобиль	Количество
1	Автомобиль главного директора	3,00
2	Дежурный автомобиль	5,00
	Итог по всем автомобилям	8,00

Рис. 8.3.10

А для табличной части документа *Установка цен на топливо* сделайте подвал, как на рис. 8.3.11.

N	Тип топлива	Цена
1	Аи 92	13,00
2	Аи 95	15,00
	Аи 95	15,00

Рис. 8.3.11

Текущий элемент

Очень часто может возникнуть необходимость совершить какие-либо операции над текущей строкой таблицы, или получить информацию о текущей строке. Для реализации подобных задач необходимо работать со свойствами *ТекущиеДанные* и *ТекущаяСтрока* таблиц.

Свойство *ТекущиеДанные* имеет тип *ДанныеФормыЭлементКоллекции*, и посредством этого свойства можно получить доступ к данным текущей строки. А свойство *ТекущаяСтрока* - это *идентификатор* текущей строки.

На рис. 8.3.12 можно увидеть, какие данные хранятся в обоих свойствах.

⊖ Элементы.ЦеныТоплива.ТекущиеДанные	ДанныеФормыЭлементКоллекции	ДанныеФормыЭлементКоллекции
ИсходныйНомерСтроки	0	Число
НомерСтроки	1	Число
⊕ ТипТоплива	Аи 92	СправочникСсылка.ТипТоплива
Цена	0	Число
Элементы.ЦеныТоплива.ТекущаяСтрока	2	Число

Рис. 8.3.12

Обратите внимание, свойство *ТекущаяСтрока* - это не индекс строки и не её порядковый номер. Как использовать это свойство, мы узнаем ниже.

Разберем простенький пример, как работать с текущими данными таблицы. Создайте обработку, форму обработки. А на форме создайте два реквизита: таблица значений «ТаблицаАвтомобилей» с одной колонкой «Автомобиль» (тип *СправочникСсылка.Автомобили*) и «КоробкаПередач» (тип *ПеречислениеСсылка.КоробкаПередач*). А также команду «Поменять коробку передач». При выполнении этой команды будем менять коробку передач у текущего автомобиля.

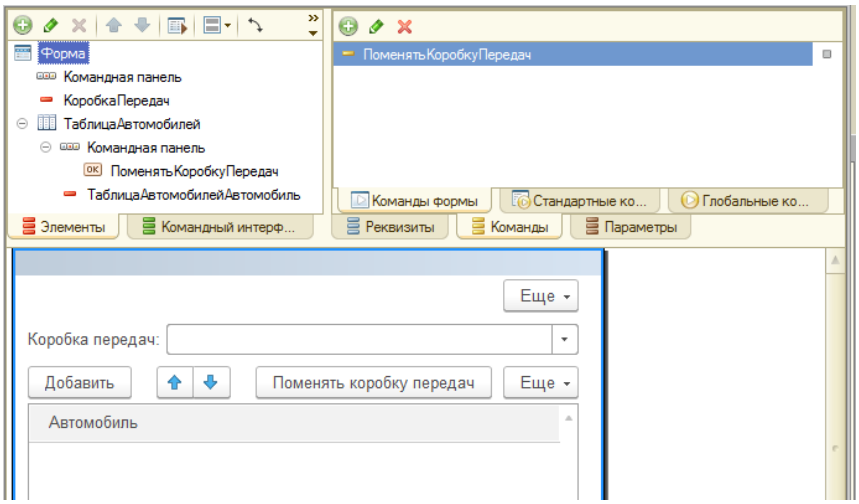


Рис. 8.3.13

Заполним таблицу значений в обработчике формы «ПриСозданииНаСервере» (см. листинг 8.3.1).

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

    Выборка = Справочники.Автомобили.Выбрать();
    Пока Выборка.Следующий() Цикл
        НовСтр = ТаблицаАвтомобилей.Добавить();
        НовСтр.Автомобиль = Выборка.Ссылка;
    КонецЦикла;

КонецПроцедуры
```

Листинг 8.3.1

Создадим обработчик команды «Поменять коробку передач» в клиентском контексте и напишем следующий код.

```
&НаСервере
Процедура ПоменятьКоробкуПередачНаСервере(Автомобиль)

    Если КоробкаПередач.Пустая() Тогда
       Возврат;
    КонецЕсли;
    Если Не ЗначениеЗаполнено(Автомобиль) Тогда
       Возврат;
    КонецЕсли;
    АвтомобильОбъект = Автомобиль.ПолучитьОбъект();
    АвтомобильОбъект.КоробкаПередач = КоробкаПередач;
    Попытка
        АвтомобильОбъект.Записать();
    Исключение
    КонецПопытки;

КонецПроцедуры

&НаКлиенте
Процедура ПоменятьКоробкуПередач(Команда)

    ТекущиеДанные = Элементы.ТаблицаАвтомобилей.ТекущиеДанные;
    Если ТекущиеДанные = Неопределено Тогда
        Возврат;
    КонецЕсли;
    ПоменятьКоробкуПередачНаСервере(ТекущиеДанные.Автомобиль);

КонецПроцедуры
```

Листинг 8.3.2

Вы этот код должны разобрать без проблем.

Перейдем к свойству «ТекущаяСтрока». Для чего нужно это свойство? С его помощью мы можем связывать текущие данные элемента форма с данными, которые хранятся в реквизите формы. Научимся это делать. Для примера решим небольшую задачу: в документе *Установка цен на топливо* создадим команду «Увеличить текущую цену в 2х» и при выполнении команды, будем увеличивать цену текущей строки таблицы в 2 раза.

Создадим команду формы документа *Установка цен на топливо* и разместим её в группе *Заполнить* командной панели таблицы «Цены» (см. рис. 8.3.14).

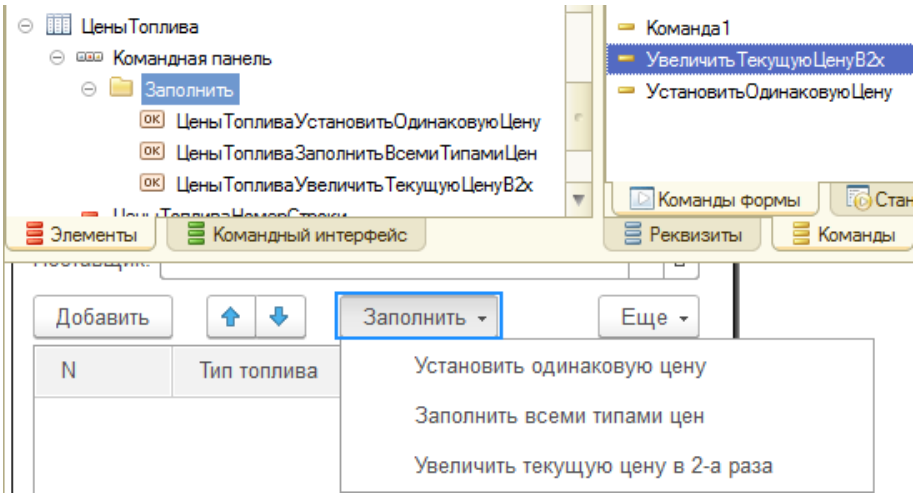


Рис. 8.3.14

Для этой команды создадим обработчик *на клиенте* и при её выполнении будем увеличивать цену текущей строки в 2 раза (см. листинг 8.3.3).

```
&НаКлиенте
Процедура УвеличитьТекущуюЦенуВ2х(Команда)
    Идентификатор = Элементы.ЦеныТоплива.ТекущаяСтрока;
    Если Идентификатор = Неопределено Тогда
        Возврат;
    КонецЕсли;

    ТекущиеДанные = Объект.ЦеныТоплива.НайтиПоИдентификатору(Идентификатор);
    Если ТекущиеДанные = Неопределено Тогда
        Возврат;
    КонецЕсли;
    ТекущиеДанные.Цена = ТекущиеДанные.Цена * 2;
КонецПроцедуры
```

Листинг 8.3.3

Разберем этот код.

В первой строке мы получили свойство таблицы формы *ТекущаяСтрока,* это свойство содержит или идентификатор текущей строки, или значение «Неопределено», если нет текущей строки.

Следующим шагом мы проверяем, не вернуло ли свойство *ТекущаяСтрока* значение *Неопределено*, и если вернуло, то выходим из процедуры.

После мы ищем текущую строку непосредственно в реквизите, т.е. в данных формы. Делаем мы это при помощи метода *НайтиПоИдентификатору* знакомого нам объекта *ДанныеФормыКоллекция*. Данный метод имеет один параметр – это идентификатор строки, которую нужно найти, - и возвращает или значение *Неопределено*, если строка с таким идентификатором в коллекции не найдена, или тип *ДанныеФормыЭлементКоллекции*, посредством которого мы получаем доступ к элементам конкретного реквизита.

Мы на всякий случай проверяем, не вернул ли метод *НайтиПоИдентификатору* значение *Неопределено*.

И в последнем шаге мы рекурсивно изменяем значение одного из свойства элемента коллекции.

Самостоятельно поработайте с текущей строкой. Сделайте команды, которые уменьшают текущую цену в 2 раза, вычитают из неё 1 рубль и т.д.

Резюме

На этом мы закончим изучать работу с таблицами. Мы изучили одну из основных коллекций значений прикладного языка 1С – таблицу значений. Таблицы значений очень часто применяются при разработке тех или иных задач, поэтому данный объект нужно знать и уметь с ним работать. Также нужно понимать, как работает этот объект на управляемой форме. Поскольку таблица значений не может существовать в тонком клиенте, то на форме идет работа с объектом *ДанныеФормыКоллекция*. Вы узнали, как работать с табличными частями метаданных (документов, справочников и т.д.) и как они отображаются на форме (в виде уже знакомого нам объекта *ДанныеФормыКоллекция*). А также освоили некоторые приемы работы с элементом формы *Таблица*: научились делать подвал и заполнять его данными, а также работать с текущей строкой. Все эти знания Вам пригодятся в практической работе.

Глава 9. Получение данных

В этом главе мы научимся получать данные из базы. Что понимается под получением данных из базы?

В дальнейшем под этим термином будем понимать получение любого набора элементов справочников, документов, записей регистров сведений, накопления и регистров бухгалтерии.

В этой главе мы научимся получать выборки данных для документов и справочников. С регистрами накоплений и сведений мы будем разбираться в следующей главе.

Данные можно получать двумя способами – используя методы менеджеров объектов (первая часть главы) и с помощью запросов (вторая часть главы).

Часть 1. Понятие выборки

Начнем с выборки данных.

Выборка - это специализированный способ перебора чего-либо. В языке программирования 1С существует много видов выборки, но в этой главе мы рассмотрим два из них - это *Выборка документов* и *Выборка справочников*.

Начнем с *Выборки справочников*.

Выборка справочников

Выборка справочников - это объект, который представляет собой специализированный способ перебора элементов справочника. *Выборка* является динамическим объектом, то есть она не получает все элементы справочника сразу, а получает их порциями, что позволяет достаточно быстро обходить справочники, состоящие из большого количества элементов. Данный объект возвращается двумя методами менеджера справочника, это: *Выбрать* и *Выбрать иерархически*.

Рассмотрим оба этих метода. И посредством работы с данными методами перейдем к выборке.

Выбрать

Метод *Выбрать* объекта *«Справочник менеджер»* формирует некоторую выборку данных по заданным условиям. Этот метод является функцией и возвращает объект *Выборка справочника*.

Рассмотрим синтаксис данного метода.

Выбрать(<Родитель>, <Владелец>, <Отбор>, <Порядок>)

Все параметры данного метода являются необязательными.

«Родитель» - применим для *иерархических* справочников. Если он указан, то будут выбираться только те элементы справочника, у которых в свойстве *Родитель* стоит указанный элемент. Если параметр не заполнен, то выберутся все элементы справочника. Чтобы выбрать элементы только верхнего уровня, необходимо указать в качестве *Родителя* пустую ссылку.

«Владелец» - необходим, если мы работаем с подчиненным справочником. Когда мы его укажем, то будет осуществлена выборка всех элементов, которые подчинены данному владельцу.

«Отбор». Вот на этом параметре мы остановимся поподробнее. Данный параметр представляет собой структуру, в которой мы можем задать критерий, по которому будет произведена выборка данных. Ключом элемента структуры будет название поля, по которому будет действовать отбор, а значение структуры – непосредственно то значение, по которому должен будет произвестись отбор. В выборках структура должна содержать только один элемент! Т.е. отбор можно осуществить только по одному реквизиту.

Обращаю Ваше внимание, что в качестве полей для отбора могут быть заданы только те поля, у которых признак индексирования установлен либо в *«Индексировать»*, либо в *«Индексировать с дополнительным упорядочиванием»*. Рассмотрим, где этот признак находится. Для этого зайдите в справочник *Автомобили* и откройте свойства элемента *Коробка передач*. Обратите внимание на параметр *Индексирование*.

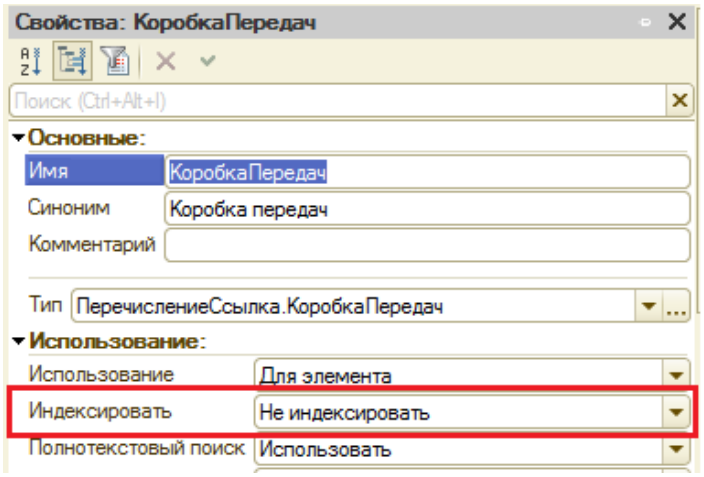


Рис. 9.1.1

У данного поля этот параметр установлен в значение *«Не индексировать»*. Поменяйте его на значение *«Индексировать»*.

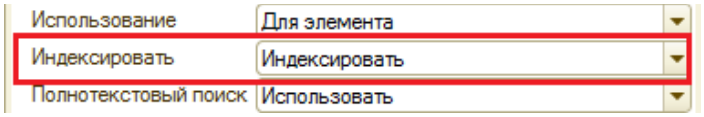


Рис. 9.1.2

Все, теперь мы его сможем использовать в отборах для выборки. Существуют всего два поля, которые могут свободно быть использованы в отборе, это поля *Код* и *Наименование*.

Поэтому прежде чем делать какую-нибудь выборку с отбором, обратите внимание на данный параметр для отбираемого поля. И только после принимайте решение об использовании отбора. Если данное поле не является индексируемым, а Вам все равно необходимо получить выборку с отбором по этому полю, то необходимо использовать язык запросов. О нем мы поговорим во второй части текущей главы.

И последний, тоже интересный параметр, - это *Порядок*, данным параметром является строка с именем реквизита, по которому будет упорядочена выборка. В этом параметре свободно могут быть указаны поля *Наименование* и *Код*, а также поля примитивных типов (*число, строка, дата, булево*), для которых признак *Индексирование* установлен либо в *«Индексировать»*, либо в *«Индексировать с дополнительным упорядочиванием»*. Сортировка данных будет осуществлена по возрастанию, но можно провести и сортировку по убыванию, написав в конце строки слово *«Убыв»*.

Например, так:

"Код Убыв"

Также можно и задать сортировку по возрастанию, написав в конце строки слово *«Возр».*

"Код Возр"

Итак, все параметры мы изучили, теперь осталось применить это на практике. Для этого сначала сделаем простую выборку справочника *Гаражи*, а потом выборку с иерархией. Результаты выборки будем выводить в таблицу значений, которая будет реквизитом формы.

На основе данных примеров Вы научитесь работать с выборками.

Создайте новую обработку, разместите на ее форме реквизит *ТаблицаГаражей* с типом «ТаблицаЗначений». В эту таблицу добавим одну колонку – *Гараж* (тип *ссылка на справочник Гаражи*). Саму таблицу значений разместите на форме в виде элемента *Таблица*. Создайте команду «Заполнить таблицу», которую необходимо разместить в командной панели таблицы на форме.

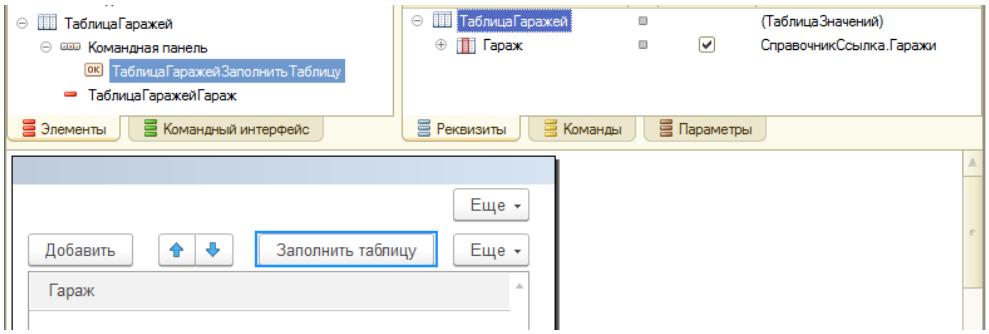


Рис. 9.1.3

Создайте обработчики команды «Заполнить таблицу» и напишите следующий код:

```
&НаСервере
Процедура ЗаполнитьТаблицуНаСервере()
    ВыборкаГаражей = Справочники.Гаражи.Выбрать();
    Пока ВыборкаГаражей.Следующий() Цикл
         НоваяСтрока = ТаблицаГаражей.Добавить();
         НоваяСтрока.Гараж = ВыборкаГаражей.Ссылка;
    КонецЦикла;
КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьТаблицу(Команда)
    ЗаполнитьТаблицуНаСервере();
КонецПроцедуры
```

Листинг 9.1.1

Разберем код выше. В первой строке процедуры *ЗаполнитьТаблицуНаСервере* мы получаем выборку элементов справочника *Гаражи*.

Метод, использующийся во второй строке, Вам ещё незнаком. Это метод *Следующий*.

Данный метод получает следующий элемент выборки. Дело в том, что любая выборка представляет собой совокупность элементов, которые были выбраны.

Обращаться к этим элементам можно непосредственно через ту переменную, в которую данная выборка записана. В нашем случае это переменная *ВыборкаГаражей*. Но, когда мы ее только получили с помощью метода *Выбрать*, она еще не позиционирована ни на каком элементе. Поэтому для того, чтобы пройтись по всем элементам выборки, нам понадобится метод *Следующий*, с помощью которого осуществляется переход на следующий элемент выборки. Данный метод является функцией и возвращает значение *Истина,* если элемент выбран, и *Ложь* - в том случае, когда *достигнут конец выборки*. Если нам необходим только *первый* элемент выборки, достаточно написать вот так:

ВыборкаГаражей = Справочники.Гаражи.Выбрать();

ВыборкаГаражей.Следующий() ;

После того, как элемент выборки выбран, то можно обращаться к нему как к элементу справочника. Поэтому мы получаем доступ ко всем реквизитам справочника и можем их использовать в дальнейшей работе. Сама переменная *ВыборкаГаражей* имеет тип *СправочникВыборка.Гаражи.*

Поэтому в данном примере мы получили ссылку на элемент справочника, обратившись к выборке.

Если бы нам надо было получить наименование элемента справочника или код, то мы тоже легко смогли бы это сделать:

Наименование = ВыборкаГаражей.Наименование;

Код = ВыборкаГаражей.Код;

Если Вы уже самостоятельно прописывали вышеприведенный код, то заметили, что когда мы пишем точку после переменной *ВыборкаГаражей*, то возникает выпадающий список, в котором перечислены все реквизиты справочника.

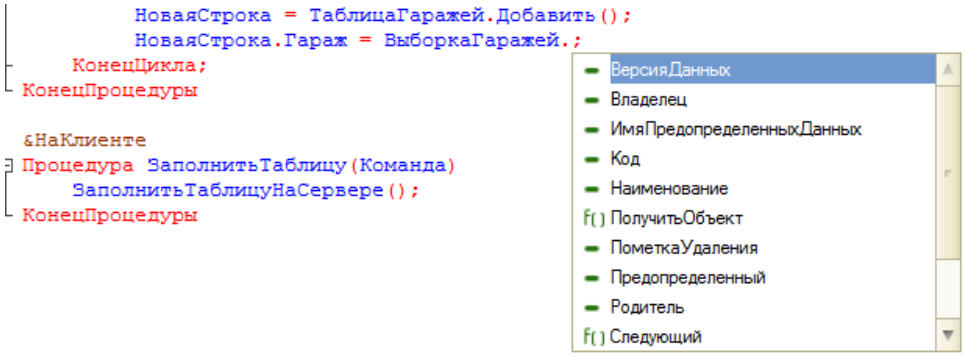


Рис. 9.1.4

Важно! Получайте реквизит именно через выборку, а не через ссылку на объект. Так Вы существенно оптимизируете свой код!

Правильно: ВыборкаГаражей.Наименование.

Неправильно: ВыборкаГаражей.Ссылка.Наименование.

У выборки справочника есть еще один метод - *ПолучитьОбъект*. С помощью данного метода мы получаем объект элемента справочника.

ВыборкаСкладов.ПолучитьОбъект();

Этот метод возвращает объект справочника или документ. С данным методом Вам все должно быть понятно, и каких-либо дополнительных пояснений не нужно.

Теперь, чтобы разобрать более подробно метод *Выбрать,* доработайте Ваш справочник складов. Добавьте в него два реквизита, это реквизит *ТолькоЛегковыеАвтомобили* (тип *булево*, неиндексируемый) и реквизит *ВремяНачалаРаботыГаража* (тип *дата*, состав даты - время, индексируемый).

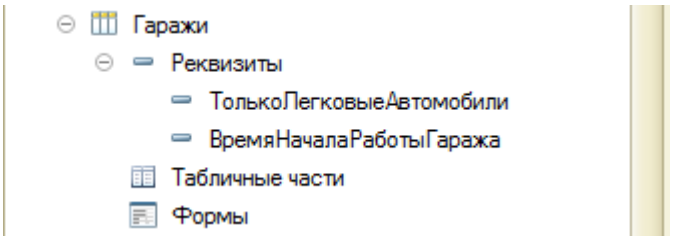


Рис. 9.1.5

И переделайте форму созданной обработки: добавьте в таблицу значений новые колонки: *Время открытия* и *Только легковые автомобили*. Разместите их в таблице на форме. У колонки *Только легковые автомобили* сделаем элемент управления – поле флажка (см. рис. 9.1.6).

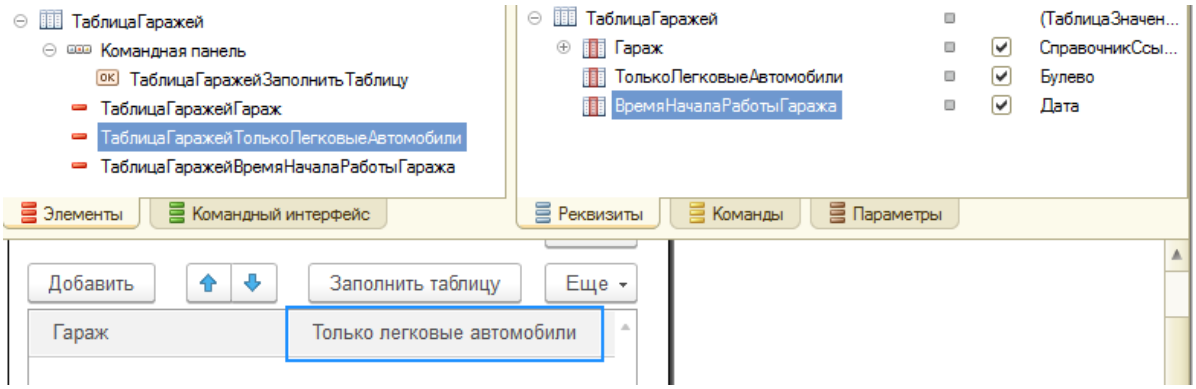


Рис. 9.1.6

Доработаем код из предыдущего листинга - допишем в конце цикла следующие строки:

```
НоваяСтрока.ВремяНачалаРаботыГаража = ВыборкаГаражей.ВремяНачалаРаботыГаража;
НоваяСтрока.ТолькоЛегковыеАвтомобили =
ВыборкаГаражей.ТолькоЛегковыеАвтомобили;
```

Листинг 9.1.2

Создайте самостоятельно разветвленную структуру складов (с группами и элементами). Как, например, на рис. 9.1.7.

Наименование	Код	Только ле...	Время начала работы гаража
Гаражи			
Автобаза №1	001		
Участок 1	010		
Гараж 1-1-1	011		9:00:00
Гараж 1-1-2	012		9:00:00
Гараж №1-1	002	✓	8:00:00
Гараж №1-2	003		10:00:00
Автобаза №2	005		
Гараж №2-1	006		7:00:00
Гараж №2-2	007		6:00:00
Офисная стоянка "Северная"	009	✓	6:00:00
Офисная стоянка "Центральная"	008	✓	7:00:00

Рис. 9.1.7

Если мы сейчас запустим обработку, то получится следующий результат (см. рис. 9.1.8).

Гараж	Только легковые автомобили	Время начала работы гаража
Автобаза №1	☐	
Автобаза №2	☐	
Гараж 1-1-1	☐	9:00:00
Гараж 1-1-2	☐	9:00:00
Гараж №1-1	☑	8:00:00
Гараж №1-2	☐	10:00:00
Гараж №2-1	☐	7:00:00
Гараж №2-2	☐	6:00:00
Офисная стоянка "Северная"	☑	6:00:00
Офисная стоянка "Центральная"	☑	7:00:00
Участок 1	☐	

Рис. 9.1.8

Усложним нашу работу с выборкой. Сначала научимся выбирать элементы только верхнего уровня, а потом элементы одной конкретной группы.

Выберем элементы только первого уровня. Для этого в первом параметре метода *Выбрать* напишем пустую ссылку на элемент справочника.

```
ВыборкаГаражей =
Справочники.Гаражи.Выбрать(Справочники.Гаражи.ПустаяСсылка());
Пока ВыборкаГаражей.Следующий() Цикл
     //обход выборки
КонецЦикла;
```

Листинг 9.1.3

Сохраните обработку и запустите. Как Вы увидели, выведены все элементы верхнего уровня, включая группы. К сожалению, мы не можем отобрать с помощью выборки только элементы справочника, поэтому если Вам не нужно появление групп в табличном поле, то можно дописать следующий код:

```
ВыборкаГаражей =
Справочники.Гаражи.Выбрать(Справочники.Гаражи.ПустаяСсылка());
Пока ВыборкаГаражей.Следующий() Цикл
    Если ВыборкаГаражей.ЭтоГруппа Тогда
        Продолжить;
    КонецЕсли;
    //обход выборки
КонецЦикла;
```

Листинг 9.1.4

Сохраните обработку, перезапустите и посмотрите, какая получилась выборка в этот раз.

Как Вы должны увидеть, вышли только элементы справочника. Теперь научимся делать выборки для конкретного родителя. Для этого разместим на форме реквизит *Гараж* с типом «СправочникСсылка.Гаражи», разместите его на форме, а у поля выбора установите для свойства *ВыборГруппИЭлементов* (категория *Использование*) значение *Групп* (см. рис. 9.1.9).

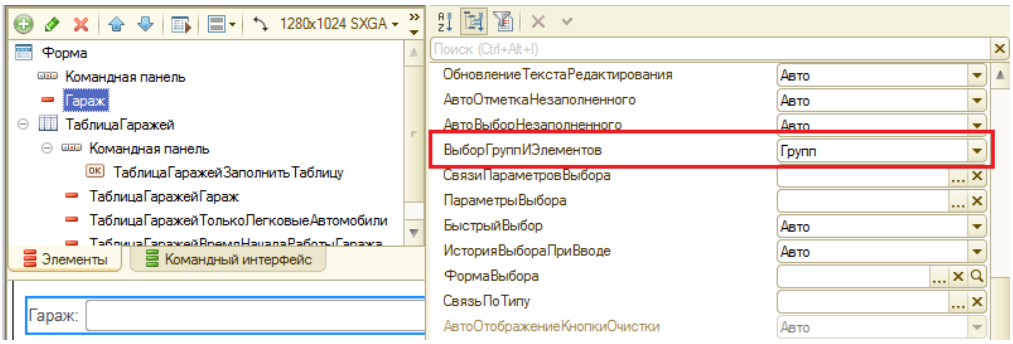


Рис. 9.1.9

Создайте новую команду «Заполнить таблицу по группе». Для удобства я все команды будут размещать в подменю «Заполнить» (см. рис. 9.1.10).

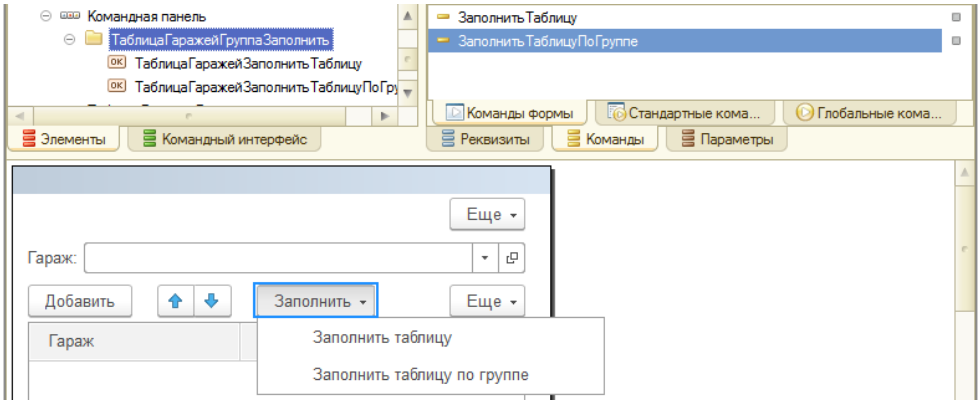


Рис. 9.1.10

В обработчиках команды «Заполнить таблицу по группе» напишите следующий код:

```
&НаСервере
Процедура ЗаполнитьТаблицуПоГруппеНаСервере()
    ВыборкаГаражей = Справочники.Гаражи.Выбрать(Гараж);//выборка элементов по
группе
    Пока ВыборкаГаражей.Следующий() Цикл
        Если ВыборкаГаражей.ЭтоГруппа Тогда
            Продолжить;
        КонецЕсли;
        НоваяСтрока = ТаблицаГаражей.Добавить();
        НоваяСтрока.Гараж = ВыборкаГаражей.Ссылка;
        НоваяСтрока.ВремяНачалаРаботыГаража =
        ВыборкаГаражей.ВремяНачалаРаботыГаража;
        НоваяСтрока.ТолькоЛегковыеАвтомобили =
        ВыборкаГаражей.ТолькоЛегковыеАвтомобили;
    КонецЦикла;
КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьТаблицуПоГруппе(Команда)
    ЗаполнитьТаблицуПоГруппеНаСервере();
КонецПроцедуры
```

Листинг 9.1.5

Посмотрите, как будет работать в этот раз Ваша обработка.

Вы должны были заметить, что если Вы выбираете группу, у которой внутри есть еще одна группа, то будут выведены элементы, которые входят в группу выбора. Но элементы, входящие в подчиненную группу, не вышли (см. рис. 9.1.11).

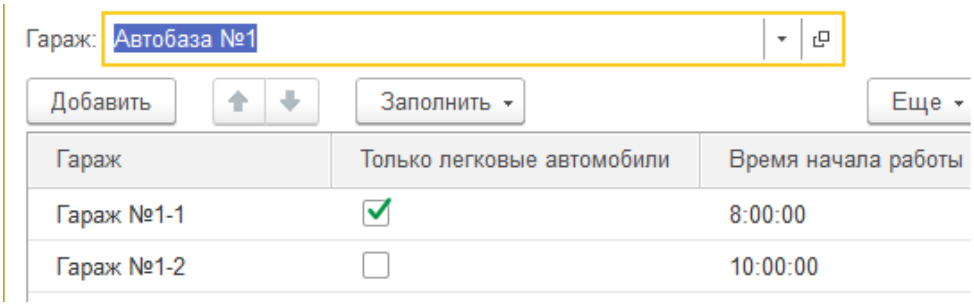


Рис. 9.1.11

Для того чтобы получить все элементы, включая элементы подчиненных справочников, необходимо использовать метод менеджера справочника *ВыбратьИерархически*. Синтаксис данного метода точно такой же, как и у метода *Выбрать*, поэтому мы не будем его разбирать, а просто продемонстрируем, как он работает. Создайте новую команду «Заполнить таблицу по группе с иерархией», в которой будет точно такой же код, как и в команде «Заполнить таблицу по группе», только метод *Выбрать* заменим на метод *ВыбратьИерархически* (см. листинг 9.1.6).

```
ВыборкаГаражей = Справочники.Гаражи.ВыбратьИерархически(Гараж);
Пока ВыборкаГаражей.Следующий() Цикл
    Если ВыборкаГаражей.ЭтоГруппа Тогда
        Продолжить;
    КонецЕсли;
    //обход по выборке
КонецЦикла;
```

Листинг 9.1.6

В результате выполнения команды должен получиться результат как на рис. 9.1.12.

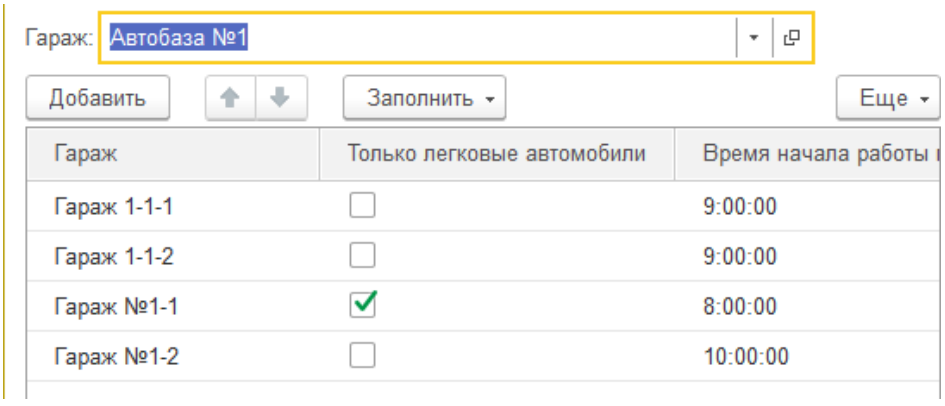


Рис. 9.1.12

Как видите, данный метод очень удобно использовать, когда работаем с иерархическими справочниками, и необходимо получить все элементы данного справочника.

Обращаю Ваше внимание, что сначала будут выбраны группы самого верхнего уровня, потом группы следующего уровня и так далее. И только после этого будут выбраны элементы, начиная с самого нижнего уровня и заканчивая нулевым уровнем. Чтобы убедиться в этом, закомментируйте код, в котором мы отказываемся выводить группы, и посмотрим, что получится, когда в поле ввода нет элемента справочника. Обратите внимание на порядок, в котором вышли строки в таблице значений.

В дальнейшем применительно к справочнику *Гаражи*, будем работать с методом *ВыбратьИерархически*.

Теперь научимся работать с отбором. Создадим отбор гаражей по *Времени начала работы гаража*. Мы сделали данный реквизит справочника индексируемым, поэтому он должен отбираться как надо.

Поскольку нам необходимо сделать отбор по времени, добавьте на форму реквизит «ВремяРаботы» (Тип *Дата*, состав даты - *Время*), и элемент поле ввода *Время* (см. рис. 9.1.13).

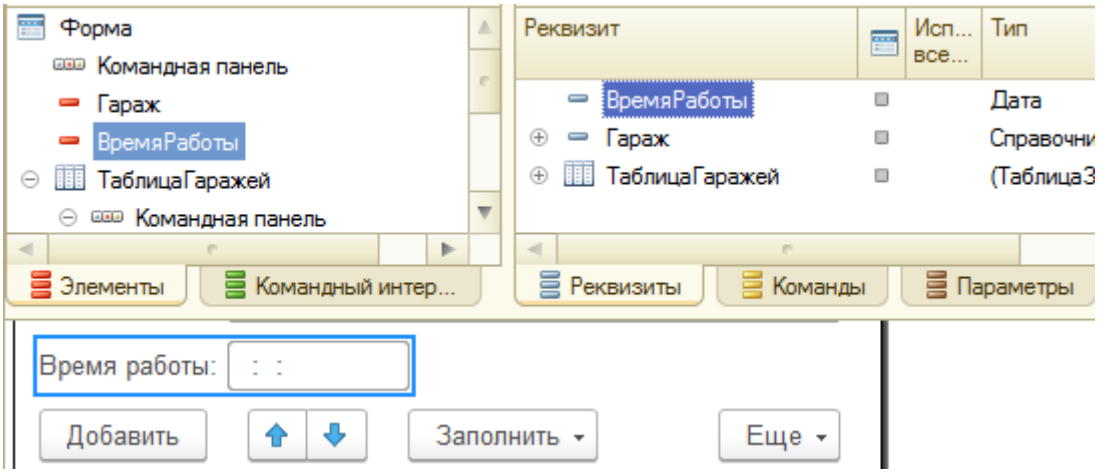


Рис. 9.1.13

Создайте новую команду «Заполнить таблицу с учетом времени», обработчики команды на сервер и на клиенте. В обработчиках напишем следующий код (см. листинг 9.1.7):

```
 &НаСервере
Процедура ЗаполнитьТаблицуСУчетомВремениНаСервере()
    Если  Не ЗначениеЗаполнено(ВремяРаботы) тогда
        ВыборкаГаражей = Справочники.Гаражи.ВыбратьИерархически(Гараж);
    иначе
        Отбор = Новый Структура;
        Отбор.Вставить("ВремяНачалаРаботыГаража",ВремяРаботы);
        ВыборкаГаражей =
Справочники.Гаражи.ВыбратьИерархически(Гараж,,Отбор);
    КонецЕсли;
    Пока ВыборкаГаражей.Следующий() Цикл
        Если ВыборкаГаражей.ЭтоГруппа Тогда
            Продолжить;
        КонецЕсли;
        //обход по выборке
    КонецЦикла;
КонецПроцедуры
```

```
&НаКлиенте
Процедура ЗаполнитьТаблицуСУчетомВремени(Команда)
    ЗаполнитьТаблицуСУчетомВремениНаСервере();
КонецПроцедуры
```

Листинг 9.1.7

Обращаю Ваше внимание, что выборка осуществляется для всех объектов, включая группы, а Вы уже знаете, что сначала выводится группа, а потом элементы, которые входят в группу. Но отбор тоже применяется для всех элементов, включая группы, поэтому если мы будем на верхнем уровне и зададим отбор *10* часов, то ничего не выйдет. Почему? Потому что у нас не отберутся группы первого уровня, для них не установлен данный реквизит, следовательно, не выведутся и элементы, которые входят в них. Поэтому список будет пустой. В то же время для групп нижнего уровня отбор сработает. Проверьте сами работу данного кода.

Теперь попробуйте сделать отбор для неиндексируемого поля, и посмотрите, что получится.

Измените код:

```
Отбор = Новый Структура;
Отбор.Вставить("ТолькоЛегковыеАвтомобили",Истина);
ВыборкаГаражей = Справочники.Гаражи.ВыбратьИерархически(Гараж,,Отбор);

Пока ВыборкаГаражей.Следующий() Цикл
    Если ВыборкаГаражей.ЭтоГруппа Тогда
        Продолжить;
    КонецЕсли;
    //обход выборки
КонецЦикла;
```

Листинг 9.1.8

Запустите обработку и посмотрите, что должно возникнуть. Как видите, возникла ошибка.

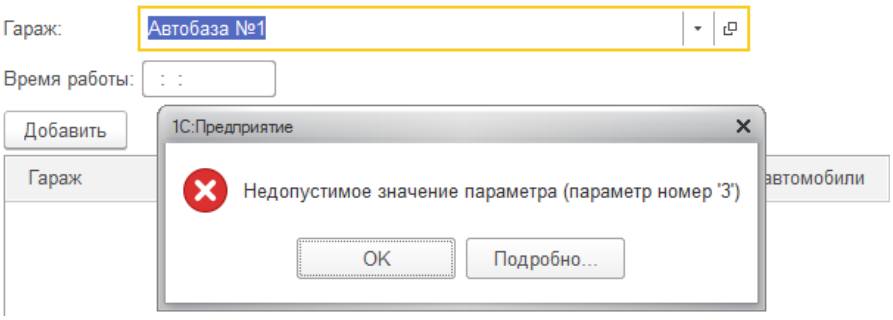


Рис. 9.1.14

Таким образом, еще раз повторюсь, отбор работает только с **индексированными** реквизитами.

Теперь рассмотрим последний параметр выборок - это *Порядок*. Выведем все наши элементы, упорядочив их по *Времени начала работы гаража*. Для этих целей создадим команду

«Заполнить таблицу с упорядочиванием», которую разместим на форме. В обработчиках команды напишем следующий код:

```
&НаСервере
Процедура ЗаполнитьТаблицуСУпорядочиваниемНаСервере()
    ВыборкаГаражей =
Справочники.Гаражи.ВыбратьИерархически(Гараж,,,"ВремяНачалаРаботыГаража");
    Пока ВыборкаГаражей.Следующий() Цикл
        Если ВыборкаГаражей.ЭтоГруппа Тогда
            Продолжить;
        КонецЕсли;
        //обход по выборке
    КонецЦикла;
КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьТаблицуСУпорядочиванием(Команда)
    ЗаполнитьТаблицуСУпорядочиваниемНаСервере();
КонецПроцедуры
```

Листинг 9.1.9

Поскольку параметр *Порядок*, так же, как и параметр *Отбор,* действует на *все* элементы справочника, включая группы, то для наглядности измените время нескольких элементов в одной группе на разное и выведете элементы только этой группы. Если Вам необходимо упорядочивание по убыванию, то изменим написание последнего параметра.

```
ВыборкаГаражей =
Справочники.Гаражи.ВыбратьИерархически(Гараж,,,"ВремяНачалаРаботыГаража
УБЫВ");
```

Листинг 9.1.10

Сохраните обработку, перезапустите, и Вы увидите, что элементы упорядочились по убыванию.

Итак, мы рассмотрели все параметры кроме второго – *Владелец*. Создадим новую обработку. В этой обработке будет два реквизита *МаркаАвтомобиля* (тип *СправочникСсылка.МаркиАвтомобилей*) и *МодельАвтомобиля* (тип *ТаблицаЗначений*, единственная колонка *Модель* с соответствующим типом), первый будет размещен на форме в виде поля ввода, а второй – таблицы (см. рис. 9.1.15).

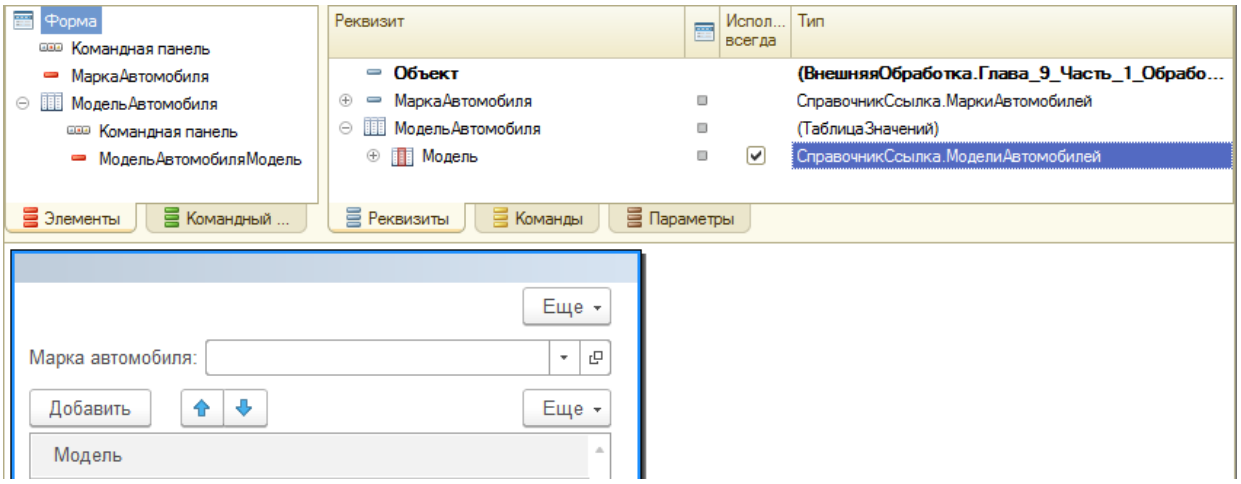


Рис. 9.1.15

В обработчиках события *ПриИзменении* поля ввода *МаркаАвтомобиля* напишем следующий код:

```
&НаСервере
Процедура МаркаАвтомобиляПриИзмененииНаСервере()
    Выборка = Справочники.МоделиАвтомобилей.Выбрать(,МаркаАвтомобиля);
    МодельАвтомобиля.Очистить();
    Пока Выборка.Следующий() Цикл
        НовСтр = МодельАвтомобиля.Добавить();
        НовСтр.Модель = Выборка.Ссылка;
    КонецЦикла;
КонецПроцедуры

&НаКлиенте
Процедура МаркаАвтомобиляПриИзменении(Элемент)
    МаркаАвтомобиляПриИзмененииНаСервере();
КонецПроцедуры
```

Листинг 9.1.11

Сохраните обработку и посмотрите на результат: после выбора определенной марки в таблице на форме должен появиться список моделей этой марки (см. рис. 9.1.16).

Рис. 9.1.16

На этом мы закончим изучать выборку справочников и перейдем к изучению выборок документов.

Выборка документов

Выборка документов, как и в случае выборки справочника, - это объект, который представляет специализированный способ перебора документов. Объект *Выборка документов* возвращается методом *Выбрать* менеджера документа.

Метод *Выбрать* имеет следующий синтаксис:

Выбрать(<ДатаНачала>, <ДатаОкончания>, <Отбор>, <Порядок>)

Все параметры являются необязательными. Рассмотрим параметры данного метода.

ДатаНачала и *ДатаОкончания* - это параметры, которые определяют период выбираемых документов. Если они не указаны, то выбираются все документы. Если указана только одна дата (*ДатаНачала* либо *ДатаОкончания*), то выбираются документы с этой даты или до этой даты.

Второй и третий параметр (*Отбор* и *Порядок*) Вам уже знакомы по выборке справочников, у них точно такой же смысл, они работают только в отношении реквизита *Дата* и индексированных реквизитов. Точно так же нужно указывать только один реквизит.

У выборки документов всего два метода, это: *ПолучитьОбъект*, который возвращает объект выбранного документа, и метод *Следующий*, который осуществляет переход на следующий элемент выборки.

Рассмотрим пример. Для большей наглядности создайте несколько документов *Прибытие в гараж* с разными датами. А также сделайте реквизит *Автомобиль* документа *индексируемым*.

В новой обработке создайте реквизит «ТаблицаДокументов» с типом *ТаблицаЗначений* (колонкам: *Номер*(тип «Строка»), *ДатаПрибытия* (тип «Дата»), *Автомобиль* (тип «СправочникСсылка.Автомобили») и *ПрибытиеАвтомобиля* (тип «ДокументСсылка.ПрибытиеВГараж»)). А также создайте команду «Заполнить таблицу», в серверном обработчике этой команды будем заполнять данную таблицу.

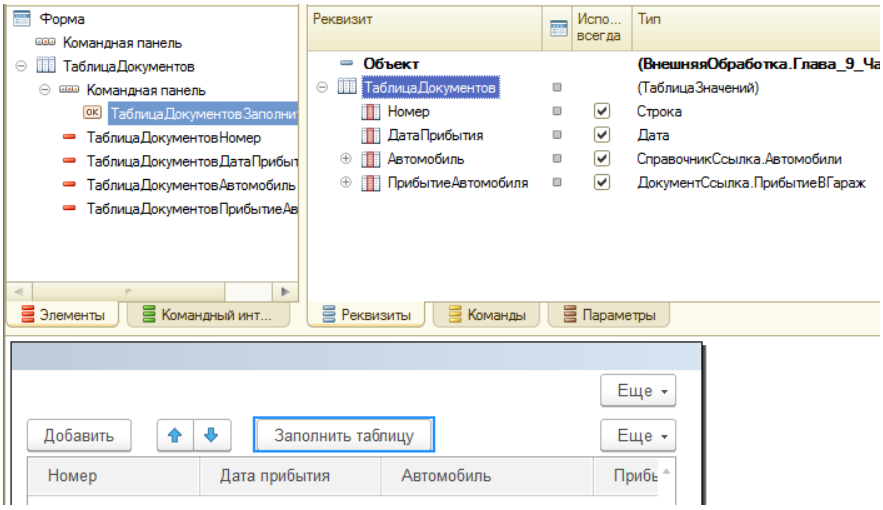


Рис. 9.1.17

```
&НаСервере
Процедура ЗаполнитьТаблицуНаСервере()
    ТаблицаДокументов.Очистить();
    Выборка = Документы.ПрибытиеВГараж.Выбрать();
    Пока Выборка.Следующий() Цикл
        НовСтр = ТаблицаДокументов.Добавить();
        НовСтр.Номер = Выборка.Номер;
        НовСтр.ДатаПрибытия = Выборка.ДатаПрибытия;
        НовСтр.Автомобиль = Выборка.Автомобиль;
        НовСтр.ПрибытиеАвтомобиля = Выборка.Ссылка;
    КонецЦикла;
КонецПроцедуры


&НаКлиенте
Процедура ЗаполнитьТаблицу(Команда)
    ЗаполнитьТаблицуНаСервере();
КонецПроцедуры
```

Листинг 9.1.12

В этом коде мы получаем выборку документов *Прибытие в гараж* и осуществляем обход элементов выборки. Во время обхода создаем новую строку табличного поля, в которую заносим соответствующие данные.

Посмотрите, как работает Ваша обработка.

Мы рассмотрели самый простой метод выборки без параметров. Теперь рассмотрим использование дат начала и конца. Для этого создайте на форме реквизит с новым для Вас типом *СтандартныйПериод* (см. рис. 9.1.18).

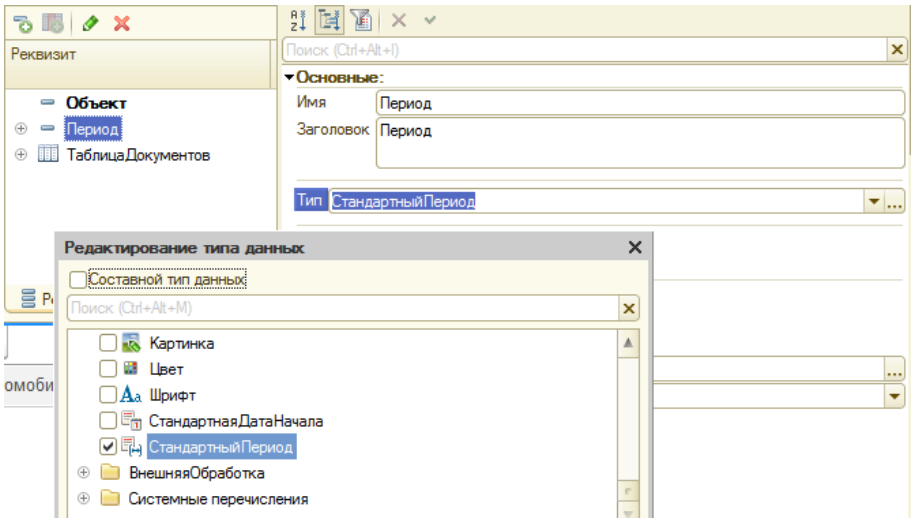


Рис. 9.1.18

Переменная данного типа имеет три реквизита. Нас интересуют *ДатаНачала* и *ДатаОконачания*, в которых, соответственно, содержится начало и конец указанного периода.

Создайте команду *ЗаполнитьТаблицуСПериодом*, разместите на форме и в серверном обработчике команды напишите следующий код:

```
&НаСервере
Процедура ЗаполнитьТаблицуСПериодомНаСервере()

    ТаблицаДокументов.Очистить();
    Выборка =
Документы.ПрибытиеВГараж.Выбрать(Период.ДатаНачала,Период.ДатаОкончания);
    Пока Выборка.Следующий() Цикл
        НовСтр = ТаблицаДокументов.Добавить();
        ЗаполнитьЗначенияСвойств(НовСтр,Выборка);
        НовСтр.ПрибытиеАвтомобиля = Выборка.Ссылка;
    КонецЦикла;

КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьТаблицуСПериодом(Команда)
    ЗаполнитьТаблицуСПериодомНаСервере();
КонецПроцедуры
```

Листинг 9.1.13

Посмотрите, как отбираются документы по датам. Как Вы должны увидеть, выводятся документы только из заданного диапазона дат.

Теперь создадим отбор по реквизиту *Автомобиль* документа. Для этого разместите на форме реквизит под названием *Автомобиль* (тип «СправочникСсылка.Автомобили»).

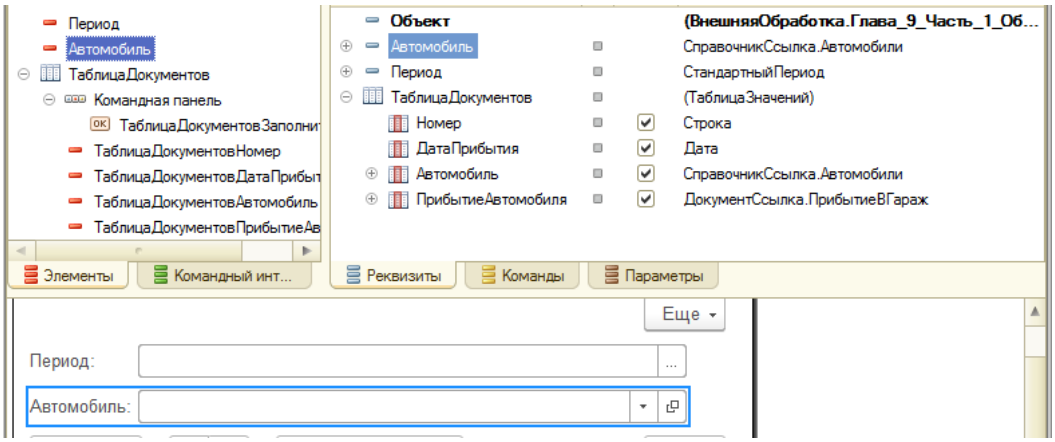


Рис. 9.1.19

Создайте новую команду «Заполнить таблицу с отбором», скопируйте код из предыдущего листинга, в котором осуществляется заполнение таблицы. И доработайте этот код следующим образом.

Если ссылка на элемент справочника *Автомобили* не пустая (он выбран), то мы создаем новую структуру, название ключа которой совпадает с названием реквизита, по которому должен быть осуществлен отбор, а в значение этого элемента запишем элемент справочника, который мы выберем. Используем данную структуру в качестве третьего параметра метода *Выбрать*.

```
&НаСервере
Процедура ЗаполнитьТаблицуСОтборомНаСервере()
    ТаблицаДокументов.Очистить();
    Если Автомобиль.Пустая() Тогда
```

```
        Выборка =
Документы.ПрибытиеВГараж.Выбрать(Период.ДатаНачала,Период.ДатаОкончания);
    иначе
        Отбор = Новый Структура("Автомобиль",Автомобиль);
        Выборка = Документы.ПрибытиеВГараж.Выбрать(Период.ДатаНачала,
                                                   Период.ДатаОкончания,
                                                   Отбор);
    КонецЕсли;
    Пока Выборка.Следующий() Цикл
        НовСтр = ТаблицаДокументов.Добавить();
        ЗаполнитьЗначенияСвойств(НовСтр,Выборка);
        НовСтр.ПрибытиеАвтомобиля = Выборка.Ссылка;
    КонецЦикла;
КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьТаблицуСОтбором(Команда)
    ЗаполнитьТаблицуСОтборомНаСервере();
КонецПроцедуры
```

Листинг 9.1.14

Посмотрите, как работает Ваша обработка.

Напоследок рассмотрим работу последнего параметра метода *Выбрать*, это параметр *Порядок*. В данном случае мы отсортируем наши документы по *Дате*.

Измените код в последнем листинге следующим образом.

```
Если Автомобиль.Пустая() Тогда
    Выборка = Документы.ПрибытиеВГараж.Выбрать(Период.ДатаНачала,
                                               Период.ДатаОкончания,,
                                               "Дата УБЫВ");
иначе
    Отбор = Новый Структура("Автомобиль",Автомобиль);
    Выборка = Документы.ПрибытиеВГараж.Выбрать(Период.ДатаНачала,
                                               Период.ДатаОкончания,
                                               Отбор,
                                               "Дата УБЫВ");
КонецЕсли;
```

Листинг 9.1.15

Мы добавили *Дату* в порядок в обоих методах выбора документов.

Посмотрите на результат.

Документы должны быть отсортированы по *Дате* в обратном порядке.

Резюме

В этой части мы научились выбирать документы и справочники с помощью методов менеджеров. Как Вы в дальнейшем поймете, если необходима сложная выборка, например,

когда условие ставится по нескольким полям, то данные методы неприменимы, для этого необходим язык запросов. Но все равно про них не нужно забывать, и если необходимо осуществить простую выборку большого числа элементов или документов, то мысль о выборках - это первая мысль, которая должна прийти Вам в голову.

Часть 2. Работа с запросами

В этой части мы научимся работать с *Запросами*. Что такое *Запрос* в понятии 1С?

Запрос - это один из механизмов доступа к данным. С помощью запроса разработчик может получать быстрый доступ к данным, он их может читать и анализировать. Но в отличие от других языков запросов, нельзя изменять данные (редактировать и удалять), то есть запросы в платформе 1С предназначены только для быстрого получения и обработки некоторой выборки из больших объемов данных. В этой книге работа с запросами дана в самом общем виде, чтобы у Вас имелось представление об этом механизме платформы 1С. Подробно запросы 1С изучаются в моем курсе «Запросы в 1С для начинающих».

Это теоретическая часть, на этом мы ее закончим и перейдем непосредственно к практике.

Научимся создавать свои собственные запросы.

Получение данных из запроса

Работа с запросами осуществляется через объект *Запрос*. Создается данный объект с помощью оператора *Новый*.

```
Запрос = Новый Запрос;
```

Теперь поставим небольшую задачу: выведем с помощью запроса все элементы справочника *Автомобили*, в таблицу значений. Раньше мы это делали с помощью выборки, а теперь проделаем с помощью запроса.

Самостоятельно создайте обработку, форму обработки, создайте на форме реквизит *ТаблицаАвтомобилей* (тип *ТаблицаЗначений*, колонки соответствуют реквизитам справочника *Автомобиль*, см. рис. 9.2.1) и разместите его в виде таблицы на форме.

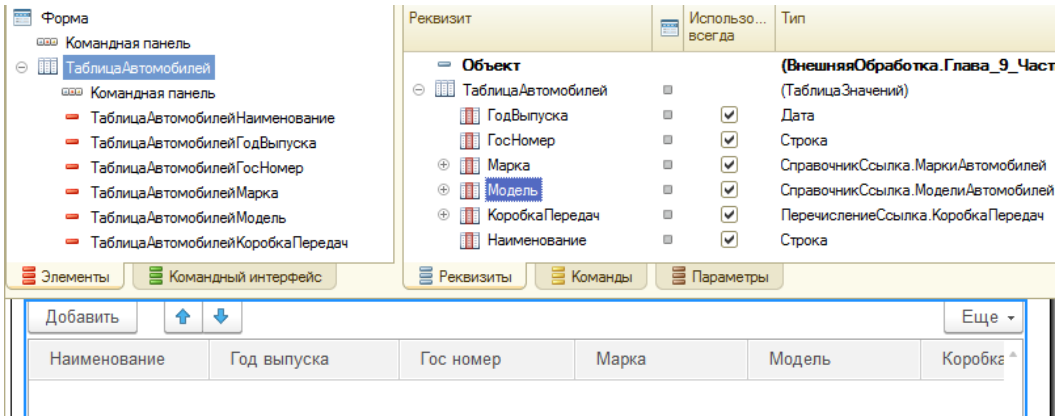


Рис. 9.2.1

Создайте команду «Заполнить таблицу» и разместите её в командной панели таблицы *ТаблицаАвтомобилей*. Для этой команды создайте обработчики на клиенте и на сервере. С

запросами мы будем работать только в серверном контексте. В принципе, с запросами можно работать и в клиентском контексте. Но Ваш код будет работать только в режиме *толстого клиента*. В *тонком клиенте* с запросами работать нельзя!

Для того, чтобы быстро и удобно создавать запросы, мы будем использовать конструктор запросов. Сам текст запроса необходимо передать в объект *Запрос*, поэтому будем использовать свойство *Текст* объекта *Запрос*. Вызовите его, и пока пропишите пустую строку (см. листинг 9.2.1).

```
&НаСервере
Процедура ЗаполнитьТаблицуНаСервере()
    ТаблицаАвтомобилей.Очистить();
    Запрос = Новый Запрос;
    Запрос.Текст = "";
КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьТаблицу(Команда)
    ЗаполнитьТаблицуНаСервере();
КонецПроцедуры
```

Листинг 9.2.1

Откроем конструктор запроса. Для этого необходимо поместить курсор между кавычек.

```
Запрос.Текст = "";
```

Рис. 9.2.2

И перейти: *«Главное меню» - «Текст» – «Конструктор запроса»*.

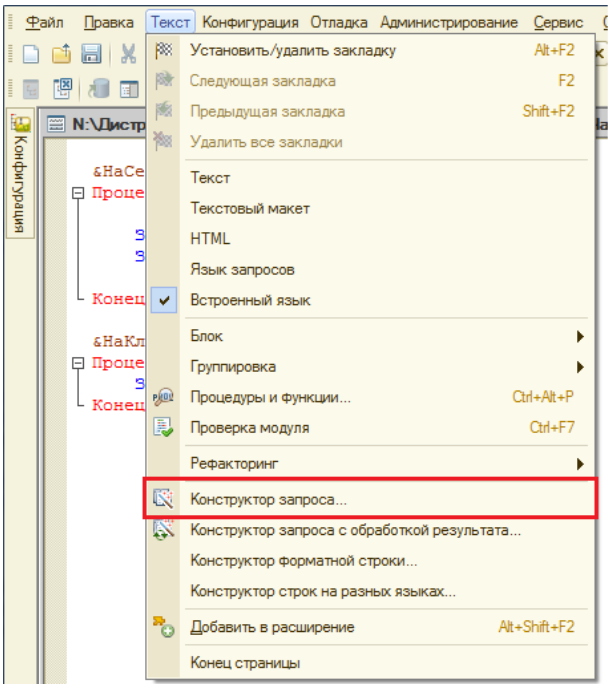


Рис. 9.2.3

Выйдет сообщение *«Не найден текст запроса, создать новый?»*, отвечаем *«Да»*. После ответа откроется конструктор запроса.

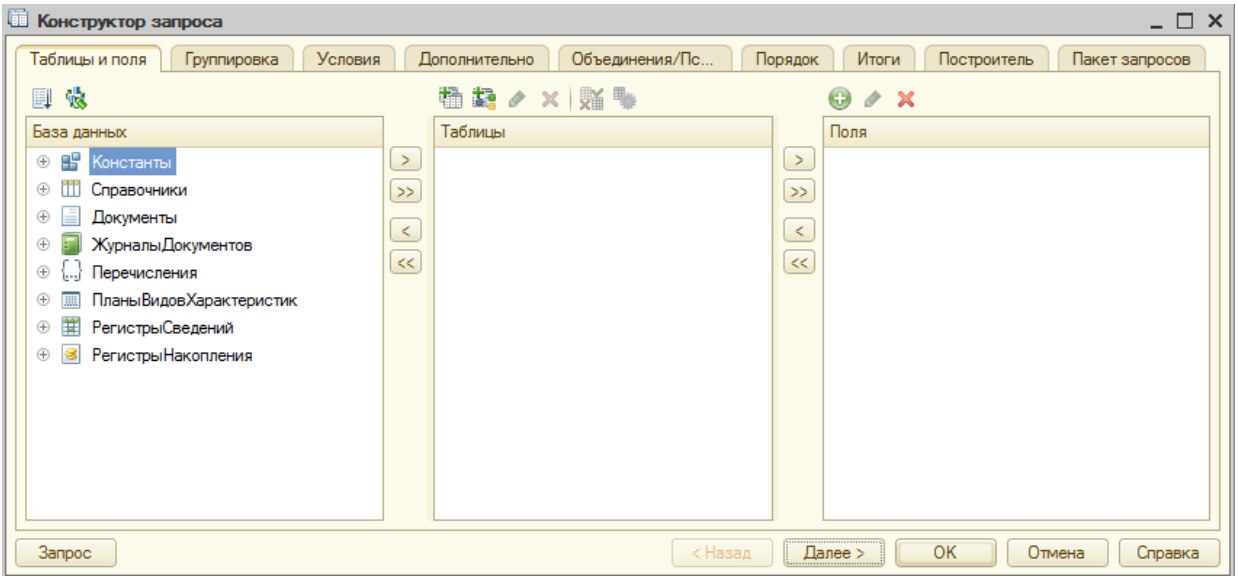


Рис. 9.2.4

Конструктор представляет форму с несколькими закладками. Нас пока интересует только первая закладка *Таблицы и поля*. Данная закладка состоит из трех списков. Самый левый список называется *База данных,* в нем отображены все метаданные базы (таблицы).

Из этого списка Вы можете выбрать любой объект, который нас интересует. Поскольку по условию задачи нужно вывести все элементы справочника *Автомобили*, то выбирайте этот справочник. Сделайте это, два раза кликнув по нему правой кнопкой мышки.

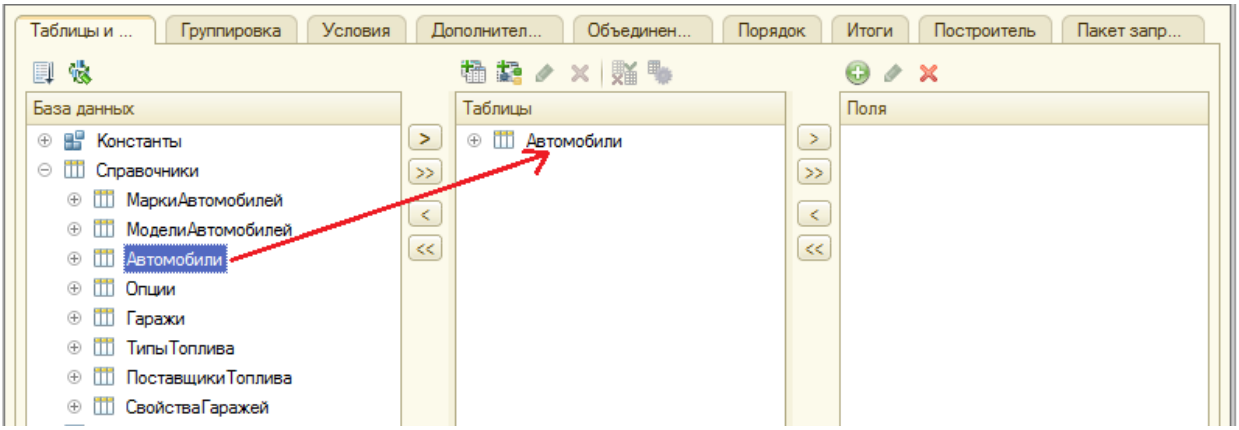


Рис. 9.2.5

Данный справочник выбран, и он появился в списке *Таблицы*. В этом списке отображаются все таблицы, данные которых можно вывести в запрос. Но наш запрос еще пустой, потому что нет полей, которые должны быть отображены в нем. Выберем поля. Для этого раскройте справочник *Автомобиль*.

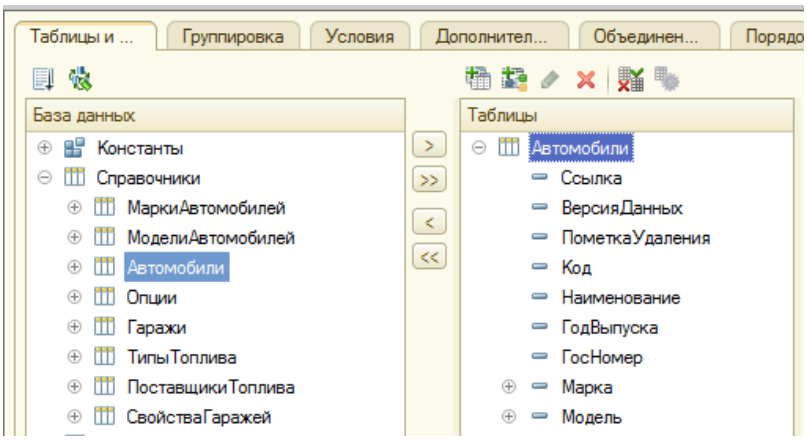


Рис. 9.2.6

И когда Вы будете дважды кликать правой кнопкой мышки по любому полю справочника, оно будет появляться в списке *Поля*. Поэтому выберем поля *Наименование*, *Год выпуска*, *ГосНомер*, *Коробка передач*, *Марка* и *Модель*.

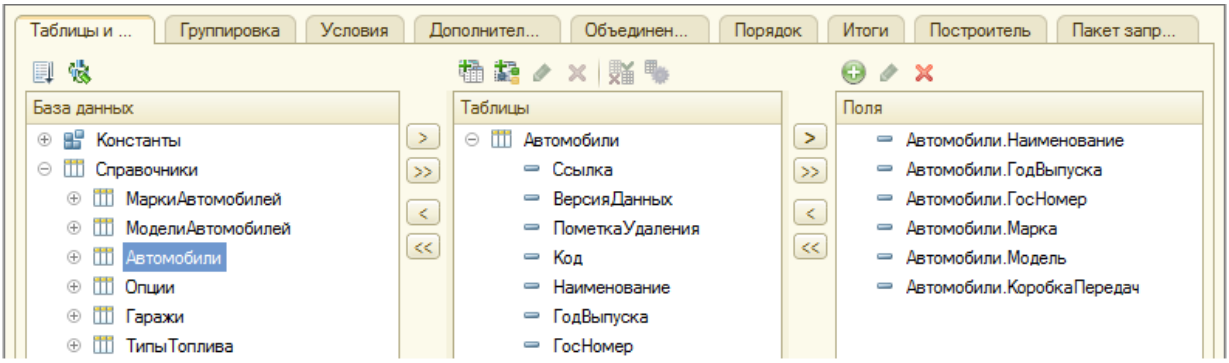


Рис. 9.2.7

В принципе, запрос создан, осталось только нажать на кнопку *ОК* конструктора. И соответствующий текст появился в модуле (см. листинг 9.2.2).

```
Запрос = Новый Запрос;
Запрос.Текст = "ВЫБРАТЬ
               |    Автомобили.Наименование КАК Наименование,
               |    Автомобили.ГодВыпуска КАК ГодВыпуска,
               |    Автомобили.ГосНомер КАК ГосНомер,
               |    Автомобили.Марка КАК Марка,
               |    Автомобили.Модель КАК Модель,
               |    Автомобили.КоробкаПередач КАК КоробкаПередач
               |ИЗ
               |    Справочник.Автомобили КАК Автомобили";
```

Листинг 9.2.2

В данном запросе три директивы: *«ВЫБРАТЬ»*, *«ИЗ»* и *«КАК»*.

Первая директива *«ВЫБРАТЬ»* определяет поля, которые будут выбраны из таблицы, они перечислены через запятую.

Вторая директива *«ИЗ»* показывает, из какого источника будут выбраны данные.

Третья директива *«КАК»* задает, каким образом будет называться источник данных в запросе.

Запрос непосредственно написан, но этого еще мало, чтобы получить данные, необходимо выполнить запрос к базе данных.

Для этого используется метод *Выполнить* объекта *Запрос*.

```
Результат = Запрос.Выполнить();
```

Листинг 9.2.3

Данный метод является функцией, которая возвращает объект, так и называющийся *Результат запроса*, этот объект предназначен для хранения полученных данных.

Но прежде чем начать обрабатывать данные, лучше проверить, есть ли они в принципе, ведь в справочнике *Автомобили* может и не быть никаких элементов. Для этого используем метод объекта *РезультатЗапроса*, который называется *Пустой*.

```
Если Результат.Пустой() Тогда
    Возврат;
КонецЕсли;
```

Листинг 9.2.4

Метод *Пустой* результат запроса возвращает *Истину,* если *Результат запроса* не содержит данных. В этом случае мы просто выходим из процедуры.

Рекомендую Вам всегда перед началом обработки результата запроса проверять его на наличие данных.

Результат получен, он не пустой, теперь нам необходимо вывести данные в таблицу значений. Сделать это можно двумя способами: используя *Выборку* и используя *Выгрузку*. Сейчас мы разберем оба этих способа.

Заполните Вашу таблицу на форме (см. листинг 9.2.5) с помощью *Выборки.*

```
Выборка = Результат.Выбрать();
Пока Выборка.Следующий() цикл
    НовСтр = ТаблицаАвтомобилей.Добавить();
    НовСтр.Наименование    = Выборка.Наименование;
    НовСтр.ГодВыпуска      = Выборка.ГодВыпуска;
    НовСтр.ГосНомер        = Выборка.ГосНомер;
    НовСтр.Марка           = Выборка.Марка;
    НовСтр.Модель          = Выборка.Модель;
    НовСтр.КоробкаПередач  = Выборка.КоробкаПередач;
КонецЦикла;
```

Листинг 9.2.5

С выборками Вы уже знакомы, только в данном случае будет объект *Выборка из результата запроса*. Как и в остальных выборках, мы также обращаемся к методу *Следующий*, чтобы позиционироваться на следующем элементе. Также получаем значения полей, только уже из запроса, обращаясь к выборке через точку, при этом указывая название поля.

Посмотрите, как работает Ваш запрос.

Наименование	Год выпуска	Гос номер	Марка	Модель	Коробка передач
Автомобиль главного директора	01.01.2016	х001кк18RU	Ford	Focus	Ручная
Дежурный автомобиль	10.05.2015	кк123ка18	Ford	Focus	Автоматическая
Авто мастера	10.10.2010	ув112ц19	Ford	Focus	Ручная
Автомобиль главного бухгалтера	01.01.2016	х0026618RU	KIA	Rio	Ручная
Автомобиль главного инженера	01.01.2016	х033ув18RU	Renault		Ручная

Рис. 9.2.8

Только что мы заполняли колонки таблицы, обращаясь к выборке, но можно сделать и проще, используя уже знакомую Вам процедуру глобального контекста *ЗаполнитьЗначенияСвойств*.

Переделайте Ваш код:

```
Пока Выборка.Следующий() цикл
    НовСтр = ТаблицаАвтомобилей.Добавить();
    ЗаполнитьЗначенияСвойств(НовСтр,Выборка);
КонецЦикла;
```

Листинг 9.2.6

*Как Вы видите, данный метод удобен для написания и экономит время программиста, но он выполняется **в два раза дольше**. Поэтому в случае больших объемов данных всегда обращайтесь непосредственно к выборке.*

Рассмотрим способ получения данных из запроса с помощью метода *Выгрузить*.

Тут все просто. Единственно, что название и тип полей в таблице должны совпадать с названием и типом полей в запросе.

```
ТаблицаАвтомобилей.Загрузить(Результат.Выгрузить());
```

Листинг 9.2.7

Сохраните, и запускаем обработку. Как видите, все прекрасно заполнилось. Но этот способ еще медленнее, чем предыдущий, поэтому используйте его только для небольших объемов данных.

Итак, мы научились создавать примитивные запросы, получать их результат, и если он заполнен данными, то выводить их разными способами. Теперь мы будем изучать непосредственно сами запросы, мы не будем затрагивать особенности их вывода. В дальнейшем самостоятельно решайте, каким способом Вам получать данные из запроса. Наиболее оптимальный на данный момент способ - это использование выборок.

Работа с секцией «ВЫБРАТЬ»

Только что мы научились работать с секцией *«ВЫБРАТЬ»*. После нее, как Вам уже известно, идет перечисление полей, которые будут отображены в запросе. Теперь рассмотрим основные директивы этой секции.

И первая директива - это директива *«РАЗЛИЧНЫЕ»*.

Создайте новую команду «Заполнить таблицу (различные)». Скопируйте в серверный обработчик команды уже созданный нами запрос и измените его – оставим только два поля. Это поле *Марка* и *Коробка передач* (см. листинг 9.2.8, рис. 9.2.9).

```
&НаСервере
Процедура ЗаполнитьТаблицуРазличныеНаСервере()
    ТаблицаАвтомобилей.Очистить();
    Запрос = Новый Запрос;
    Запрос.Текст = "ВЫБРАТЬ
                   |    Автомобили.Марка КАК Марка,
                   |    Автомобили.КоробкаПередач КАК КоробкаПередач
                   |ИЗ
                   |    Справочник.Автомобили КАК Автомобили";
    Результат = Запрос.Выполнить();
    Если Результат.Пустой() Тогда
        Возврат;
    КонецЕсли;
    Выборка = Результат.Выбрать();
    Пока Выборка.Следующий() цикл
        НовСтр = ТаблицаАвтомобилей.Добавить();
        ЗаполнитьЗначенияСвойств(НовСтр,Выборка);
    КонецЦикла;
КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьТаблицуРазличные(Команда)
    ЗаполнитьТаблицуРазличныеНаСервере();
КонецПроцедуры
```

Листинг 9.2.8

Для того, чтобы открыть конструктор уже созданного запроса, необходимо поставить курсор в любое место запроса, выполнить команду в меню Текст: «Главное меню» - «Текст» – «Конструктор запроса».

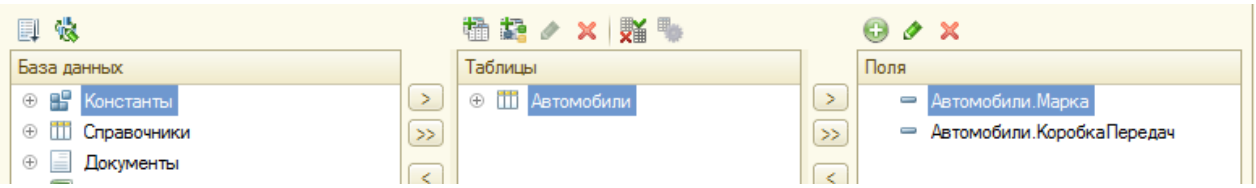


Рис. 9.2.9

Посмотрим, как у нас заполнится таблица на форме.

Наименование	Год выпуска	Гос номер	Марка	Модель	Коробка передач
			Ford		Ручная
			Ford		Автоматическая
			Ford		Ручная

Рис. 9.2.10

Естественно, остальные поля не заполнились, но нам они и не нужны, для тренировки сделайте их невидимыми (для продвинутых: делайте их невидимыми программно). Но среди тех строк, что вышли, мы видим повторяющиеся.

Наименование	Год выпуска	Гос номер	Марка	Модель	Коробка передач
			Ford		Ручная
			Ford		Автоматическая
			Ford		Ручная

Рис. 9.2.11

Директива *«РАЗЛИЧНЫЕ»* как раз необходима для того, чтобы в результате запроса не было повторяющихся строк. Поставим ее. Для этого зайдите обратно в конструктор.

Перейдите на закладку *«Дополнительно»* и поставьте флажок напротив выражения *«Без повторяющихся»*.

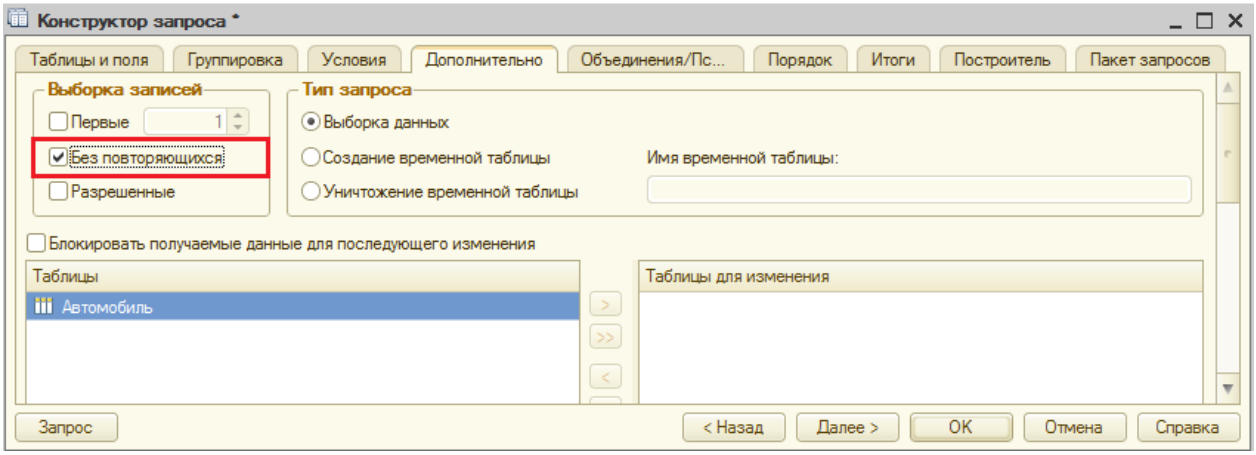


Рис. 9.2.12

После этого можно смело нажимать кнопку *ОК*. У Вас должен получиться следующий запрос:

```
Запрос.Текст = "ВЫБРАТЬ РАЗЛИЧНЫЕ
               |    Автомобили.Марка КАК Марка,
               |    Автомобили.КоробкаПередач КАК КоробкаПередач
               |ИЗ
               |    Справочник.Автомобили КАК Автомобили";
```

Листинг 9.2.9

Как Вы видите, после директивы *«ВЫБРАТЬ»* появилась директива *«РАЗЛИЧНЫЕ»*, которая и будет выводить только уникальные записи. Посмотрите, как работает Ваш новый запрос. Остались только уникальные записи.

Наименование	Год выпуска	Гос номер	Марка	Модель	Коробка передач
			Ford		Ручная
			Ford		Автоматическая

Рис. 9.2.13

Следующая директива – это директива *«ПЕРВЫЕ»*.

Данная директива позволяет выводить только несколько первых записей (число может быть любое, в пределах разумного). Данное свойство очень удобно, когда нужно вывести только одну запись.

Сейчас мы так и поступим: выведем только одну запись предыдущего запроса.

Создайте опять новую команду «Заполнить таблицу (первые)», в серверный обработчик которой скопируете код из самой первой команды. Зайдите опять в конструктор запроса, перейдите на закладку *«Дополнительно»* и установите флажок рядом со словом *«Первые»*.

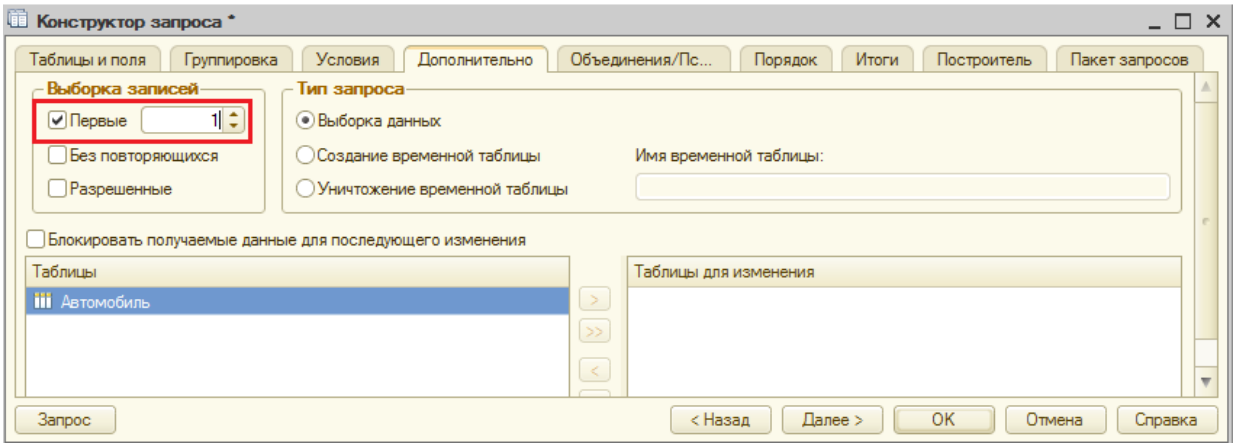


Рис. 9.2.14

И в соседнем поле цифру *1* оставьте как есть. Нажмите кнопку *ОК*, и запрос приобретет следующий вид:

```
Запрос.Текст = "ВЫБРАТЬ ПЕРВЫЕ 1
               |    Автомобили.Наименование КАК Наименование,
               |    Автомобили.ГодВыпуска КАК ГодВыпуска,
               |    Автомобили.ГосНомер КАК ГосНомер,
               |    Автомобили.Марка КАК Марка,
               |    Автомобили.Модель КАК Модель,
               |    Автомобили.КоробкаПередач КАК КоробкаПередач
               |ИЗ
               |    Справочник.Автомобили КАК Автомобили ";
```

Листинг 9.2.10

Как видите, в запросе после директивы «ВЫБРАТЬ» появилась директива *«ПЕРВЫЕ»* и рядом с ней цифра *1*. Сохраняем, и запустите обработку. Вышла только одна запись. Она же была и первая.

Наименование	Год выпуска	Гос номер	Марка	Модель	Коробка передач
Автомобиль гл...	01.01.2016	х001кк18RU	Ford	Focus	Ручная

Рис. 9.2.15

Условия запроса и передача параметров

Как Вы уже знаете, в выборке справочников или объектов можно отобрать только по одному, при этом индексируемому, реквизиту. В запросах таких ограничений нет, можно разрабатывать любые выборки данных с любым количеством отборов. Сейчас мы научимся это делать.

Добавьте на форму два реквизита, *Марка автомобиля* и *Коробка передач* (с соответствующими типами). И разместите эти реквизиты в виде полей ввода.

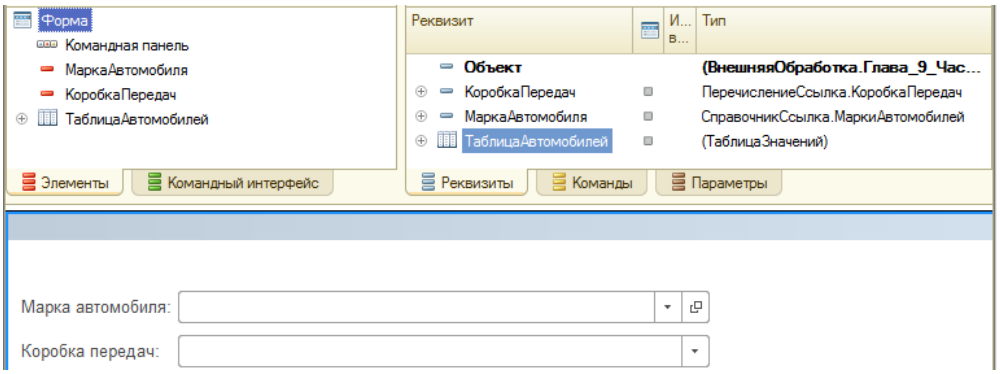


Рис. 9.2.16

Создайте новую команду «Заполнить таблицу с отбором», скопируйте в её серверный обработчик код из самой первой команды. При выполнении этой команды будем отбирать все соответствующие заданным условиям автомобили.

В начале процедуры обработчика команды в клиентском контексте напишите код, который будет выкидывать из процедуры, если один из параметров пустой. Для этого будем использовать уже знакомую Вам функцию *ЗначениеЗаполнено*.

```
&НаКлиенте
Процедура ЗаполнитьТаблицуСОтборами(Команда)
    Если Не ЗначениеЗаполнено(МаркаАвтомобиля) или
         Не ЗначениеЗаполнено(КоробкаПередач) тогда
         Сообщить("Не заполнены отборы");
         Возврат;
    КонецЕсли;
    ЗаполнитьТаблицуСОтборамиНаСервере();
КонецПроцедуры
```

Листинг 9.2.11

Я сделал намеренно проверку в клиентском контексте, чтобы лишний раз не осуществлять серверный вызов.

Теперь нам необходимо поставить условия для запроса в северной процедуре команды «Заполнить таблицу с отборами». Зайдите в конструктор уже созданного запроса и перейдите на закладку *Условия*.

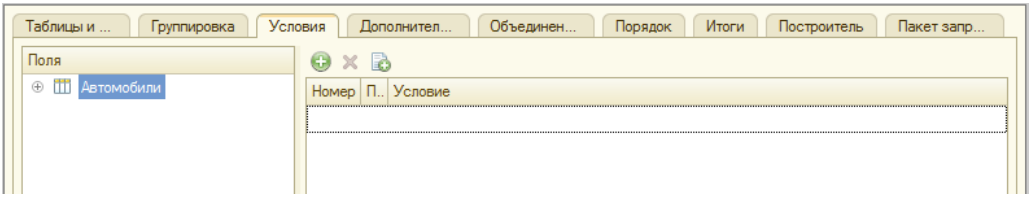


Рис. 9.2.17

В левой части условия находятся поля, которые можно отобрать. Как видите, их больше, чем полей, выбранных в первой закладке. Мы можем ставить условия по всем полям выбранных таблиц (даже тех, которых не будет в выборке). Пока выбрана только одна таблица *Автомобили*, вот ее поля можно и отбирать.

Выберите два поля: *Марка* и *Коробка передач*.

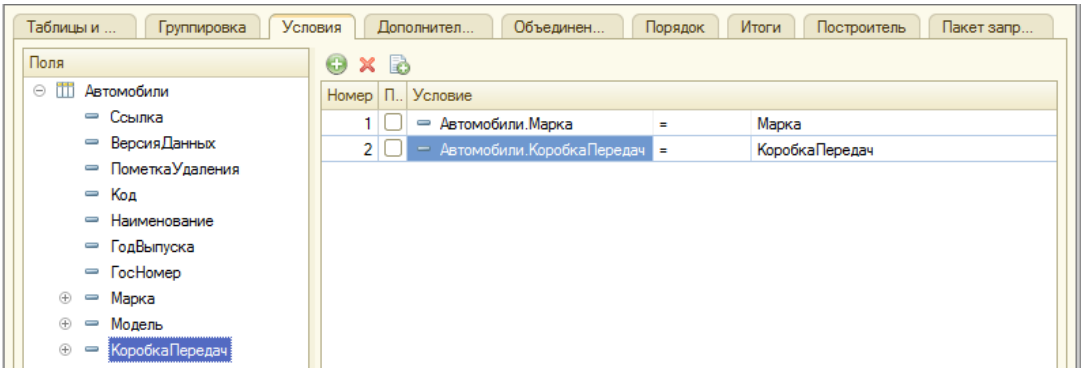


Рис. 9.2.18

Как видите, строчки заполнились автоматически. Разберем область справа.

В колонке *Условие* расположены непосредственно те поля, на которые ставятся условия. Их в каждой строке можно менять. В следующей колонке расположен *Способ сравнения*. Он по умолчанию равен *«Равно»*. Но можно и поменять на другое сравнение.

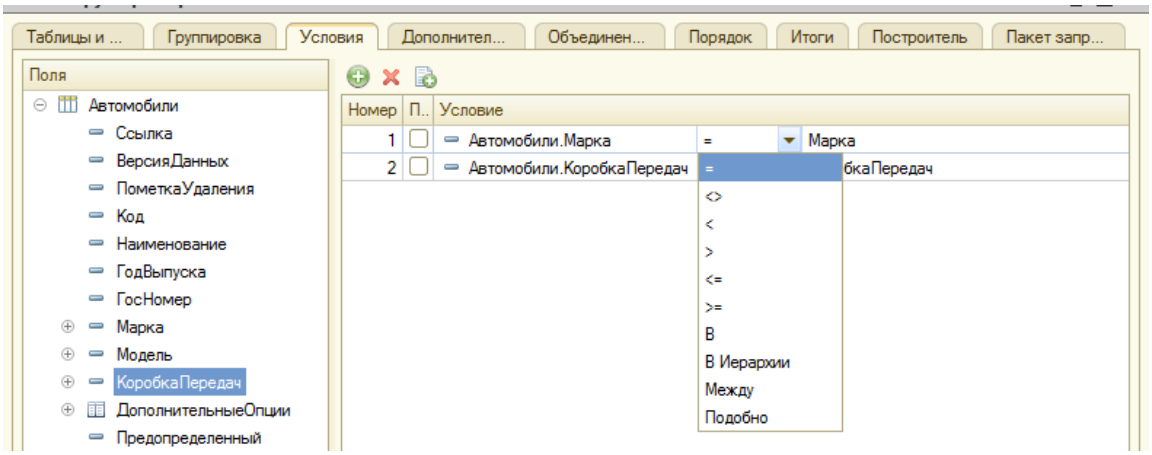


Рис. 9.2.19

В следующей колонке расположено непосредственно название параметра, как мы будем его передавать в запрос. Мы также можем поменять его. Если мы поставим флажок в колонку *Произвольное*, то сможем редактировать само условие.

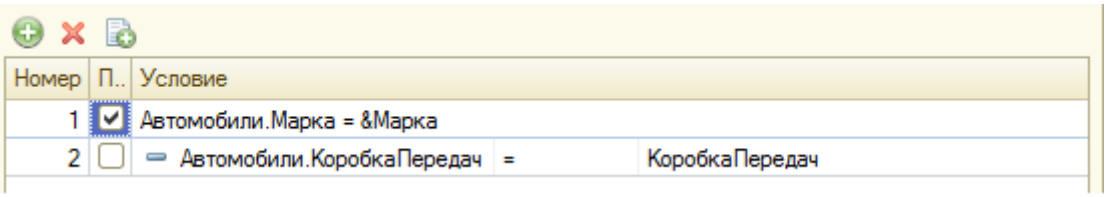


Рис. 9.2.20

Пока Вы не будете ничего менять, а просто оставите запрос как есть и нажмете на кнопку *ОК* конструктора запросов. Запрос будет иметь следующий вид:

```
Запрос.Текст = "ВЫБРАТЬ
               |    Автомобили.Наименование КАК Наименование,
               |    Автомобили.ГодВыпуска КАК ГодВыпуска,
               |    Автомобили.ГосНомер КАК ГосНомер,
               |    Автомобили.Марка КАК Марка,
               |    Автомобили.Модель КАК Модель,
               |    Автомобили.КоробкаПередач КАК КоробкаПередач
               |ИЗ
               |    Справочник.Автомобили КАК Автомобили
               |ГДЕ
               |    Автомобили.Марка = &Марка
               |    И Автомобили.КоробкаПередач = &КоробкаПередач";
```

Листинг 9.2.12

Вы видите, что добавилось еще одна директива - *«ГДЕ»*, оно определяет условия, которые накладываются на нашу выборку. Они перечислены после данной директивы, обратите внимание, что они логически связываются по операции *«И»*. Но Вы можете заменить ее на *«ИЛИ»*, если этого будет требовать задача (правда, не рекомендуется это делать в целях производительности), тогда Вам придется подправить запрос вручную. Все параметры, которые передаются в запрос извне, начинаются с символа &.

Итак, запрос написан, теперь нам необходимо передать значения наших реквизитов с формы в запрос. Делать мы это будем, используя метод *УстановитьПараметр* объекта *Запрос*.

```
Запрос.УстановитьПараметр("Марка",МаркаАвтомобиля);
Запрос.УстановитьПараметр("КоробкаПередач",КоробкаПередач);
Результат = Запрос.Выполнить();
```

Листинг 9.2.13

Данный метод имеет два обязательных параметра. Первый - это имя устанавливаемого в запросе параметра. А второй – непосредственно значение данного параметра, которое будет передано в запрос. В нашем случае это значение реквизита. Обращаю Ваше внимание, что первый параметр должен в точности совпадать с параметром в запросе, выделенным символом &.

В этот раз сообщите пользователю о пустом результате.

```
Если Результат.Пустой() Тогда
    Сообщить("Нет автомобилей с маркой " +
             Строка(МаркаАвтомобиля) +
             " и коробкой передач " +
             Строка(КоробкаПередач));
    Возврат;
КонецЕсли;
```

Листинг 9.2.14

Сохраните обработку. И посмотрите, как она работает.

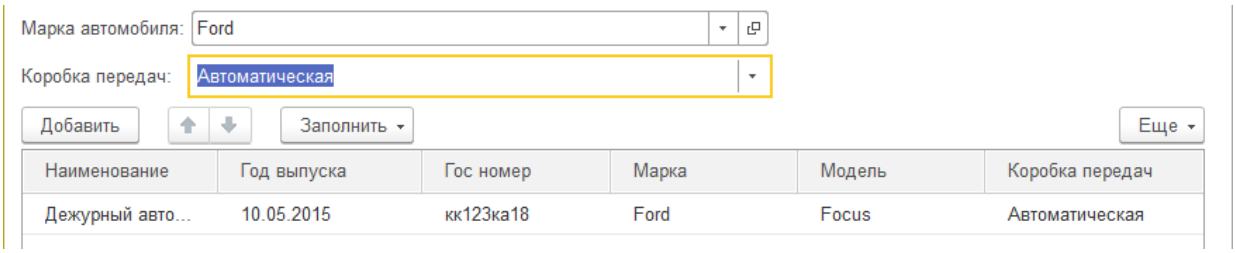


Рис. 9.2.21

Отборы работают как надо.

Рассмотрим еще один момент. К примеру, можно ли передать в условие сразу несколько элементов справочника *Марка*, чтобы вывелись автомобили марки *Ford* и *Renault*? Да, можно, для этого необходимо использовать вид сравнения *«В»*.

Измените Вашу обработку – добавьте реквизит «СписокМарок» с типом *СписокЗначений* (тип значения *СправочникСсылка.МаркиАвтомобилей*), который будем заполнять при открытии всеми марками автомобилей, присутствующими в справочнике. В запрос будем передавать только те марки, напротив которых стоит отметка.

Добавьте список на форму в виде таблицы и «перетащите» колонки *Значение* и *Пометка*. Также нашу основную таблицу сгруппируем с таблицей списка значений, чтобы они друг относительно друга располагались горизонтально (см. рис. 9.2.22).

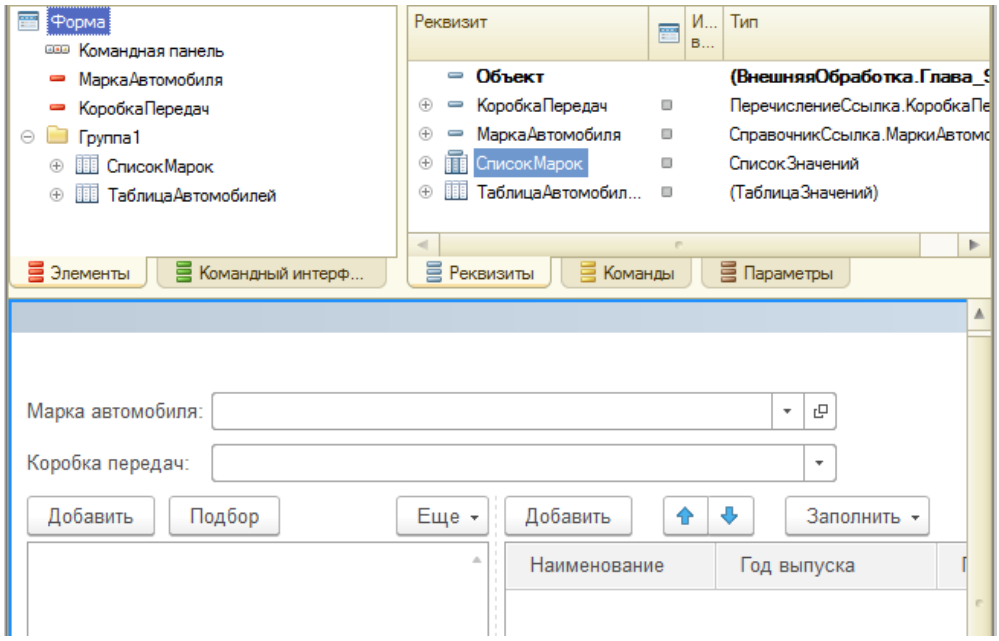


Рис. 9.2.22

Теперь в обработчике формы «ПриСозданииНаСервере», напишем код, заполняющий данный список.

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)
    ВыборкаМарка = Справочники.МаркиАвтомобилей.Выбрать();
    Пока ВыборкаМарка.Следующий() цикл
        СписокМарок.Добавить(ВыборкаМарка.Ссылка,,Ложь);
    КонецЦикла;
КонецПроцедуры
```

Листинг 9.2.15

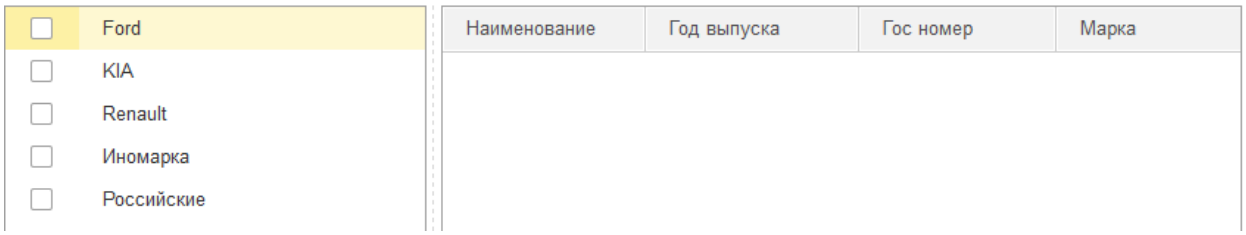


Рис. 9.2.23

Создадим команду «Заполнить таблицу с отбором по списку», скопируем в её серверный обработчик код из самой первой команды и доработаем его. Первым делом будем выгружать в массив помеченные элементы списка *СписокМарок.* Если массив окажется пустым (т.е. нет помеченных элементов), то выйдем из процедуры (см. листинг 9.2.16).

```
МассивМарок = Новый Массив;
Для Каждого ЭлСписка из СписокМарок цикл
    Если ЭлСписка.Пометка тогда
        МассивМарок.Добавить(ЭлСписка.Значение);
    КонецЕсли;
КонецЦикла;
```

```
Если МассивМарок.Количество() = 0 тогда
    Сообщить("Выберете хотя бы одну марку");
    Возврат;
КонецЕсли;
```

Листинг 9.2.16

Данный код прост и понятен, и не требует дополнительных пояснений. Осталось отредактировать имеющийся запрос: передадим в него в качестве параметра массив.

Откройте Ваш запрос в конструкторе запроса и перейдите на закладку условия. Создайте новое условие, в которое выберете поле *Марка* и вид сравнения *«В»*, а параметр назовите *«МассивМарок»*.

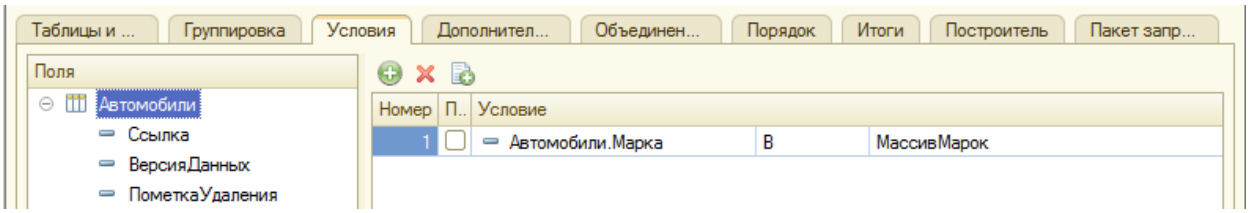


Рис. 9.2.24

Нажимаем кнопку *ОК,* и у Вас получится следующий запрос:

```
Запрос = Новый Запрос;
Запрос.Текст = "ВЫБРАТЬ
               |    Автомобили.Наименование КАК Наименование,
               |    Автомобили.ГодВыпуска КАК ГодВыпуска,
               |    Автомобили.ГосНомер КАК ГосНомер,
               |    Автомобили.Марка КАК Марка,
               |    Автомобили.Модель КАК Модель,
               |    Автомобили.КоробкаПередач КАК КоробкаПередач
               |ИЗ
               |    Справочник.Автомобили КАК Автомобили
               |ГДЕ
               |    Автомобили.Марка В(&МассивМарок)";
```

Листинг 9.2.17

Как видите, этот вид сравнения отличен от предыдущих и сам параметр задан в скобках. Следующим шагом установим параметры и выведем все в таблицу на форме.

```
Запрос.УстановитьПараметр("МассивМарок",МассивМарок);
Результат = Запрос.Выполнить();
Если Результат.Пустой() Тогда
    Возврат;
КонецЕсли;
Выборка = Результат.Выбрать();
Пока Выборка.Следующий() цикл
    НовСтр = ТаблицаАвтомобилей.Добавить();
    ЗаполнитьЗначенияСвойств(НовСтр,Выборка);
КонецЦикла;
```

Листинг 9.2.18

Посмотрите, как работает данный отбор.

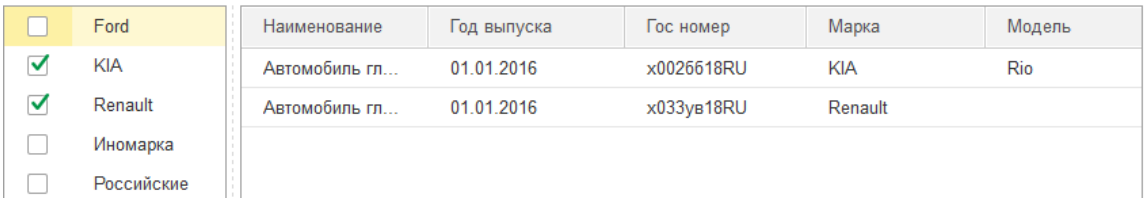


Рис. 9.2.25

Прекрасно видно, что вывелись именно те марки, которые были отмечены флажками.

С отборами мы закончили, и теперь перейдем к следующему пункту, это *Упорядочивание*.

Упорядочивание

Как Вы помните, в выборках можно было получить данные в определенном порядке, т.е. упорядоченные, но можно было использовать только индексируемые поля. В запросах тоже есть такая возможность, и она не зависит от индексации. Поэтому получим данные из справочника *Автомобили*, где упорядочим два поля. Первое поле - *Год выпуска*, второе поле - *Гос.номер*. Также создайте команду «Заполнить таблицу с упорядочиванием», в серверный обработчик которой скопируйте код из обработчика самой первой команды. Зайдите в конструктор имеющегося запроса и перейдите на закладку *Порядок*, где нужно выбрать два поля: *ГодВыпуска* и *ГосНомер* (см. рис. 9.2.26).

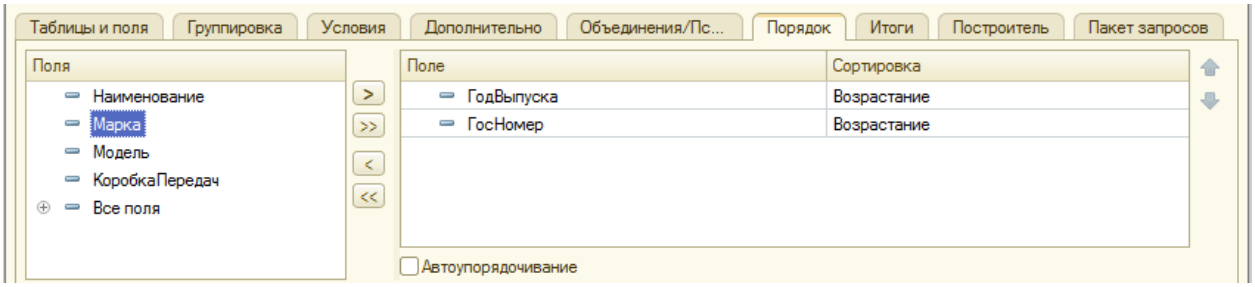


Рис. 9.2.26

Нажимаем кнопку *ОК* конструктора запросов, и получается следующий запрос:

```
Запрос.Текст = "ВЫБРАТЬ
               |    Автомобили.Наименование КАК Наименование,
               |    Автомобили.ГодВыпуска КАК ГодВыпуска,
               |    Автомобили.ГосНомер КАК ГосНомер,
               |    Автомобили.Марка КАК Марка,
               |    Автомобили.Модель КАК Модель,
               |    Автомобили.КоробкаПередач КАК КоробкаПередач
               |ИЗ
               |    Справочник.Автомобили КАК Автомобили
               |
               |УПОРЯДОЧИТЬ ПО
               |    ГодВыпуска,
               |    ГосНомер";
```

Листинг 9.2.19

Видно что, появились новая директива *«УПОРЯДОЧИТЬ ПО»*, после этих слов через запятую перечисляются поля, по которым будет вестись упорядочивание.

Обращаю Ваше внимание, что можно перечислить неограниченное количество полей. Также можно указать и обратную сортировку.

Зайдите опять в конструктор запроса и поставьте для поля *ГосНомер* сортировку в *Убывание*.

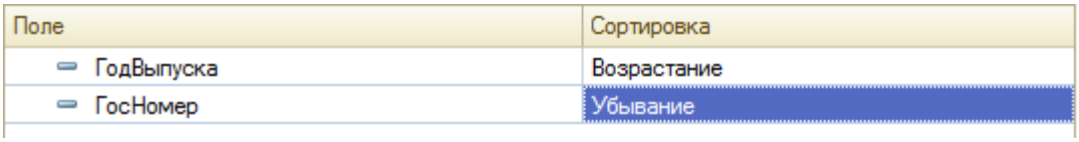


Рис. 9.2.27

В этом случае получиться следующий запрос:

```
Запрос.Текст = "ВЫБРАТЬ
                |    Автомобили.Наименование КАК Наименование,
                |    Автомобили.ГодВыпуска КАК ГодВыпуска,
                |    Автомобили.ГосНомер КАК ГосНомер,
                |    Автомобили.Марка КАК Марка,
                |    Автомобили.Модель КАК Модель,
                |    Автомобили.КоробкаПередач КАК КоробкаПередач
                |ИЗ
                |    Справочник.Автомобили КАК Автомобили
                |
                |УПОРЯДОЧИТЬ ПО
                |    ГодВыпуска,
                |    ГосНомер УБЫВ";
```

Листинг 9.2.20

Мы видим, что рядом с полем *ГосНомер* появилось слово *«Убыв»*. Это означает, что конкретное поле будет отсортировано по убыванию.

Посмотрите, как отсортируются Ваши данные.

Наименование	Год выпуска	Гос номер	Марка	Модель	Коробка пер
Авто мастера	10.10.2010	ув112ц19	Ford	Focus	Ручная
Дежурный авто...	10.05.2015	кк123ка18	Ford	Focus	Автоматиче
Автомобиль гл...	01.01.2016	х033ув18RU	Renault		Ручная
Автомобиль гл...	01.01.2016	х0026618RU	KIA	Rio	Ручная
Автомобиль гл...	01.01.2016	х001кк18RU	Ford	Focus	Ручная

Рис. 9.2.28

Мы видим, что данные отсортированы по *Году выпуска*, а потом по *Гос.номеру* автомобиля.

Работа с полями

Если какое-нибудь поле имеет ссылочный тип, то можно получить значение реквизита того объекта, на которое оно ссылается. Для этого достаточно обратиться к реквизиту данного объекта через точку. Эта операция называется *разыменование полей*. Продемонстрируем. Для этого измените последний запрос, и получим код марки автомобиля.

Создайте новую команду «Заполнить таблицу с кодом марки» и, как обычно, скопируйте в серверный обработчик этой команды код из серверного обработчика самой первой команды.

Зайдите в конструктор запроса, раскройте таблицу *Автомобили*, обратите внимание на реквизит *Марка*, нажмите на плюс рядом с ним и раскройте его.

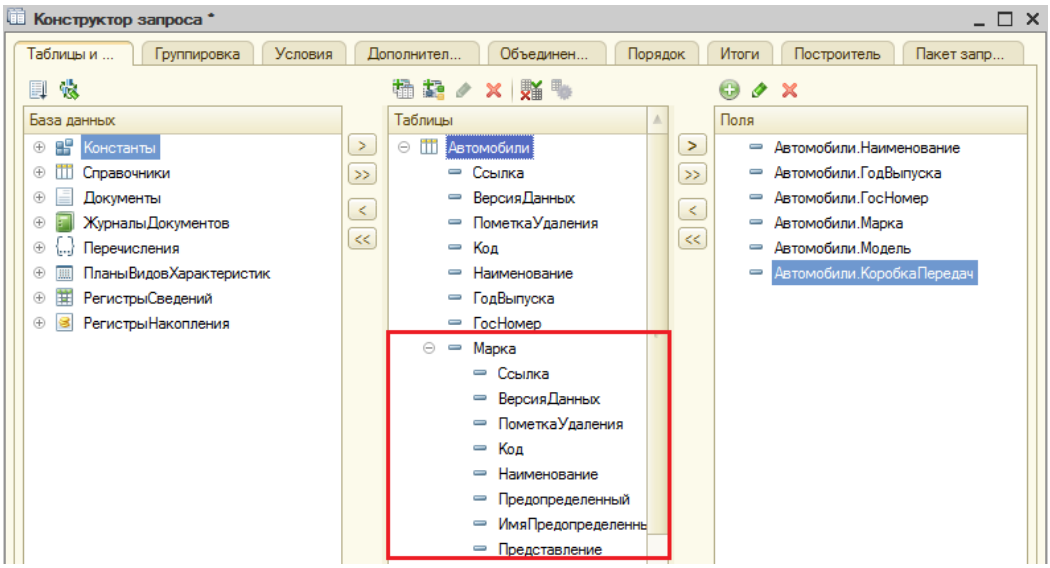


Рис. 9.2.29

В этом выпадающем списке должны появиться все реквизиты, которые есть у справочника *Марка*. Выбираем *Код*.

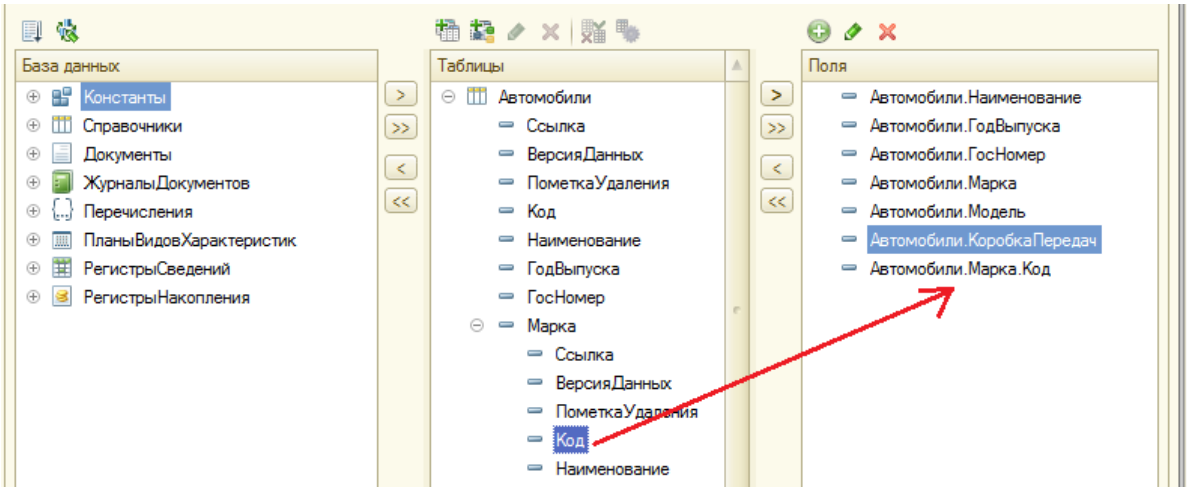


Рис. 9.2.30

Видите: поле *Код* вышло через вторую точку. Теперь о том, каким образом мы будем обращаться к данному полю в выборке по результату запроса. Точно узнать об этом можно, если перейдете на закладку *Объединения/Псевдонимы.*

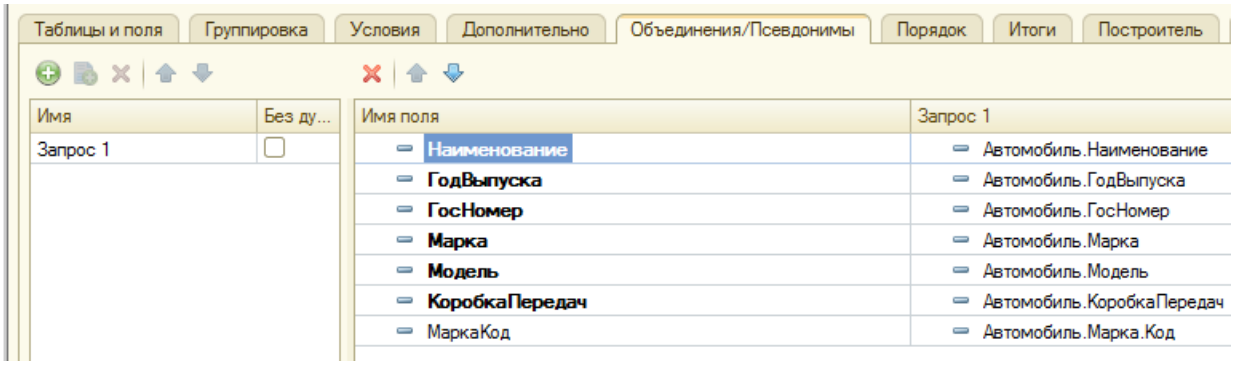


Рис. 9.2.31

Левая таблица *Объединения* нас пока не интересуют, а вот на правую таблицу *Псевдонимы* мы обратим внимание. В этой таблице с левой стороны идет название поля, а с правой - что это поле представляет собой в запросе.

Левую часть мы можем менять по своему усмотрению. Поэтому изменим поле *Марка код* на *Код марки*.

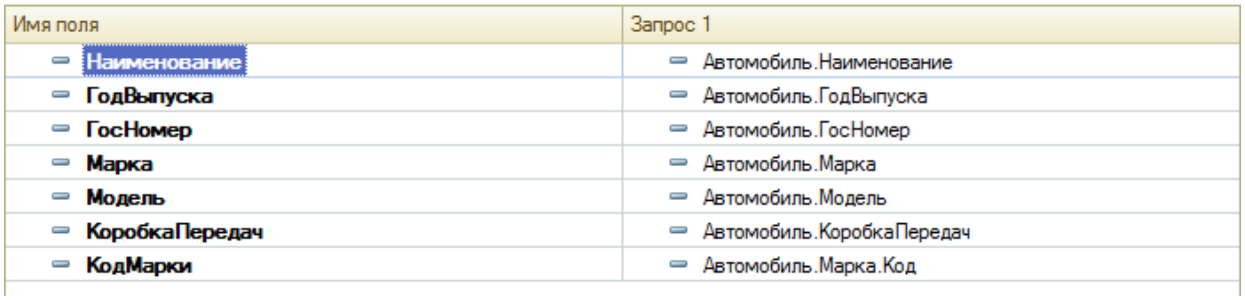


Рис. 9.2.32

Теперь самостоятельно добавьте новое поле *Код марки* в таблицу значений и соответствующую колонку в таблицу формы. А потом выведите в таблицу *Код марки*, как это показано на рис. 9.2.33.

Наименование	Год выпуска	Гос номер	Код марки	Марка	Модель	Коробка передач
Автомобиль главного ди...	01.01.2016	х001кк18RU	003	Ford	Focus	Ручная
Дежурный автомобиль	10.05.2015	кк123ка18	003	Ford	Focus	Автоматическая
Авто мастера	10.10.2010	ув112ц19	003	Ford	Focus	Ручная
Автомобиль главного бу...	01.01.2016	х0026618RU	004	KIA	Rio	Ручная
Автомобиль главного ин...	01.01.2016	х033ув18RU	005	Renault		Ручная

Рис. 9.2.33

Обращаю Ваше внимание: если Вам нужно получить реквизит поля ссылочного типа, то получать его нужно через разыменование полей, а не уже в выборке через точку. Так Вы существенно оптимизируете Ваш запрос.

Объединение запросов

Поставим задачу: необходимо вывести в одну таблицу документы прибытия и выбытия автомобилей. С теми знаниями, которые есть у Вас, ее можно решить только одним способом: сначала с помощью запроса вывести документы прибытия, а потом выбытия. Это будет, во-первых, трудоемко, а во-вторых, этот процесс будет долог по времени, т.к. понадобится два раза обращаться к базе. Есть гораздо более быстрый способ - использовать объединение таблиц.

Создайте новую обработку, форму обработки, реквизит *ТаблицаДокументов* с типом *ТаблицаЗначений*, у которого будет единственная колонка *Документ* (составной тип данных *ссылка на документы прибытия и выбытия*). Данный реквизит будет размещен на форме в виде таблицы (см. рис. 9.2.34).

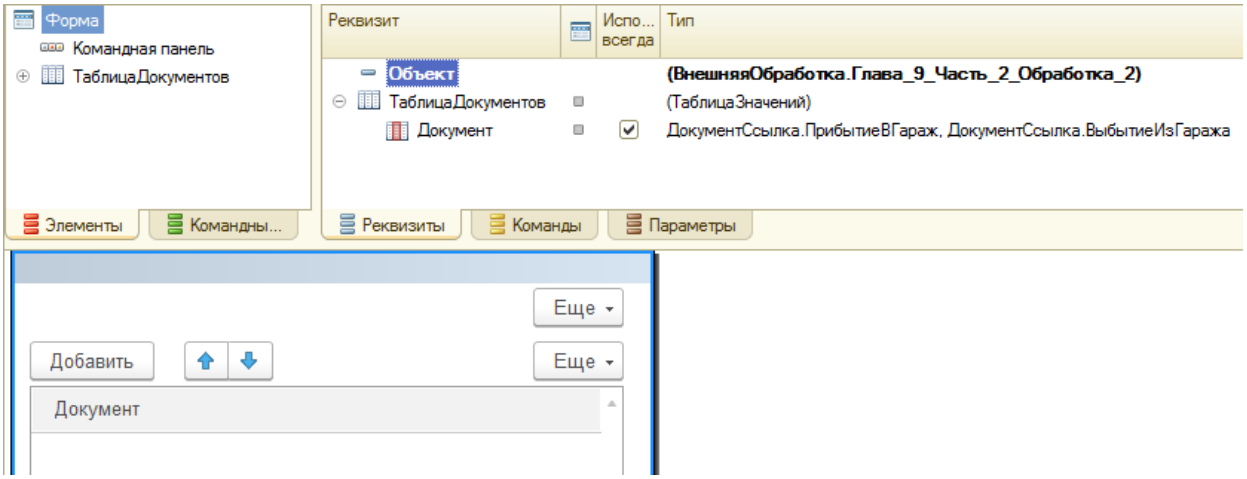


Рис. 9.2.34

Создадим команду «Заполнить таблицу» и в серверном обработчике команды создадим новый запрос, зайдя в конструктор запросов. В конструкторе выберете документ *Прибытие в гараж* и его поле *Ссылка* (см. рис. 9.2.35).

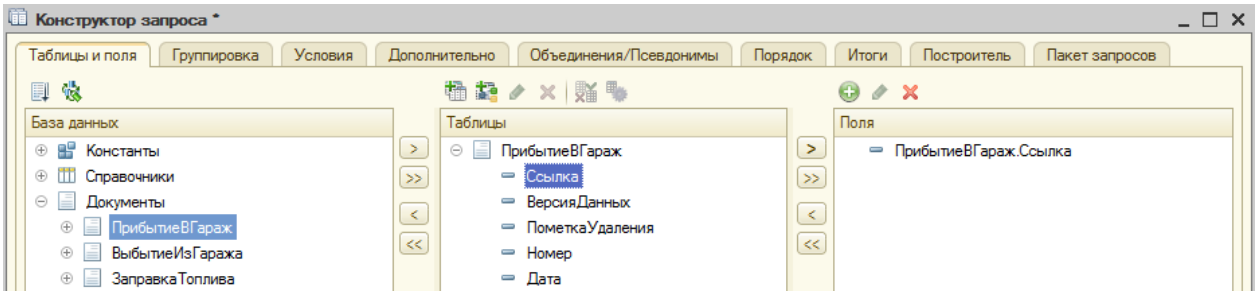


Рис. 9.2.35

Сейчас необходимо создать второй запрос с документом *Выбытие из гаража*. Для этого перейдите в уже знакомую Вам закладку *Объединения/Псевдонимы* и в левой таблице нажмите кнопку *Добавить*.

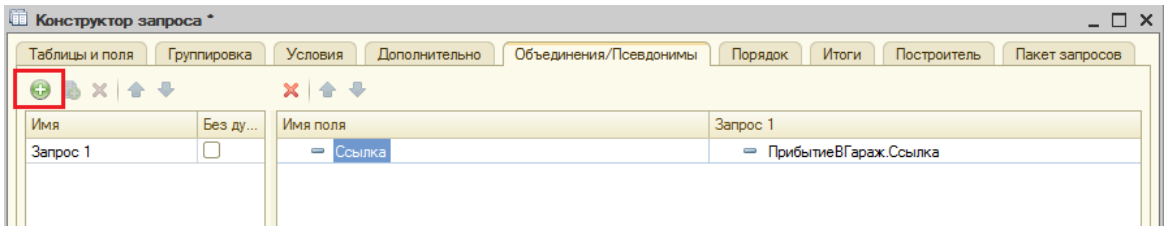


Рис. 9.2.36

Будет создан новый запрос, и автоматически Вы перейдете на него.

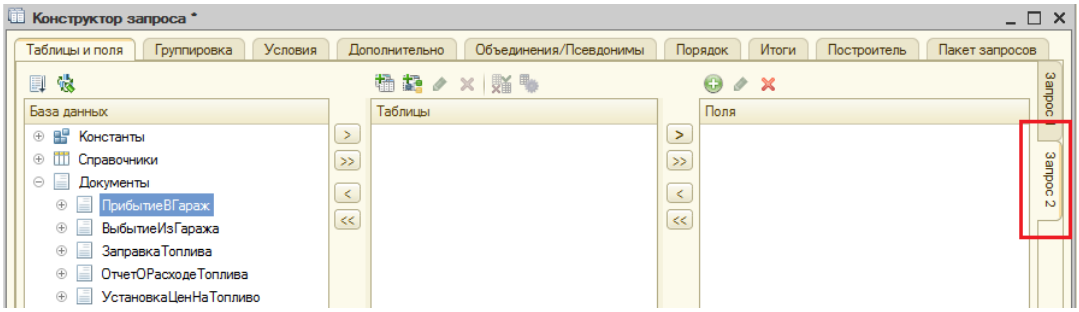


Рис. 9.2.37

Справа у конструктора видны закладки запросов. Добавьте новую таблицу *Выбытие из гаража* и у нее поле *Ссылка*.

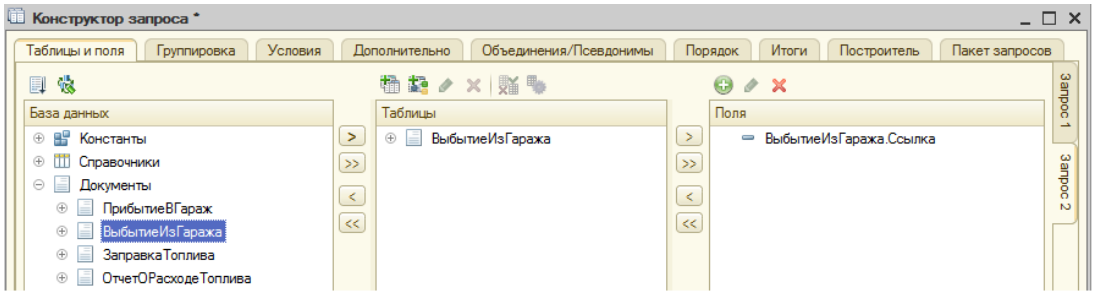


Рис. 9.2.38

Нажмите кнопку *ОК*. И получился следующий запрос:

```
Запрос.Текст = "ВЫБРАТЬ
               |    ПрибытиеВГараж.Ссылка КАК Ссылка
               |ИЗ
               |    Документ.ПрибытиеВГараж КАК ПрибытиеВГараж
               |
               |ОБЪЕДИНИТЬ ВСЕ
               |
               |ВЫБРАТЬ
               |    ВыбытиеИзГаража.Ссылка
               |ИЗ
               |    Документ.ВыбытиеИзГаража КАК ВыбытиеИзГаража";
```

Листинг 9.2.21

Вы видите, появились два новых ключевых слова: *«ОБЪЕДИНИТЬ»* и *«ВСЕ»*. Слово *«ОБЪЕДИНИТЬ»* означает, что два запроса объединены и будут в одной выборке. А слово *«ВСЕ»*

значит, что при объединении возможны дубли. Конкретно в этом случае такое невозможно, но бывают запросы, при объединении которых могут быть одинаковые наборы данных. Чтобы они не дублировались, необходимо убрать слово *«ВСЕ»* после слова *«ОБЪЕДИНИТЬ»*. Сделать это можно либо вручную, либо в конструкторе.

Сделаем это в конструкторе. Зайдите обратно в конструктор. Перейдите на закладку *Объединения/Псевдонимы*. И напротив второго запроса ставим флажок в колонке *Без дублей*.

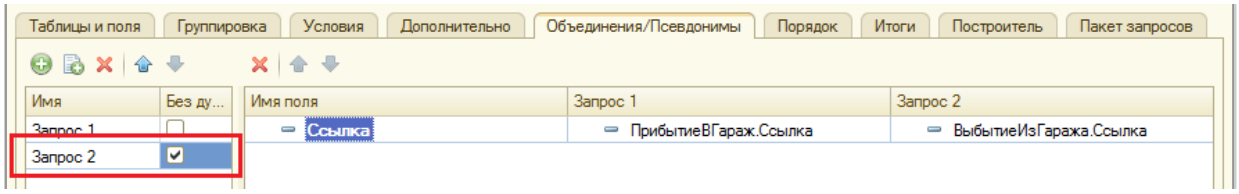


Рис. 9.2.39

Нажмите кнопку *ОК*, и слово *«ВСЕ»* должно исчезнуть из запроса.

Остался последний момент. У нас у таблицы значений единственная колонка имеет название *Документ*, а в запросе отображается название *Ссылка*. Дадим нашему полю псевдоним под названием «Документ» во все той же закладке *Объединения/Псевдонимы* (см. рис. 9.2.40). Для того, чтобы можно было использовать метод «ЗаполнитьЗначенияСвойств».

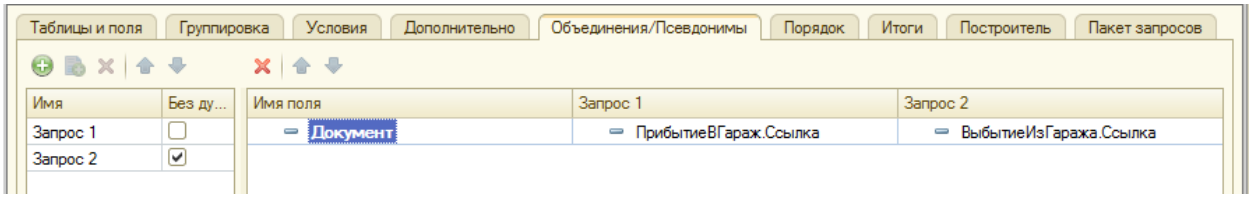


Рис. 9.2.40

Обратите внимание, мы даем псевдоним поля только для первой таблицы. После сохранения запроса ключевое слово «КАК» будет присутствовать в самом верхнем запросе (см. листинг 9.2.22).

Напишите код по выводу документов в таблицу значений самостоятельно.

Запустите обработку и посмотрите, что у нас выйдет (см. рис. 9.2.41).

```
Запрос.Текст = "ВЫБРАТЬ
               |    ПрибытиеВГараж.Ссылка КАК Документ
               |ИЗ
               |    Документ.ПрибытиеВГараж КАК ПрибытиеВГараж
               |
               |ОБЪЕДИНИТЬ
               |
               |ВЫБРАТЬ
               |    ВыбытиеИзГаража.Ссылка
               |ИЗ
               |    Документ.ВыбытиеИзГаража КАК ВыбытиеИзГаража";
```

Листинг 9.2.22

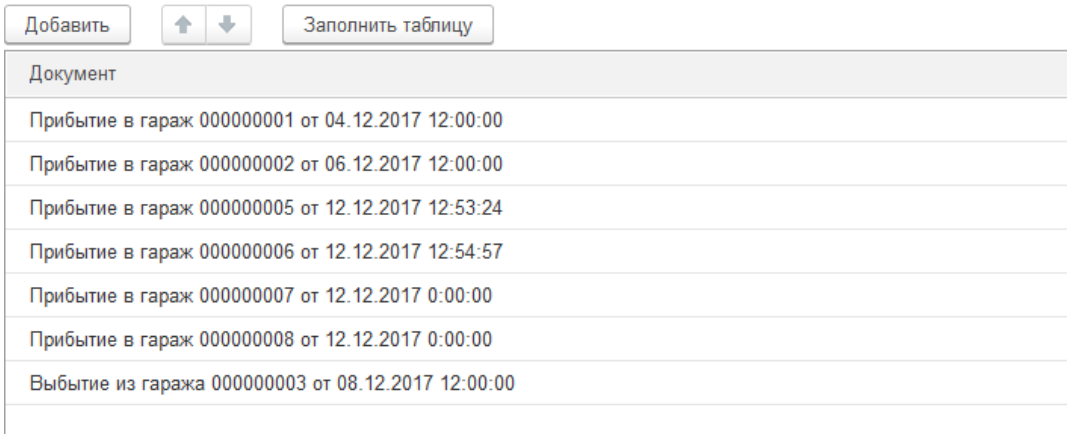


Рис. 9.2.41

Теперь добавим еще одно поле в нашу таблицу на форме, это *Дата Прибытия Выбытия*, в данное поле должны попадать даты прибытия автомобиля и даты выбытия автомобиля. Для этого зайдите обратно в конструктор запроса, и в первом запросе добавляем поле *Дата прибытия*.

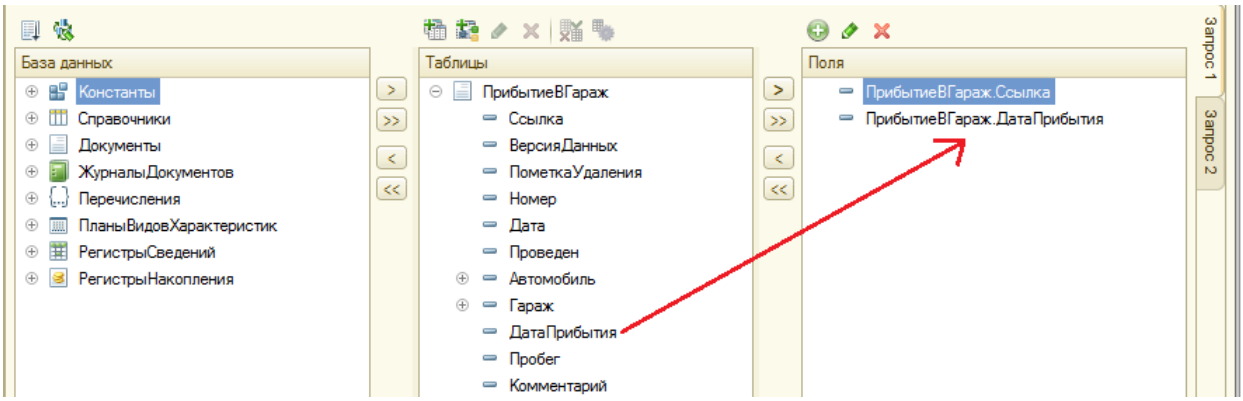


Рис. 9.2.42

Следующим шагом перейдите на закладку *Объединения/Псевдонимы*, и переименуйте наше поле в «ДатаПрибытияВыбытия».

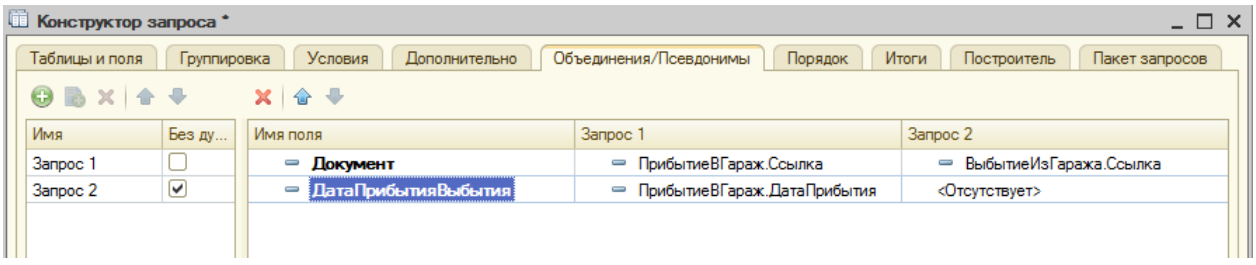


Рис. 9.2.43

Потом перейдите во второй запрос и добавьте новое поле - *Дата выбытия*.

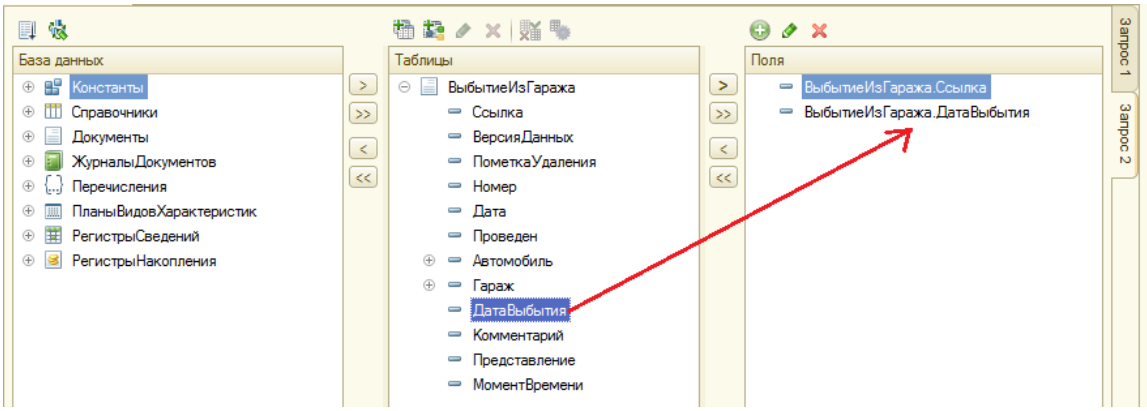


Рис. 9.2.44

Вернитесь опять на закладку *Объединения /Псевдонимы*. И Вы увидите, что в правой таблице появилась новая запись:

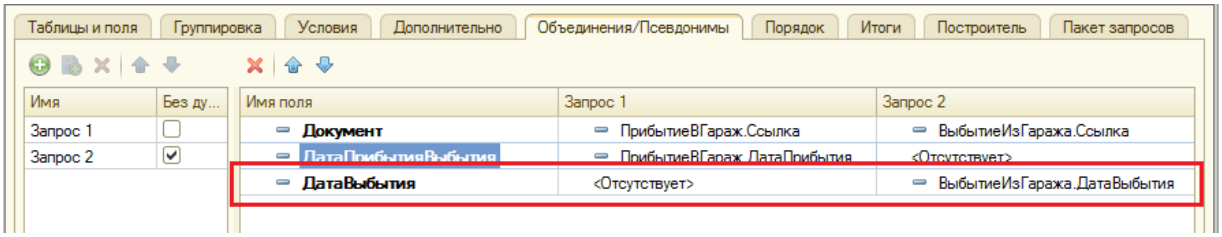


Рис. 9.2.45

Нам этого не надо, поэтому мы во второй строке во втором запросе выбираем *Дату выбытия.*

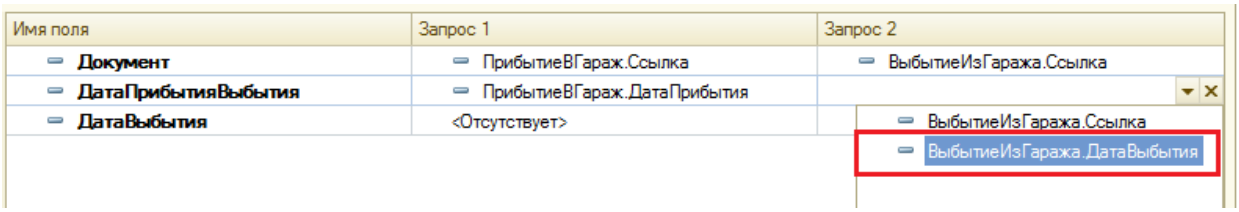


Рис. 9.2.46

Третья строка сразу исчезает (см. рис. 9.2.47), нажимаем кнопку *ОК,* и получится следующий запрос (см. листинг 9.2.23).

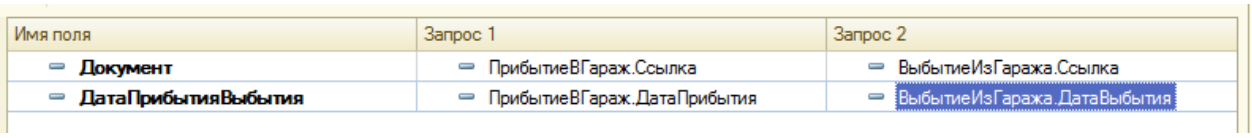


Рис. 9.2.47

```
Запрос.Текст = "ВЫБРАТЬ
               |    ПрибытиеВГараж.Ссылка КАК Документ,
               |    ПрибытиеВГараж.ДатаПрибытия КАК ДатаПрибытияВыбытия
               |ИЗ
               |    Документ.ПрибытиеВГараж КАК ПрибытиеВГараж
               |
               |ОБЪЕДИНИТЬ
               |
               |ВЫБРАТЬ
               |    ВыбытиеИзГаража.Ссылка,
               |    ВыбытиеИзГаража.ДатаВыбытия
               |ИЗ
               |    Документ.ВыбытиеИзГаража КАК ВыбытиеИзГаража";
```

Листинг 9.2.23

Как видно из кода, все псевдонимы к полям присваиваются в самом первом запросе. Доработайте самостоятельно обработку, чтобы даты прибытия и выбытия выводились в таблицу формы.

Документ	Дата прибытия выбытия
Прибытие в гараж 000000001 от 04.12.2017 12:00:00	04.12.2017
Прибытие в гараж 000000002 от 06.12.2017 12:00:00	06.12.2017
Прибытие в гараж 000000005 от 12.12.2017 12:53:24	12.12.2017
Прибытие в гараж 000000006 от 12.12.2017 12:54:57	15.12.2017
Прибытие в гараж 000000007 от 12.12.2017 0:00:00	12.12.2017
Прибытие в гараж 000000008 от 12.12.2017 0:00:00	12.12.2017
Выбытие из гаража 000000003 от 08.12.2017 12:00:00	08.12.2017

Рис. 9.2.48

Итак, мы научились объединять запросы. Хочу заметить, что объединять можно не только два, но и три, и четыре запроса. Количество их не ограничено. Но псевдонимы всегда будут присваиваться только в первом запросе.

Соединение таблиц в запросе

Рассмотрим поэтапно небольшую задачу. Первым делом выведем в таблицу на форме все непомеченные на удаление марки автомобилей из одноименного справочника. Создайте для этого обработку, форму обработки, реквизит *ТаблицаМарокИМоделей* с типом *ТаблицаЗначений* и с одной колонкой: *Марка* с соответствующими типом. Поместите этот реквизит на форму в виде таблицы. Создайте команду «Заполнить таблицу», которую разместите в командной панели таблицы формы.

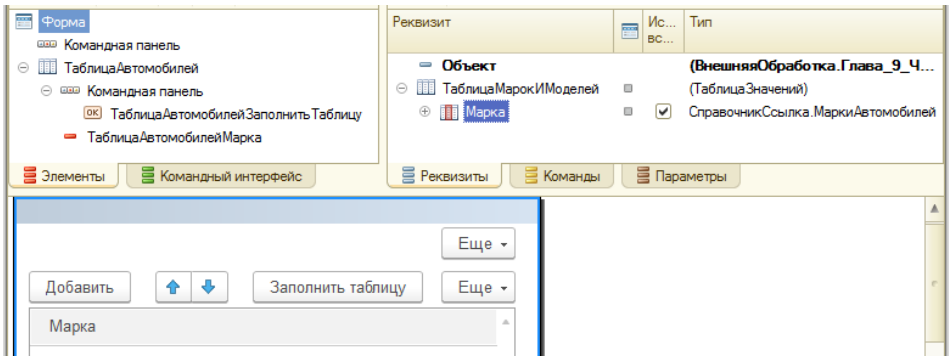


Рис. 9.2.49

Напишите запрос и обработку запроса, с помощью которого будете выводить в табличное поле все непомеченные на удаление элементы справочника *Марка автомобиля*.

```
&НаСервере
Процедура ЗаполнитьТаблицуНаСервере()
    Запрос = Новый Запрос;
    Запрос.Текст = "ВЫБРАТЬ
                   |    МаркиАвтомобилей.Ссылка КАК Марка
                   |ИЗ
                   |    Справочник.МаркиАвтомобилей КАК МаркиАвтомобилей
                   |ГДЕ
                   |    МаркиАвтомобилей.ПометкаУдаления = ЛОЖЬ";
    Результат = запрос.Выполнить();
    Если Результат.Пустой() Тогда
        Возврат;
    КонецЕсли;
    ТаблицаМарокИМоделей.Очистить();
    Выборка = Результат.Выбрать();
    Пока Выборка.Следующий() Цикл
        НовСтр = ТаблицаМарокИМоделей.Добавить();
        ЗаполнитьЗначенияСвойств(НовСтр,Выборка);
    КонецЦикла;
КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьТаблицу(Команда)
    ЗаполнитьТаблицуНаСервере();
КонецПроцедуры
```

Листинг 9.2.24

Обратите внимание, как мы отсекаем помеченные на удаление элементы справочника. Делаем мы это при помощи уже знакомой нам директивы *ГДЕ*, только в этот раз мы не используем параметр, а сразу присваиваем полю определенное значении *ЛОЖЬ*. Чтобы это можно было сделать, в конструкторе необходимо установить флаг «Произвольный» и написать вручную код: *МаркаАвтомобиля.ПометкаУдаления = ЛОЖЬ*

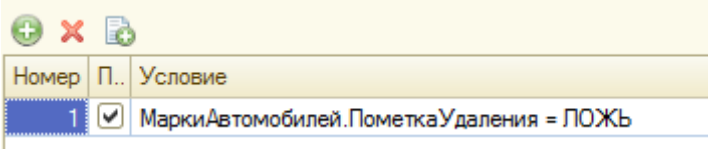


Рис. 9.2.50

При сравнении булевых значений можно не использовать непосредственно оператор сравнения «=», А просто в условии указывать нужное поле, тип которого *Булево*. Если нужно отобрать записи, в которых это поле *Истина*, то просто в условии прописываем нужное булево поле *МаркаАвтомобиля.ПометкаУдаления*. А если наоборот – записи, где это поле *Ложь*, то необходимо использовать булев оператор «Не», тогда условие будет следующего вида: *НЕ МаркаАвтомобиля.ПометкаУдаления.* Переделаем наше условие:

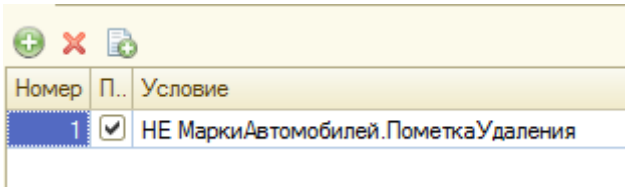


Рис. 9.2.51

Получится запрос следующего вида:

```
Запрос.Текст = "ВЫБРАТЬ
               |    МаркиАвтомобилей.Ссылка КАК Марка
               |ИЗ
               |    Справочник.МаркиАвтомобилей КАК МаркиАвтомобилей
               |ГДЕ
               |    НЕ МаркиАвтомобилей.ПометкаУдаления";
```

Листинг 9.2.25

Сохраним обработку и проверим, как работает Ваш запрос.

Рис. 9.2.52

Как видите, все марки автомобилей из данного справочника вышли в таблицу формы.

Теперь усложним задачу. В четвертой главе мы создали подчиненный справочнику *МаркиАвтомобилей* справочник *МоделиАвтомобилей*, в котором мы указываем модели у конкретных марок.

Теперь нам необходимо узнать, у каких из выведенных марок имеются модели в справочнике *МоделиАвтомобилей*. Как это сделать? Для этого мы соединим таблицу *МаркиАвтомобилей* с таблицей *МоделиАвтомобилей.*

Добавим еще одну колонку в таблицу значений – это *Модель*, соответствующего типа (см. рис. 9.2.53).

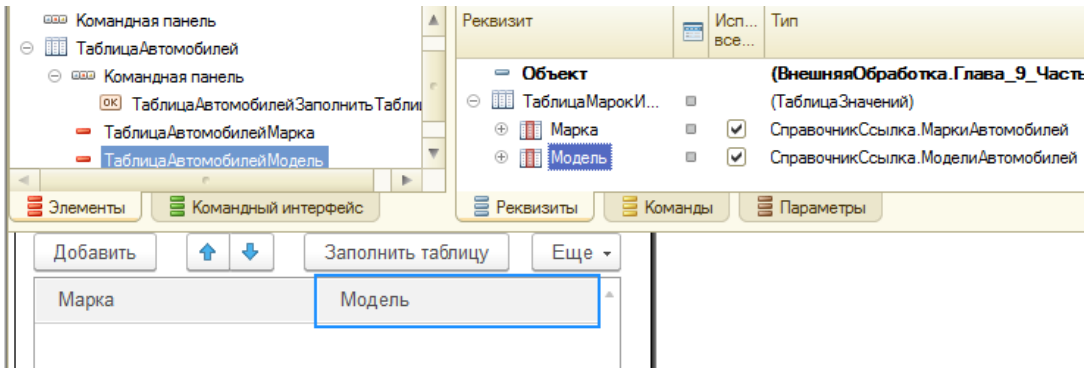


Рис. 9.2.53

Теперь изменим последний запрос. Зайдем обратно в конструктор запросов и раскроем ветку *Справочники* (см. рис. 9.2.54).

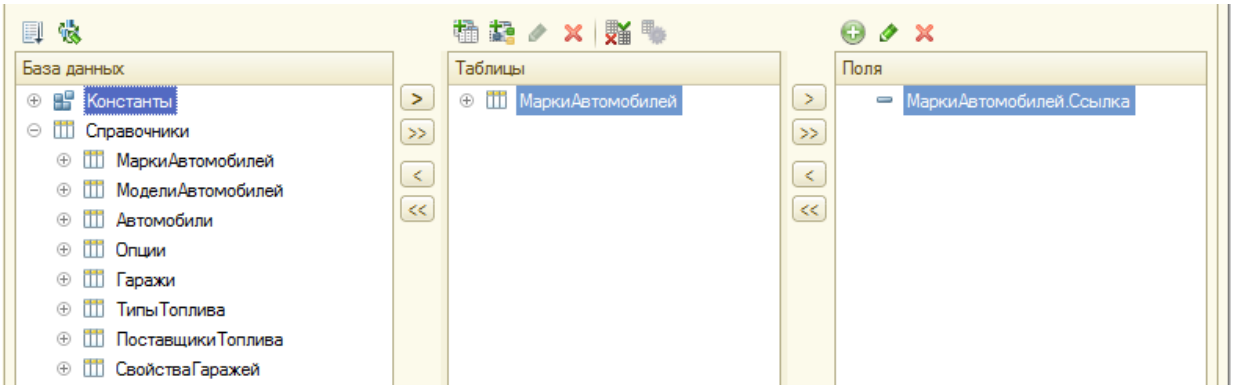


Рис. 9.2.54

И выберите таблицу *МоделиАвтомобилей,* дважды щелкнув по ней мышкой.

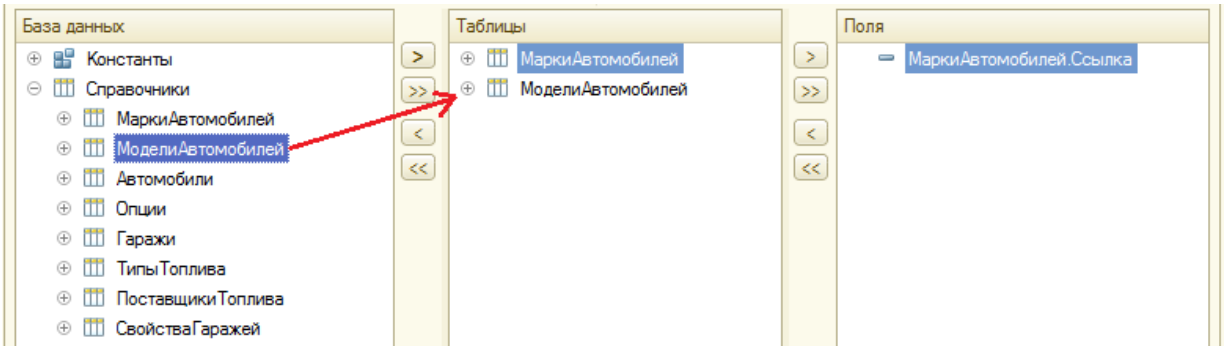


Рис. 9.2.55

Таблица выбралась. Теперь нам надо установить связь. Связь определяет, по каким полям наши таблицы будут между собой соединены. Если мы не определим ее, то будут выведены записи всех таблиц одновременно. Перейдите на закладку *Связи* и удалите имеющуюся связь, т.к. наша задача - самим научиться устанавливать связи.

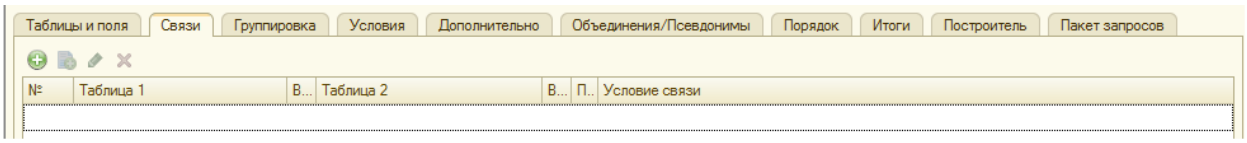


Рис. 9.2.56

Создадим новую связь, нажав кнопку *Добавить*. Появиться строка с пустыми колонками.

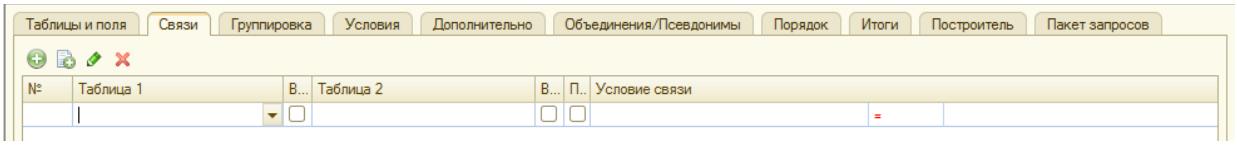


Рис. 9.2.57

Выберите первую таблицу. Это будет таблица *МаркиАвтомобилей*.

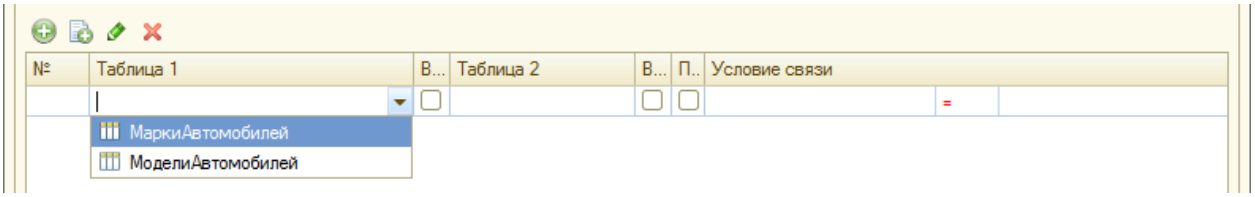


Рис. 9.2.58

Теперь выбираем вторую таблицу – *МоделиАвтомобилей*.

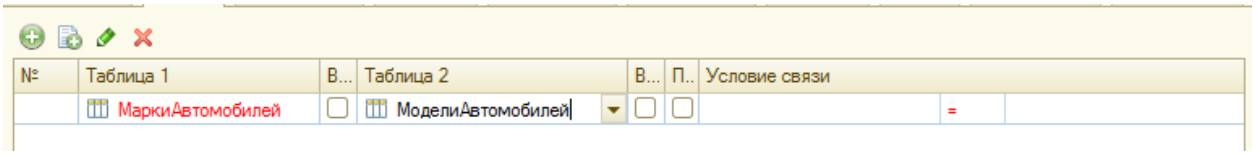


Рис. 9.2.59

Осталось выбрать поля, по которым будет установлено соединение. Со стороны таблицы *МаркиАвтомобилей* - это будет поле *Ссылка*, а со стороны таблицы *МоделиАвтомобилей* - это поле *Владелец*.

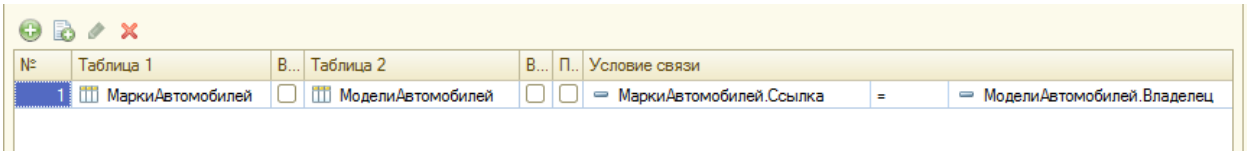


Рис. 9.2.60

Сейчас мы рассмотрим *Левое соединение*. Это значит, что будут выведены все записи из левой таблицы и те записи, для которых будет найдено соответствие, из правой. Поэтому мы

поставим галочку рядом с таблицей *МаркиАвтомобилей*, это значит, что будут выведены все записи из данной таблицы.

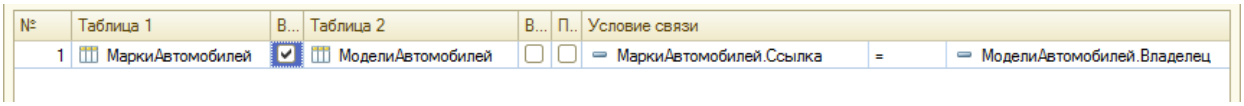


Рис. 9.2.61

Выберите поле *Ссылка* из таблицы *МоделиАвтомобилей* в окне *Поля* закладки *Таблицы и поля*.

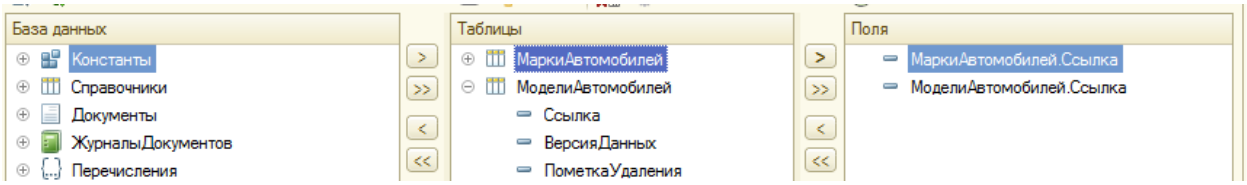


Рис. 9.2.62

Присвоим этому полю псевдоним «Модель».

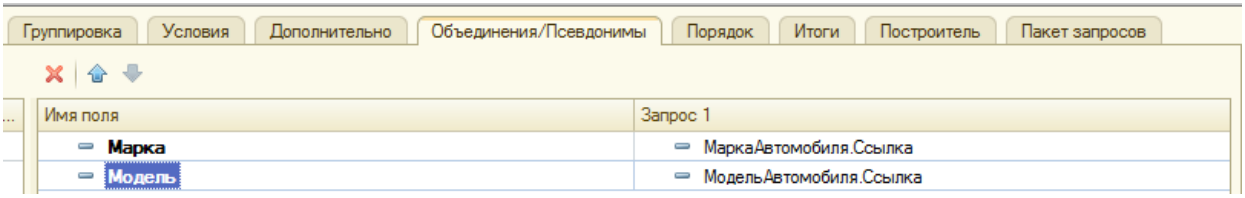


Рис. 9.2.63

Сохраните запрос, нажав кнопку *ОК*. Теперь Ваш запрос выглядит так:

```
ВЫБРАТЬ
    МаркиАвтомобилей.Ссылка КАК Марка,
    МоделиАвтомобилей.Ссылка КАК Модель
ИЗ
    Справочник.МаркиАвтомобилей КАК МаркиАвтомобилей
        ЛЕВОЕ СОЕДИНЕНИЕ Справочник.МоделиАвтомобилей КАК МоделиАвтомобилей
        ПО МаркиАвтомобилей.Ссылка = МоделиАвтомобилей.Владелец
ГДЕ
    НЕ МаркиАвтомобилей.ПометкаУдаления
```

Листинг 9.2.26

Вы видите, что появилось несколько новых директив - это *«ЛЕВОЕ СОЕДИНЕНИЕ»*, оно обозначает, что таблица *МаркаАвтомобиля* связана с таблицей *МоделиАвтомобиля* левым соединением, и директива *«ПО»*, после которой идет перечисление полей, по которым связаны наши таблицы. В данном случае они связаны по одному полю, но могут быть связаны и по нескольким.

Теперь сохраним обработку и запустим ее в программе «1С:Предприятие».

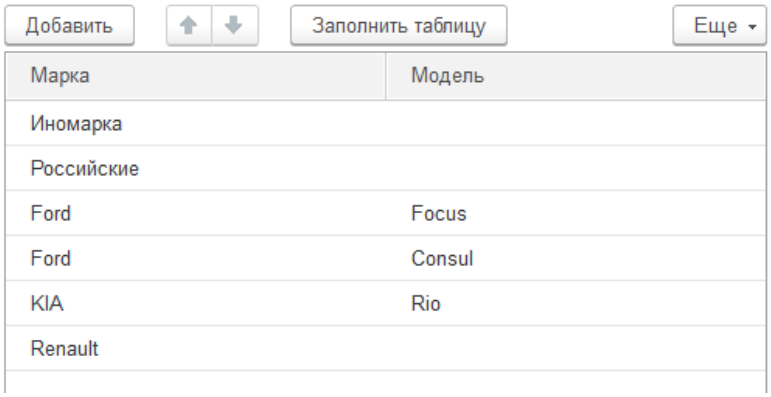


Рис. 9.2.64

Первое что нам бросается в глаза, это то, что два раза выведена строка с маркой Ford, так произошло потому, что для *марки автомобиля* Ford указано две модели - Focus и Consul. Поэтому в результат запроса вышла комбинация всех вариантов.

Также обратите внимание на пустые строки в колонке *Модель*, это значит, что у данных марок нет моделей в справочнике. И поэтому таблица вернула значение *Null*.

А как нам вывести только те марки, у которых есть модели? Сделать это можно тремя способами.

Первый способ – вывести, используя внутреннее соединение. Зайдите обратно в конструктор запросов, перейдите на закладку *Связи* и уберите флажок напротив таблицы *МаркаАвтомобиля*.

Когда нет флажков ни слева, ни справа, это значит, что будет использовано внутреннее соединение. Нажмите кнопку *ОК* и посмотрите, что получилось.

```
ВЫБРАТЬ
  МаркиАвтомобилей.Ссылка КАК Марка,
  МоделиАвтомобилей.Ссылка КАК Модель
ИЗ
  Справочник.МаркиАвтомобилей КАК МаркиАвтомобилей
      ВНУТРЕННЕЕ СОЕДИНЕНИЕ Справочник.МоделиАвтомобилей КАК
МоделиАвтомобилей
      ПО МаркиАвтомобилей.Ссылка = МоделиАвтомобилей.Владелец
ГДЕ
  НЕ МаркиАвтомобилей.ПометкаУдаления
```

Листинг 9.2.27

Видно, что директива *«ЛЕВОЕ СОЕДИНЕНИЕ»* поменялась на *«ВНУТРЕННЕЕ СОЕДИНЕНИЕ»*. Это значит, что будут выводиться только те записи, для которых установлена связь.

Посмотрите, как будет формироваться наша таблица. Сохраните обработку и запустите заново.

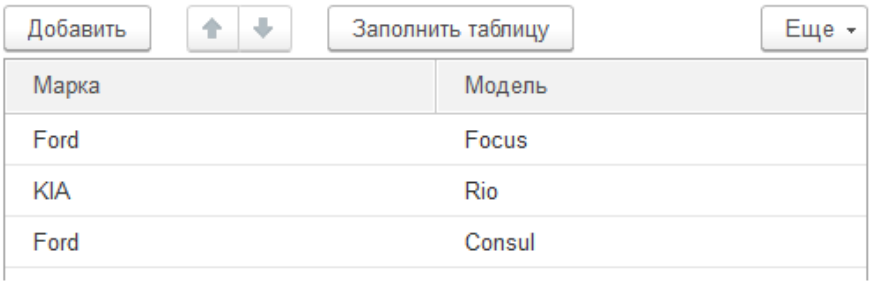


Рис. 9.2.65

Мы видим, что вышли только те марки, для которых имеются подчиненные элементы в справочнике *МоделиАвтомобилей*.

Второй способ вывести только те марки, у которых есть модели - это использовать *Правое соединение*.

Мы не сможем настроить его в конструкторе, поэтому просто вручную напишем в запросе:

```
ВЫБРАТЬ
  МаркиАвтомобилей.Ссылка КАК Марка,
  МоделиАвтомобилей.Ссылка КАК Модель
ИЗ
  Справочник.МаркиАвтомобилей КАК МаркиАвтомобилей
      ПРАВОЕ СОЕДИНЕНИЕ Справочник.МоделиАвтомобилей КАК МоделиАвтомобилей
      ПО МаркиАвтомобилей.Ссылка = МоделиАвтомобилей.Владелец
ГДЕ
  НЕ МаркиАвтомобилей.ПометкаУдаления
```

Листинг 9.2.28

Правое соединение означает, что выберутся все записи из таблицы справа и те записи, которые им соответствуют, из левой таблицы. Посмотрите самостоятельно, что будет с записями.

1С не рекомендует использовать правое соединение, а вместо этого применять левое, просто меняя таблицы местами.

Сделайте это самостоятельно и посмотрите, что получится.

Остался еще один вид соединения - это *Полное соединение*. Полное соединение - это, по сути, и левое, и правое соединение, т.е. объединяются данные таблицы слева и таблицы справа. Причем выводятся записи, которые удовлетворяют условию, а также все остальные.

Для того чтобы разобрать данное соединение, рассмотрим следующую задачу. У нас есть документ *Прибытие автомобиля* и документ *Выбытие автомобиля*. Соединим эти два документа по полю *Автомобиль* полным соединением.

Для этого создайте новую обработку, форму, на которой разместите реквизит «Таблица документов» с типом *ТаблицаЗначений*, у которой будут следующие колонки: *Прибытие* (ссылка на документ *Прибытие в гараж*), *Выбытие* (ссылка на документ *Выбытие из гаража*), *Автомобиль прибытия* и *Автомобиль выбытия* (ссылка на справочник «Автомобиль»).

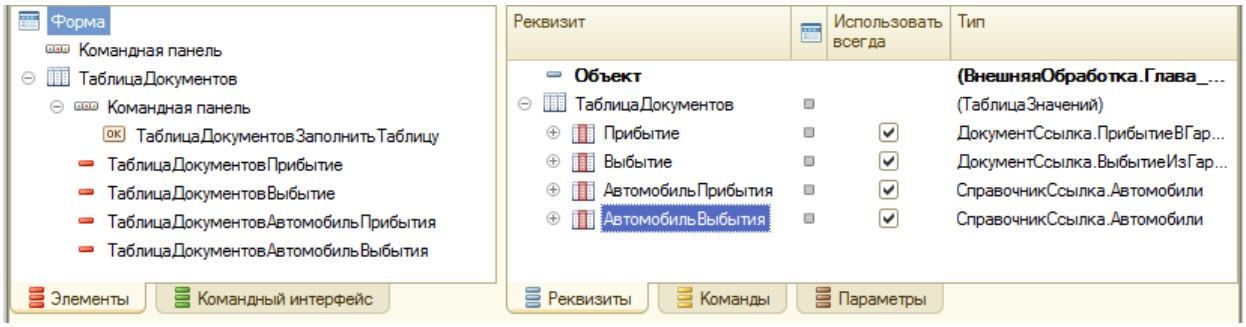


Рис. 9.2.66

Создадим команду «Заполнить таблицу» и для этой команды создадим обработчики на сервере и на клиенте. В серверном обработчике создадим новый объект «Запрос». Зайдем в конструктор запросов и выберем две таблицы – документ *Прибытие в гараж* и документ *Выбытие из гаража*.

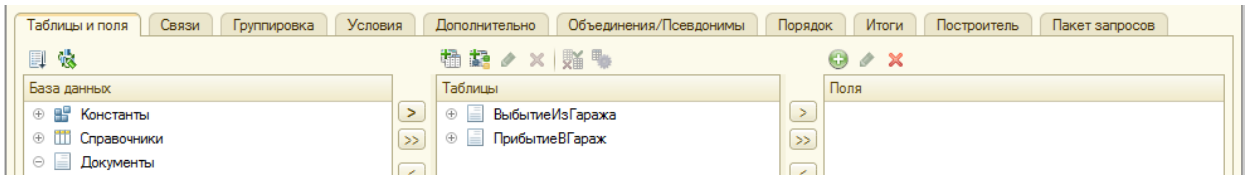


Рис. 9.2.67

Следующим шагом выберите следующие поля: поле *Ссылка* документа *Прибытие в гараж,* поле *Ссылка* документа *Выбытие из гаража*, поле *Автомобиль* документа *Прибытие в гараж.* И поле *Автомобиль* документа *Выбытие из гаража*.

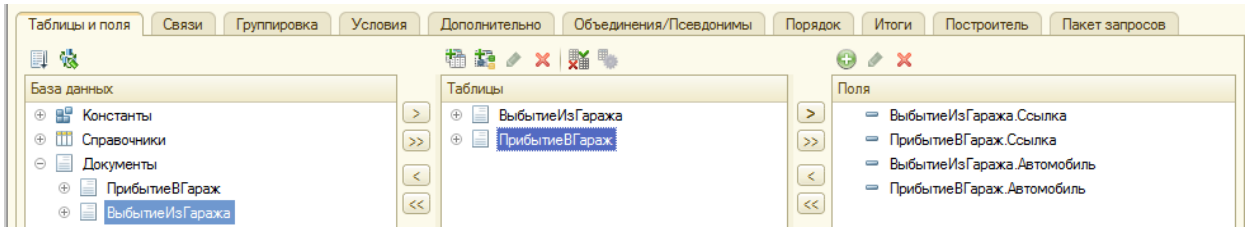


Рис. 9.2.68

Теперь перейдите на закладку *Объединения /Псевдонимы* и переименуйте поля по названию колонок в таблице значений.

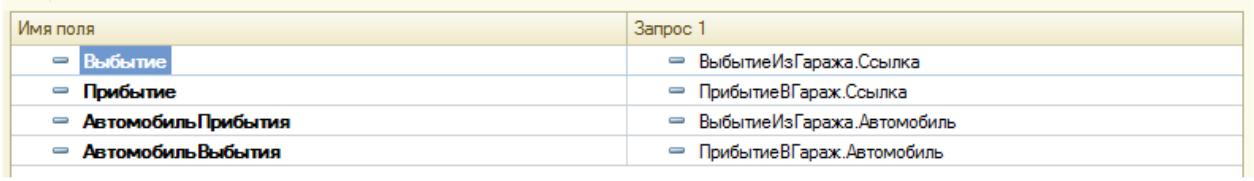


Рис. 9.2.69

Перейдите на закладку *Связи* и создайте соединение этих документов по полю *Автомобиль* и по полю *Гараж.*

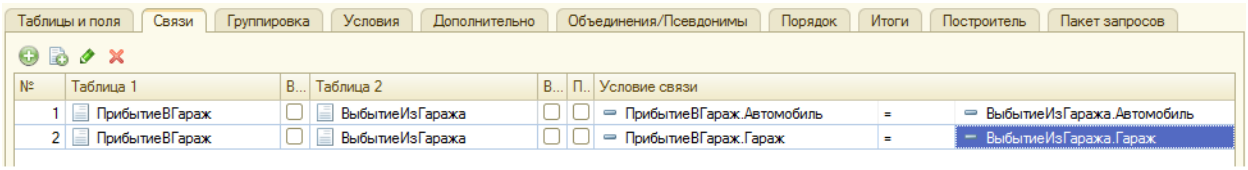


Рис. 9.2.70

Соединение будет *полное*, поэтому везде ставим флажки.

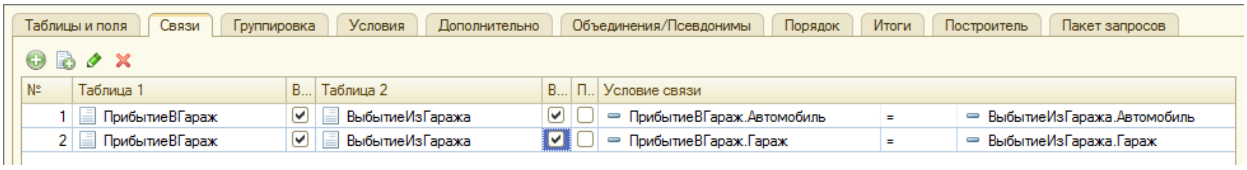


Рис. 9.2.71

В результате у нас получился следующий запрос:

```
ВЫБРАТЬ
    ВыбытиеИзГаража.Ссылка КАК Выбытие,
    ПрибытиеВГараж.Ссылка КАК Прибытие,
    ВыбытиеИзГаража.Автомобиль КАК АвтомобильПрибытия,
    ПрибытиеВГараж.Автомобиль КАК АвтомобильВыбытия
ИЗ
    Документ.ПрибытиеВГараж КАК ПрибытиеВГараж
        ПОЛНОЕ СОЕДИНЕНИЕ Документ.ВыбытиеИзГаража КАК ВыбытиеИзГаража
        ПО ПрибытиеВГараж.Автомобиль = ВыбытиеИзГаража.Автомобиль
            И ПрибытиеВГараж.Гараж = ВыбытиеИзГаража.Гараж
```

Листинг 9.2.29

Самостоятельно напишите теперь код, который будет заполнять Вашу таблицу. Сохраните обработку и посмотрите, как она работает.

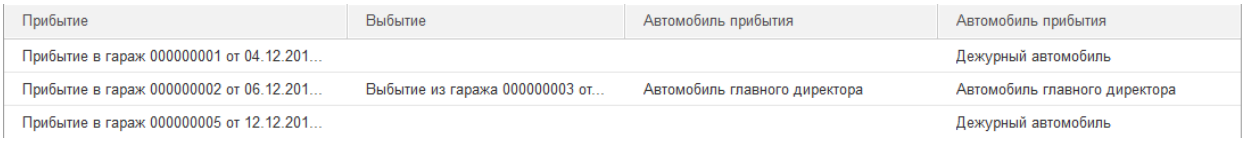


Рис. 9.2.72

Самостоятельно теперь попробуйте поменять на внутреннее, левое или правое соединение в запросе и посмотрите, что получилось.

Группировка данных

Разберем такую интересную возможность в запросах программы 1С, как *Группировка данных*.

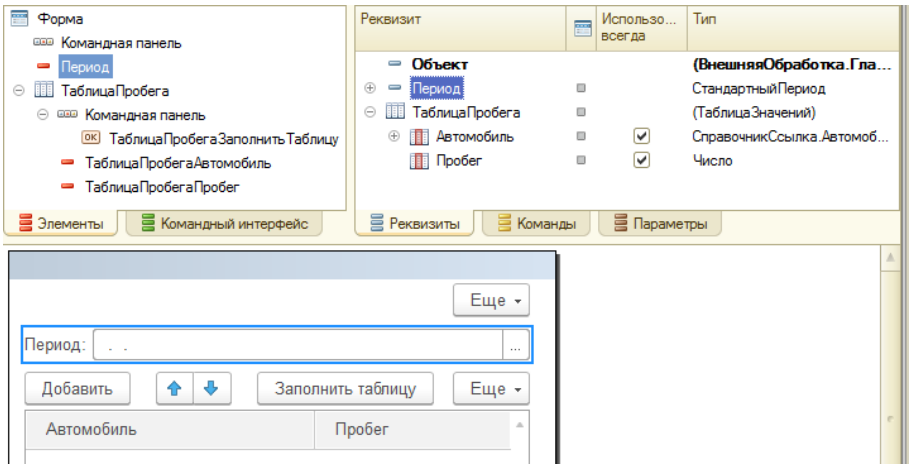


Рис. 9.2.73

Поставим следующую задачу: необходимо узнать, сколько километров проехал каждый автомобиль в сумме за какой-либо период.

Для решения будем использовать документ *Прибытие в гараж*, у которого есть реквизит *Пробег*. *Данный пример имеет чисто академический характер. Нельзя получать какое-то агрегатное значение по документам. Для этого нужно использовать регистры накопления!*

Для реализации задачи создайте обработку, форму этой обработки, разместите на ней реквизит *Период* с типом *СтандартныйПериод* и реквизит *ТаблицаПробега* с типом *ТаблицаЗначений* и колонками *Автомобиль* (тип *СправочникСсылка.Автомобили*) и *Пробег* (тип Число(10,2). И разместите это все на форме (см. рис. 9.2.73). А также создайте команду «Заполнить таблицу», которую разместите в командной панели таблицы формы.

В серверном обработчике команды «Заполнить таблицу» создадим объект «Запрос» с пустым текстом, откроем конструктор запроса и выберем таблицу *Прибытие в гараж*, а поля - *Автомобиль* и *Пробег*.

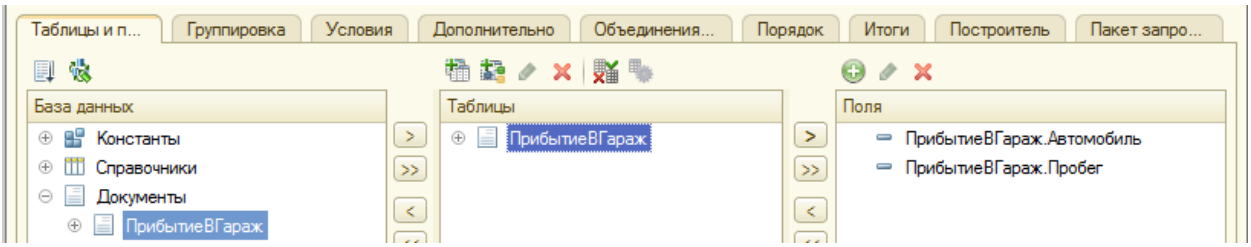


Рис. 9.2.74

Если мы на этом остановимся, то в наш список будет выведено множество одинаковых автомобилей, с разными пробегами, которые соответствуют разным документам. Нам это не

очень интересно, так как нужно знать общую цифру по каждому автомобилю. Для этого перейдите на закладку *Группировка*.

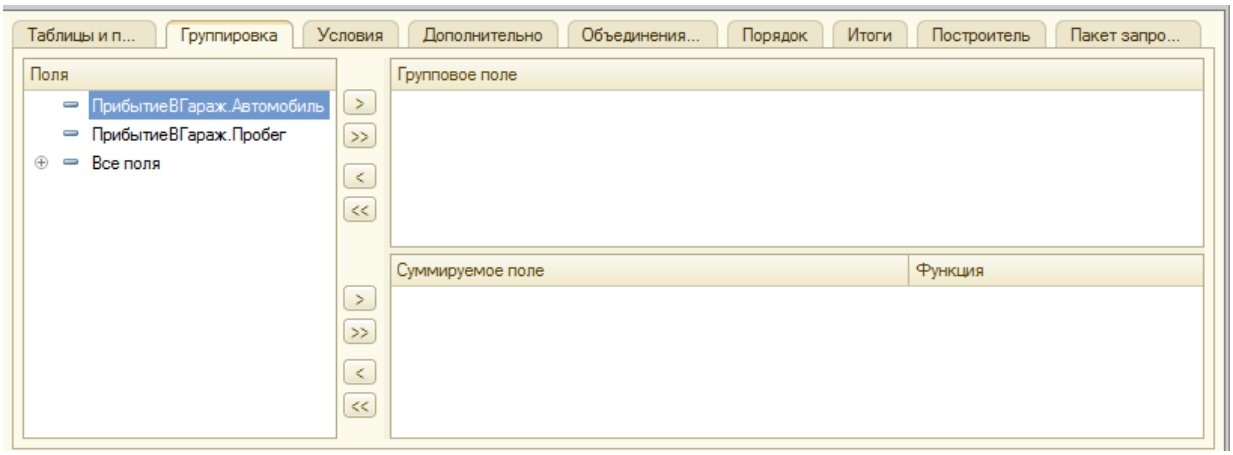


Рис. 9.2.75

В этой закладке три окна: *Общее с полями* (слева), *Поля для групповых полей* (справа, верх), поместите туда поле *Автомобиль* и *Поля для суммовых полей* (справа вниз), по ним будет какой-то итог. Поместите в него поле *Пробег.*

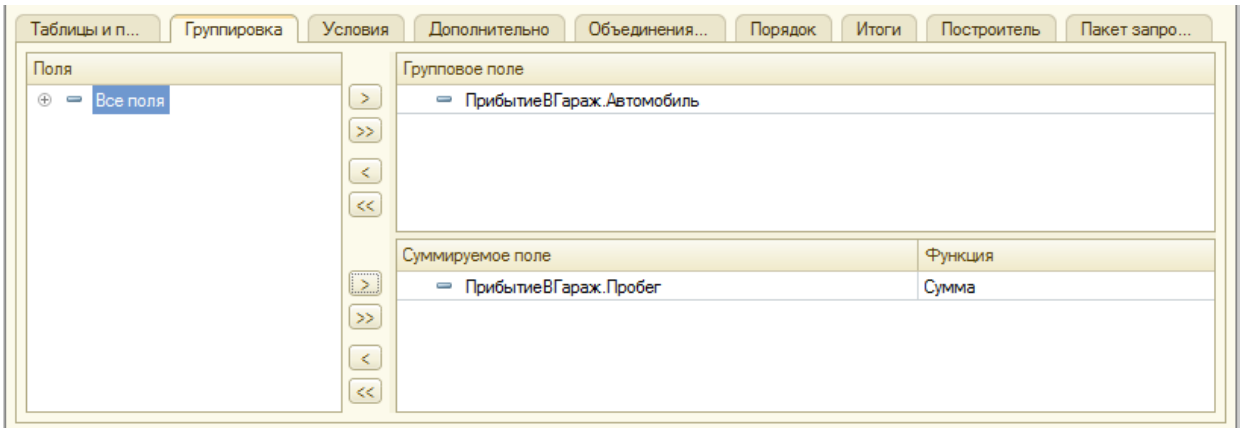


Рис. 9.2.76

Следующим шагом Вам необходимо выбрать агрегатную функцию.

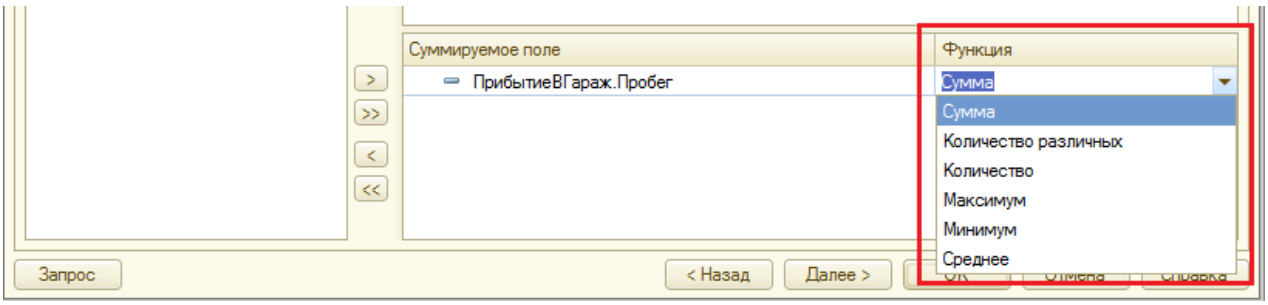


Рис. 9.2.77

Это может быть: сумма всех пробегов данного автомобиля, количество разных пробегов или общее количество, максимальный или минимальный пробег по всем документам, среднее значение. По условиям задачи выберем функцию *Сумма*.

Обращаю Ваше внимание, что в поле групповых полей должны быть все поля, которые выбраны в запрос, но которые не выбраны в суммовые поля. Если Вы этого не сделаете, то за Вас это сделает компилятор при сохранении запроса. Поэтому если Вы выберете еще и поле *Ссылка*, то оно тоже попадет в группировку, и тогда никаких итоговых цифр по автомобилю не будет, т.к. выйдут записи в связке *Автомобиль* плюс *Документ*, и по этой связке будут вычислены суммы.

После этого пройдем в *Условия* и поставим ограничения по дате.

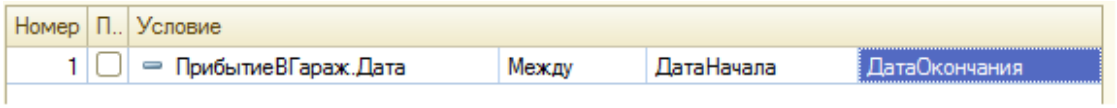


Рис. 9.2.78

Нажимаем кнопку *ОК* конструктора запросов и получаем запрос. Он будет выглядеть следующим образом:

```
ВЫБРАТЬ
    ПрибытиеВГараж.Автомобиль КАК Автомобиль,
    СУММА(ПрибытиеВГараж.Пробег) КАК Пробег
ИЗ
    Документ.ПрибытиеВГараж КАК ПрибытиеВГараж
ГДЕ
    ПрибытиеВГараж.Дата МЕЖДУ &ДатаНачала И &ДатаОкончания

СГРУППИРОВАТЬ ПО
    ПрибытиеВГараж.Автомобиль
```

Листинг 9.2.30

Как Вы видите, появилась новая директива - *«СГРУППИРОВАТЬ ПО»*, которая указывает, по какому полю будет осуществлена группировка. Кроме того, появилась функция *«СУММА»*, которая суммирует значения нужного нам поля.

Теперь напишите код, выводящий полученные данные в таблицу.

```
Запрос.УстановитьПараметр("ДатаНачала",Период.ДатаНачала);
Запрос.УстановитьПараметр("ДатаОкончания", Период.ДатаОкончания);

Результат = Запрос.Выполнить();
Если Результат.Пустой() Тогда
    Возврат;
КонецЕсли;

Выборка = Результат.Выбрать();
Пока Выборка.Следующий() Цикл
   НовСтр = ТаблицаПробега.Добавить();
   ЗаполнитьЗначенияСвойств(НовСтр,Выборка);
КонецЦикла;
```

Листинг 9.2.31

Сохраните и посмотрите на результат. Попробуйте самостоятельно поработать с другими функциями суммируемого поля.

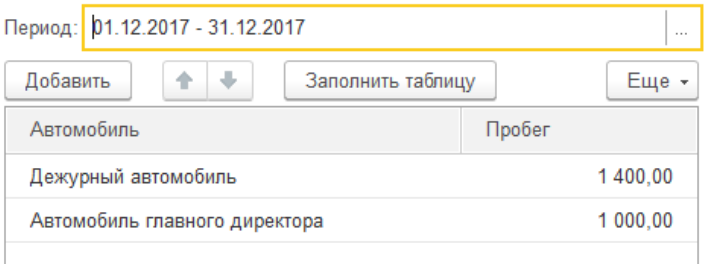


Рис. 9.2.79

Остальные функции языка запросов 1С

Напоследок рассмотрим остальные функции языка запросов 1С, которые пригодятся Вам в дальнейшей работе. Это будет не полный перечень, но наиболее часто применяемый.

И первая функция - это функция *Значение*.

Значение

Иногда необходимо задать условие, в котором поле сравнивается с каким-нибудь предопределенным значением в базе данных. Это может быть конкретный элемент перечисления, либо предопределенный элемент справочника, либо пустая ссылка справочника или документа. Для этого используется функция *Значение*.

Рассмотрим пример. Выведем все элементы справочника *Автомобили* с автоматической коробкой передач.

Создайте обработку, разместите на ней реквизит *ТаблицаАвтомобилей* с типом *ТаблицаЗначений*, который содержит две колонки – *Автомобиль* и *КоробкаПередач* с соответствующими типами. А также команду «Заполнить таблицу», для которой создайте обработчики на сервере и на клиенте.

В серверном обработчике команды создайте новый объект запрос и зайдите в конструктор запросов. Выберем справочник *Автомобили* и два поля – *Ссылка* и *КоробкаПередач* (назначьте им псевдонимы *Автомобиль* и *КоробкаПередач*) Перейдите на закладку *Условие*, выберите поле *Коробку передач* и поставьте флажок *Произвольное.*

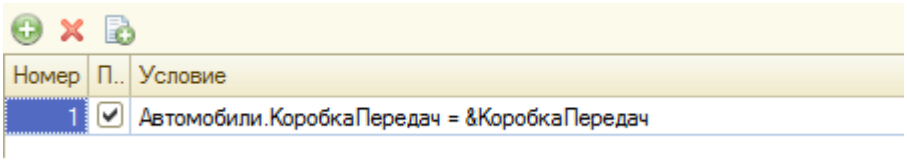


Рис. 9.2.80

И измените условие следующим образом:

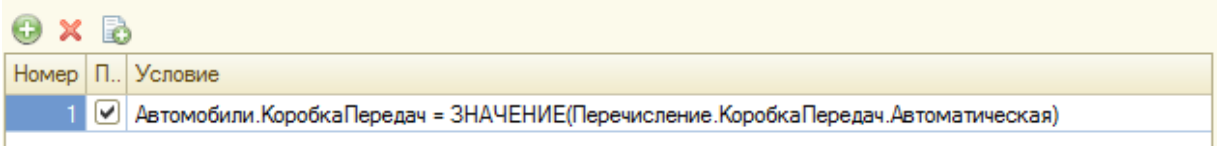


Рис. 9.2.81

Запрос написан, нажимаем кнопку *ОК*, запрос должен приобрести следующий вид:

```
ВЫБРАТЬ
    Автомобили.Ссылка КАК Автомобиль,
    Автомобили.КоробкаПередач КАК КоробкаПередач
ИЗ
    Справочник.Автомобили КАК Автомобили
ГДЕ
    Автомобили.КоробкаПередач =
ЗНАЧЕНИЕ(Перечисление.КоробкаПередач.Автоматическая)
```

Листинг 9.2.32

При помощи функции *Значение* в языке запросов можно указывать предопределенные значения справочников и планов видов характеристик. Например:

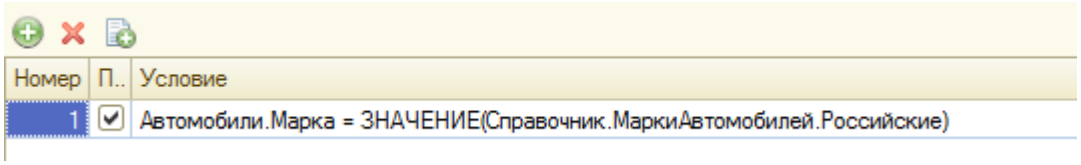


Рис. 9.2.82

А также пустую ссылку, например, мы можем отобрать все элементы справочника «Автомобиль», где не указана марка.

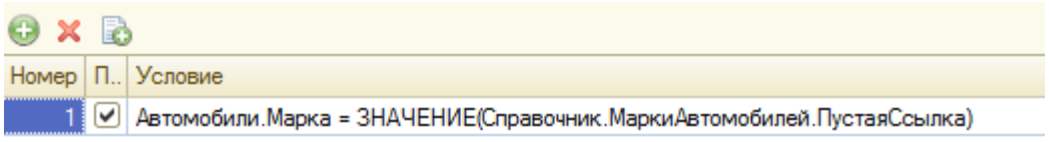


Рис. 9.2.83

Синтаксис выражения, которое нужно указать в функции *Значение*, должен быть следующим:

НазваниеГруппыМетаданных.НазваниеМетаданного.НазваниеЗначения

Где:

НазваниеГруппыМетаданных – название группы метаданных в единственном числе («Справочник», «Перечисление» и т.д.);

НазваниеМетаданного – название конкретного объекта метаданных, как указанно в конфигураторе («КоробкаПередач», «МаркиАвтомобилей» и т.д.);

НазваниеЗначения – название значения, которое может быть у конкретного метаданного, это или название какого-то значения перечисления, или название предопределенного элемента справочника или *ПустаяСсылка*.

Допишите сами вывод данных в таблицу значений. Посмотрите, как выполняется Ваш запрос. У Вас должны выйти только автомобили с автоматической коробкой передач.

Выбор

Функция *Выбор* аналогична функции *Если* в языке программирования, она используется в полях, условиях и пр. и имеет следующий синтаксис:

```
ВЫБОР
    КОГДА <Выражение> ТОГДА <Выражение>
    КОГДА <Выражение> ТОГДА <Выражение>
    ……
    ИНАЧЕ <Выражение>
КОНЕЦ
```

Вы видите, что может быть указано неограниченное количество условий *КОГДА*.

Рассмотрим пример: будем выводить документы *Прибытие в гараж* вместе со значением реквизита *Пробег*, и если значение этого реквизита больше *100*, то в поле должна отражаться строка *Пробег завышен*.

Самостоятельно создайте форму, нужные реквизиты и все остальное (для колонки, в которую будете выводить пробег, установите составной тип: число и строка). Тут мы разберем только запрос.

Зайдите в конструктор. Выберете таблицу *ПрибытиеВГараж*, поля *Ссылка*, *Автомобиль* и *Пробег*. Для поля *Ссылка* укажите псевдоним *ПрибытиеВГараж*.

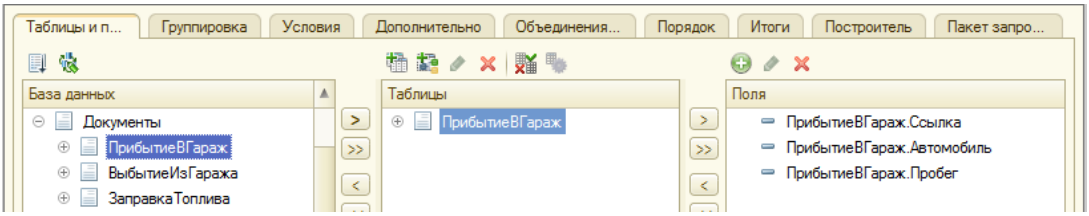


Рис. 9.2.84

Выделите поле *Пробег* и нажмите кнопку *Изменить поле*.

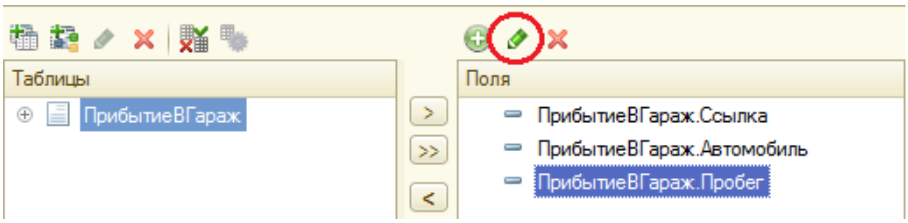


Рис. 9.2.85

Открылось окно, в котором Вы напишете наше условие.

```
ВЫБОР
    КОГДА ПрибытиеВГараж.Пробег > 100
        ТОГДА "Пробег завышен"
    ИНАЧЕ ПрибытиеВГараж.Пробег
КОНЕЦ
```

Листинг 9.2.33

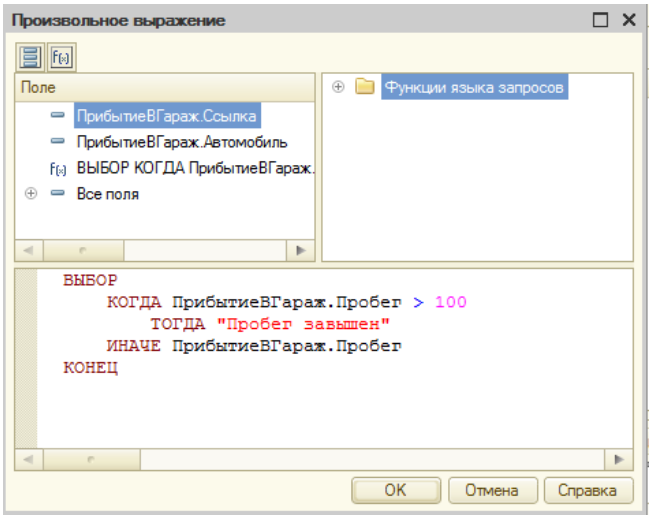


Рис. 9.2.86

Не забудьте - зайдите в закладку *Объединения/Псевдонимы* и поменяйте название этого поля на *Пробег* (после написания условия название сбросится). Самостоятельно напишите код вывода в таблицу и посмотрите на результат.

Прибытие в гараж	Автомобиль	Пробег
Прибытие в гараж 000000001 от 04.12.2017 12:00:00	Дежурный автомобиль	Пробег завышен
Прибытие в гараж 000000002 от 06.12.2017 12:00:00	Автомобиль главного ди...	75
Прибытие в гараж 000000005 от 12.12.2017 12:53:24	Дежурный автомобиль	50
Прибытие в гараж 000000006 от 12.12.2017 12:54:57	Дежурный автомобиль	100
Прибытие в гараж 000000007 от 12.12.2017 0:00:00	Дежурный автомобиль	100
Прибытие в гараж 000000008 от 12.12.2017 0:00:00	Дежурный автомобиль	50

Рис. 9.2.87

Ссылка

Рассмотрим еще одну функцию языка запроса - это *Ссылка*. *Ссылка* - это логический оператор для проверки поля на конкретную ссылку. Обычно применяется, когда некоторое поле может иметь составной тип.

Например, будем выводить журнал документов *Прибытие Выбытие автомобилей*, где если документ имеет тип *ДокументСсылка.ПрибытиеВГараж,* то напишем текст *«Прибытие»*, а иначе (*ДокументСсылка.ВыбытиеИзГаража*) - текст *«Выбытие»*.

Самостоятельно создайте обработку, форму, команду формы, таблицу значений в качестве реквизита с двумя колонками: *Документ* (тип *Строка*), и *Ссылка* (составной тип *ДокументСсылка.ПрибытиеВГараж* и *ДокументСсылка.ВыбытиеИзГаража*). Создайте в обработчике команды объект *Запрос.* Зайдите в конструктор запроса. Выберете таблицу журнал *Прибытие и Выбытие автомобилей*, и поле *Ссылка.*

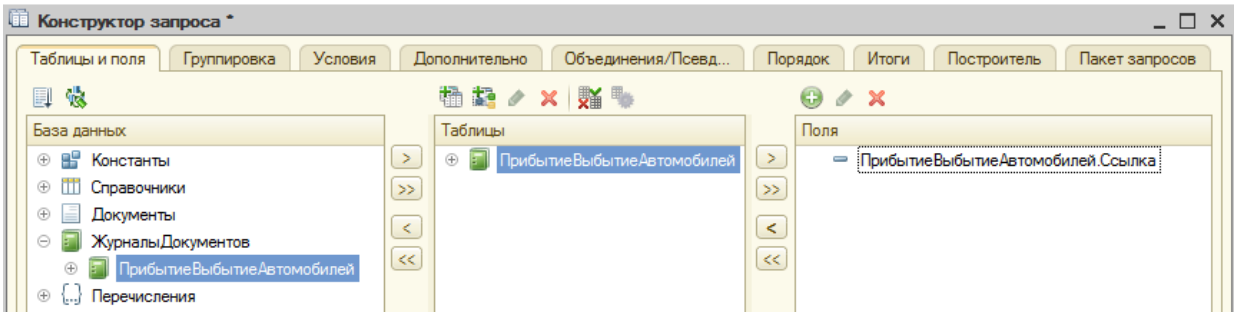


Рис. 9.2.88

Создайте новое поле вручную. Для этого нажмите на кнопку *Добавить* в окне *Поля*.

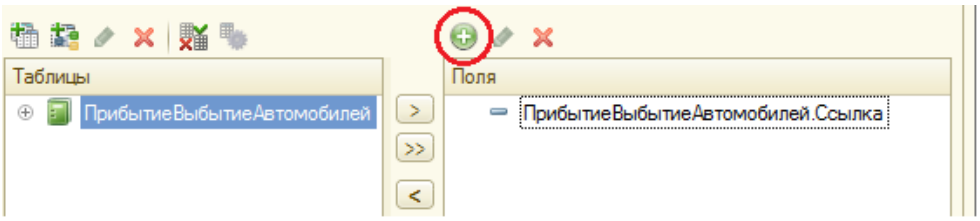


Рис. 9.2.89

В открывшемся поле напишите следующий код:

```
ВЫБОР
    КОГДА ПрибытиеВыбытиеАвтомобилей.Ссылка ССЫЛКА Документ.ПрибытиеВГараж
        ТОГДА "Прибытие"
    КОГДА ПрибытиеВыбытиеАвтомобилей.Ссылка ССЫЛКА Документ.ВыбытиеИзГаража
        ТОГДА "Выбытие"
КОНЕЦ
```

Листинг 9.2.34

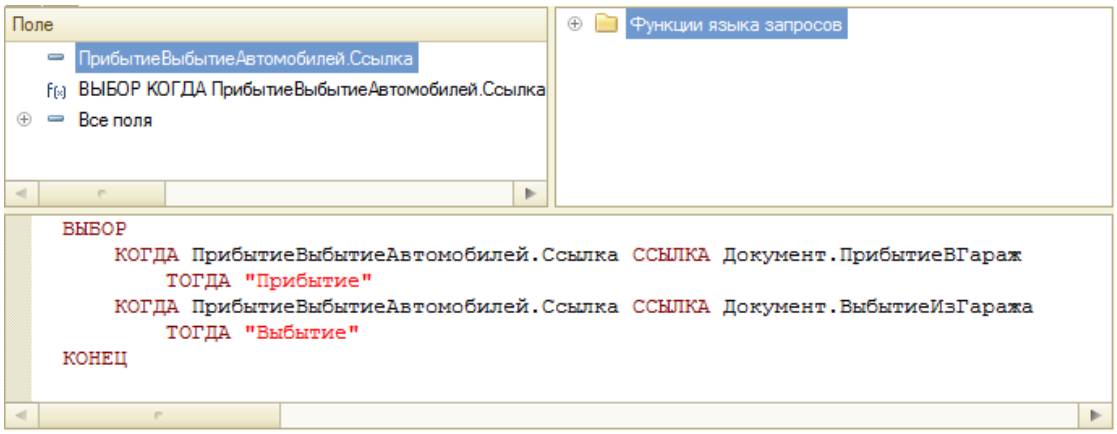


Рис. 9.2.90

Сохраните запрос. Самостоятельно напишите код вывода данных в табличное поле и посмотрите, что получится.

Документ	Ссылка
Прибытие	Прибытие в гараж 000000001 от 04.12.2017 12:00:00
Выбытие	Выбытие из гаража 000000003 от 08.12.2017 12:00:00
Прибытие	Прибытие в гараж 000000002 от 06.12.2017 12:00:00

Рис. 9.2.91

Синтаксис написания выражения после ключевого слова «ССЫЛКА» следующий:

НазваниеГруппыМетаданных.НазваниеМетаданного

Где:

НазваниеГруппыМетаданных – название группы метаданных в единственном числе («Справочник», «Документ» и т.д.);

НазваниеМетаданного – название конкретного объекта метаданных, как указанно в конфигураторе («ПрибытиеВГараж», «МаркиАвтомобилей» и т.д.);

При помощи функции *Ссылка* можно также устанавливать различные условия. Доработаем предыдущий запрос: выведем только документы *ВыбытиеИзГаража*.

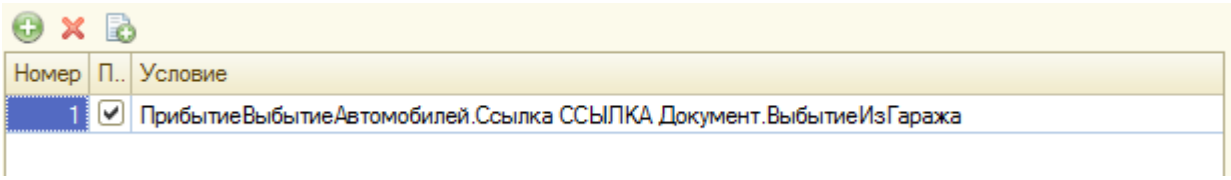


Рис. 9.2.92

Должен получиться следующий запрос:

```
ВЫБРАТЬ
    ПрибытиеВыбытиеАвтомобилей.Ссылка КАК Ссылка,
    ВЫБОР
        КОГДА ПрибытиеВыбытиеАвтомобилей.Ссылка ССЫЛКА
Документ.ПрибытиеВГараж
            ТОГДА "Прибытие"
        КОГДА ПрибытиеВыбытиеАвтомобилей.Ссылка ССЫЛКА
Документ.ВыбытиеИзГаража
            ТОГДА "Выбытие"
    КОНЕЦ КАК Документ
ИЗ
    ЖурналДокументов.ПрибытиеВыбытиеАвтомобилей КАК
ПрибытиеВыбытиеАвтомобилей
ГДЕ
    ПрибытиеВыбытиеАвтомобилей.Ссылка ССЫЛКА Документ.ВыбытиеИзГаража
```

Листинг 9.2.35

Проверьте самостоятельно работу этого запроса.

Есть NULL

Рассмотрим следующую функцию, которая очень полезна в практической работе, - это функция *Есть null*.

Для рассмотрения этой функции возьмем запрос, в котором мы соединяли таблицу «МаркиАвтомобилей» с таблицей «МоделиАвтомобилей» (см. стр. 577). У нас получился следующий запрос:

```
ВЫБРАТЬ
   МаркиАвтомобилей.Ссылка КАК Марка,
   МоделиАвтомобилей.Ссылка КАК Модель
ИЗ
   Справочник.МаркиАвтомобилей КАК МаркиАвтомобилей
       ЛЕВОЕ СОЕДИНЕНИЕ Справочник.МоделиАвтомобилей КАК МоделиАвтомобилей
       ПО МаркиАвтомобилей.Ссылка = МоделиАвтомобилей.Владелец
ГДЕ
   НЕ МаркиАвтомобилей.ПометкаУдаления
```

Листинг 9.2.36

Как Вы должны помнить, в том случае, когда у марки не было моделей, выходили пустые поля. И в этих полях находится значение null, т.е. отсутствие каких-либо значений. Такое возникает, как правило, при различных соединениях таблиц (левых, правых, полных), когда не найдены соответствия по полям. С помощью функции *есть null* можно это значение обработать.

Скопируйте обработку, где Вы соединяли таблицы *МаркаАвтомобиля* и *МодельАвтомобиля*. В реквизите с типом *ТаблицаЗначений* для колонки *Модель* сделайте составной тип (*Строка* и *СправочникСсылка.МоделиАвтомобилей*).

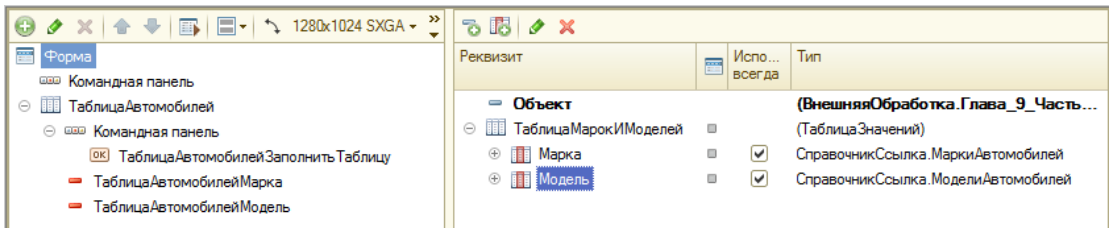


Рис. 9.2.93

Откройте конструктор уже существующего запроса и на закладке «Таблицы и поля» в окне *Поля* выделите поле *МоделиАвтомобилей.Ссылка* и нажмите кнопку «Изменить текущий элемент».

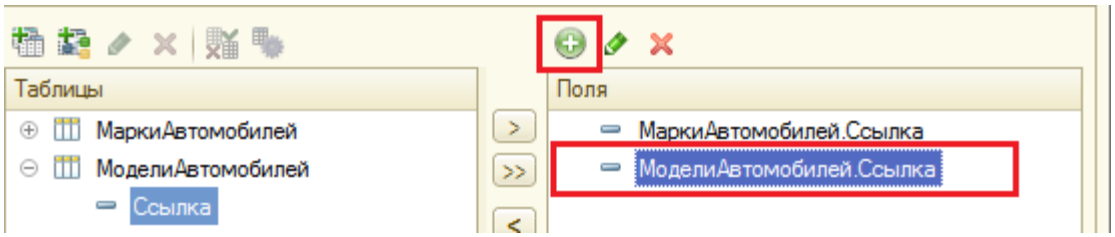


Рис. 9.2.94

В открывшемся окне напишите следующий код:

```
ВЫБОР
    КОГДА МоделиАвтомобилей.Ссылка ЕСТЬ NULL
        ТОГДА "Нет моделей у этой марки"
    ИНАЧЕ МоделиАвтомобилей.Ссылка
КОНЕЦ
```

Листинг 9.2.37

У Вас должен получиться следующий запрос:

```
ВЫБРАТЬ
    МаркиАвтомобилей.Ссылка КАК Марка,
    ВЫБОР
        КОГДА МоделиАвтомобилей.Ссылка ЕСТЬ NULL
            ТОГДА "Нет моделей у этой марки"
        ИНАЧЕ МоделиАвтомобилей.Ссылка
    КОНЕЦ КАК Модель
ИЗ
    Справочник.МаркиАвтомобилей КАК МаркиАвтомобилей
        ЛЕВОЕ СОЕДИНЕНИЕ Справочник.МоделиАвтомобилей КАК МоделиАвтомобилей
        ПО МаркиАвтомобилей.Ссылка = МоделиАвтомобилей.Владелец
ГДЕ
    НЕ МаркиАвтомобилей.ПометкаУдаления
```

Листинг 9.2.38

Сохраните обработку и посмотрите на результат её работы.

Марка	Модель
Иномарка	Нет моделей у этой марки
Российские	Нет моделей у этой марки
Ford	Focus
Ford	Consul
KIA	Rio
Renault	Нет моделей у этой марки

Рис. 9.2.95

Мы получили гораздо более красивую картинку, чем было раньше. Функция *Есть null* - это метод сравнения, который возвращает *Истину*, если поле содержит значение *null*, и *Ложь* - в противном случае.

ЕстьNULL

Выражение *Есть NULL* возвращает либо *Истину*, либо *Ложь*. Но есть еще функция, которая, по сути, позволяет нам не использовать директиву *«ВЫБОР»* для подобных сравнений. Это функция ***ЕстьNull***.

Заменим в предыдущем запросе директиву *«ВЫБОР»* на данную функцию. Для этого зайдите обратно в конструктор запроса и в соответствующее поле.

Измените данное поле.

```
ЕСТЬNULL(МодельАвтомобиля.Ссылка, "Нет моделей у этой марки")
```

Листинг 9.2.39

Сохраните обработку и посмотрите на результат. Результат будет тот же самый.

Рассмотрим синтаксис данной функции:

ЕСТЬNULL(<Поле для сравнения>, <Значение вместо null>)

Данная функция проверяет первый параметр, и если он не равен *NULL*, то возвращает его, иначе возвращает второй параметр.

Я рекомендую Вам использовать функцию *ЕстьNull* для замены пустых значений в полях, но метод сравнения *Есть Null* Вам может пригодиться в условиях.

Резюме

На этом мы закончим изучение языка запросов платформы 1С. Я дал Вам довольно много для начинающих, но, надеюсь, Вы с легкостью освоите эту главу, т.к. все было предоставлено в виде маленьких шагов и на наглядных примерах.

Конечно, есть еще и более интересные вещи в запросах, такие как временные таблицы и передача таблиц значений в запрос, Вы это сможете изучить самостоятельно или с помощью моего видео-курса «Запросы в 1С для начинающих», у Вас, как у покупателя этой книги есть промо-код на скидку в 25 % - hrW0rl9Nnx.

Глава 10. Регистры сведений и накоплений

Десятую главу мы посвятим регистрам сведений и регистрам накоплений. Мы научимся создавать, изменять и удалять записи регистров сведений и накоплений программным способом. Научимся получать выборку записей регистров и работать с ними в запросах.

Начнем мы нашу главу с регистров сведений.

Часть 1. Регистры сведений. Работа с записями

Если Вы позабыли, то напомню. *Регистры сведений* в системе 1С призваны хранить информацию, которая используется в прикладной задаче. Данная информация может изменяться со временем, тогда регистр будет *периодическим*, либо быть постоянной, в этом случае мы имеем *непериодический* регистр сведений. Например, информация о привязке автомобиля к гаражу не изменяется во времени, но информация о ценах на топливо изменяется каждый день (и не по разу).

Автомобиль	Гараж
Авто мастера	Офисная стоянка "Центральная"
Дежурный автомобиль	Офисная стоянка "Северная"
Автомобиль главного директора	Гараж №1-1

Рис. 10.1.1

Цены на топливо

Поиск (Ctrl+F) Еще

Период	Регистратор	Но...	Тип топл...	Поставщик топ...	Цена
27.09.2017 12:00:30	Установка цен на топливо 000000001 от 27.09.2017 12:00:30	1	Аи 92	ТопливоОйл	36,00
27.09.2017 12:00:30	Установка цен на топливо 000000001 от 27.09.2017 12:00:30	2	Аи 95	ТопливоОйл	36,00
27.09.2017 12:00:30	Установка цен на топливо 000000002 от 27.09.2017 12:00:30	1	Аи 92	ТопливоОйл	37,00
27.09.2017 12:00:30	Установка цен на топливо 000000002 от 27.09.2017 12:00:30	2	Аи 95	ТопливоОйл	37,00
27.12.2017 18:49:43	Установка цен на топливо 000000003 от 27.12.2017 18:49:43	1	Аи 92	ТопливоОйл	13,00
27.12.2017 18:49:43	Установка цен на топливо 000000003 от 27.12.2017 18:49:43	2	Аи 95	ТопливоОйл	15,00
27.12.2017 18:49:43	Установка цен на топливо 000000004 от 27.12.2017 18:49:43	1	Аи 92	НефтьСэйлл	18,00
27.12.2017 19:08:36	Установка цен на топливо 000000005 от 27.12.2017 19:08:36	1	Аи 92	ТопливоОйл	13,00
27.12.2017 19:08:36	Установка цен на топливо 000000005 от 27.12.2017 19:08:36	2	Аи 95	ТопливоОйл	15,00
27.12.2017 19:08:36	Установка цен на топливо 000000006 от 27.12.2017 19:08:36	1	Аи 92	НефтьСэйлл	18,00

Рис. 10.1.2

Как Вы видите на картинках, во втором случае (рис. 10.1.2) в записях присутствует поле *Период*, а в первом случае (рис. 10.1.1) данного поля нет. Вспомним, где мы указываем периодичность регистра.

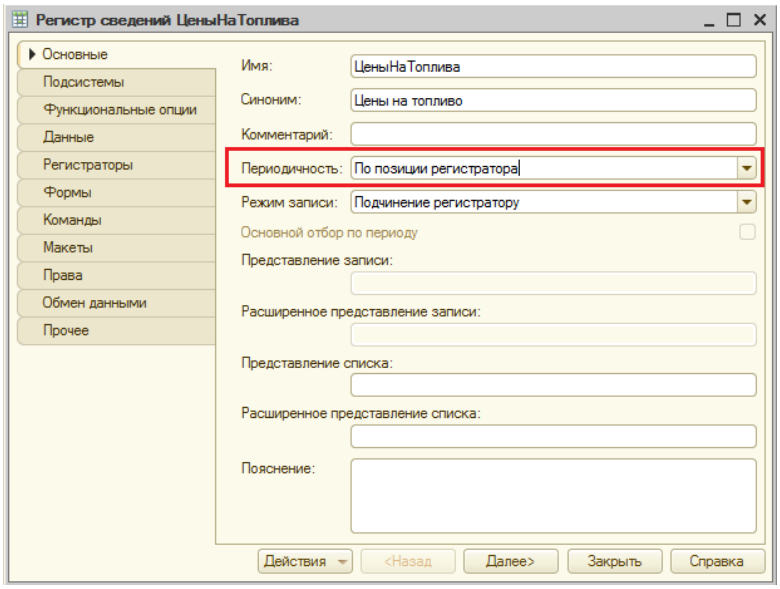


Рис. 10.1.3

Периодичность может быть разная, самая маленькая – в пределах регистратора (на одну секунду может быть несколько документов-регистраторов), самая большая - год.

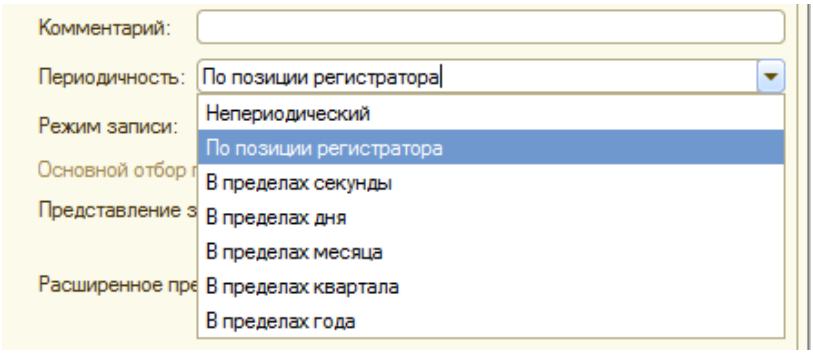


Рис. 10.1.4

Вся информация в регистрах хранится в определенных разрезах данных, которые называются *Измерениями*. У регистра должно быть хотя бы одно измерение. Например, у автомобиля может быть только один основной гараж, поэтому в регистре сведений *Основной гараж автомобиля* только одно измерение – *Автомобиль*.

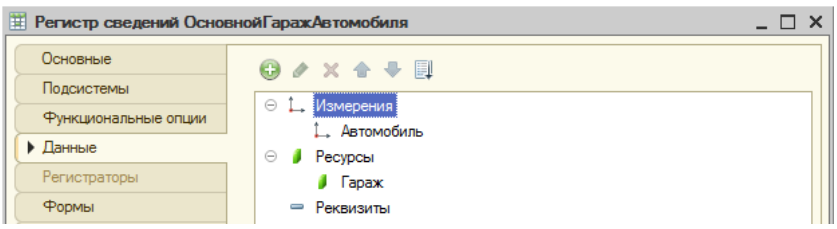


Рис. 10.1.5

А информация о ценах на топливо может храниться в двух разрезах – *Тип топлива* и *Поставщик топлива*. Поэтому у регистра *ЦеныНаТопливо* два измерения – *ТипТоплива* и *ПоставщикТоплива*.

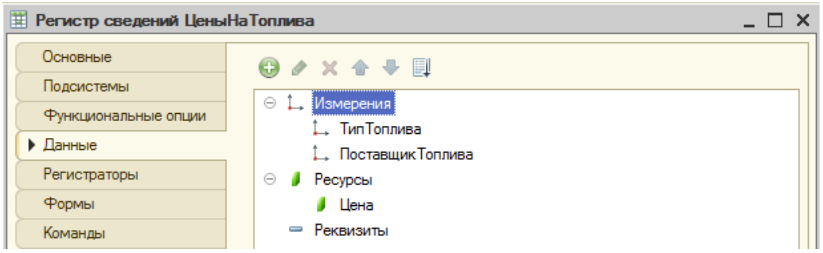


Рис. 10.1.6

Сами данные хранятся в *Ресурсах*.

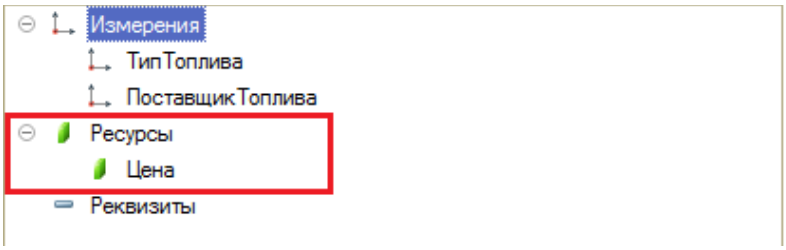


Рис. 10.1.7

Также можно создавать и использовать *Реквизиты*.

Рис. 10.1.8

Реквизиты, в отличие от ресурсов, являются не более чем справочной информацией и не возвращаются стандартными функциями работы с регистрами сведений.

Строки регистра сведений с определенным значением ресурса для определенного набора измерений называются *Записями*.

Из назначения регистра сведений выходит, что не может быть двух записей с одинаковым набором измерений (и одинаковым периодом, в случае периодических регистров). Действительно, например, не может быть для одного автомобиля два основных гаража. Если в прикладной задаче это возможно, то необходимо либо добавить еще одно измерение (например, перечисление *Лето-Зима*), либо сделать данный регистр периодическим (например, с периодичностью в год).

Теперь изучим, каким образом вносятся записи в регистр сведений. Осуществить это можно двумя способами: вручную и с помощью документов.

За это отвечает свойство регистров *Режим записи.*

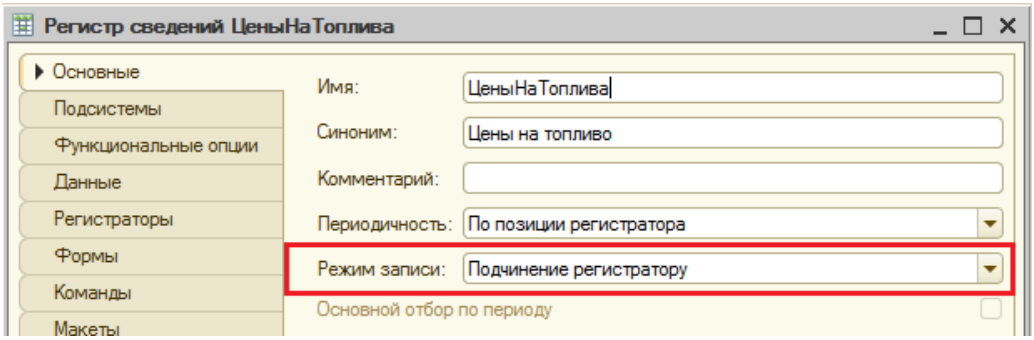


Рис. 10.1.9

В том случае, если записи вносятся вручную пользователем, мы имеем регистр сведений с *Независимым режимом записи* (как это сделано в регистре сведений *Основной гараж автомобиля*).

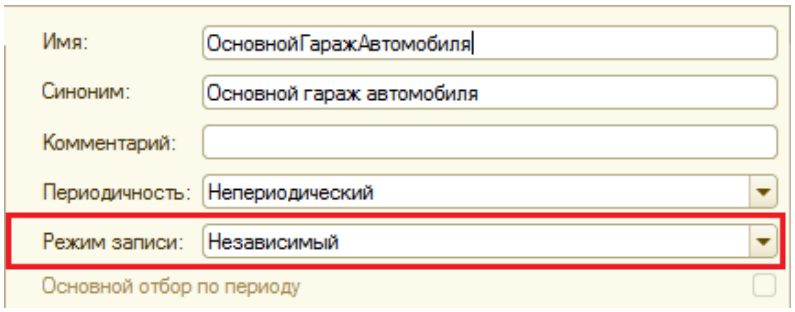


Рис. 10.1.10

Когда же записи создаются документом (регистратором), то мы имеем дело с режимом записи *Подчинение регистратору* (регистр сведений *Цены на топливо*).

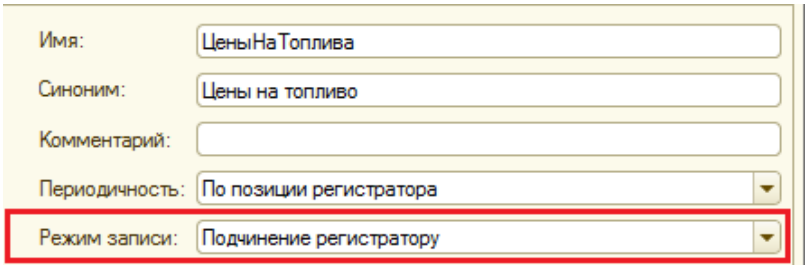
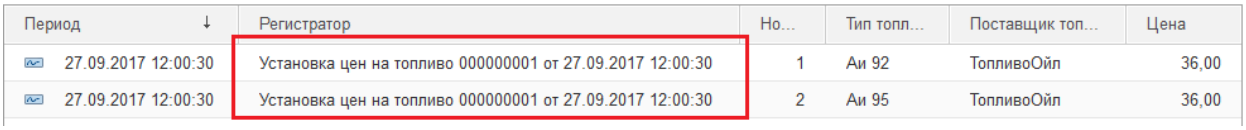


Рис. 10.1.11

Обращаю Ваше внимание, что в случае независимого регистра Вы можете редактировать любые записи вручную, в противном случае записи будут закрыты от редактирования. Документ, который будет вносить запись в подчиненный регистр сведений, называется *Регистратор*, в этом случае записи жестко подчинены регистратору.



Период	Регистратор	Но...	Тип топл...	Поставщик топ...	Цена
27.09.2017 12:00:30	Установка цен на топливо 000000001 от 27.09.2017 12:00:30	1	Аи 92	ТопливоОйл	36,00
27.09.2017 12:00:30	Установка цен на топливо 000000001 от 27.09.2017 12:00:30	2	Аи 95	ТопливоОйл	36,00

Рис. 10.1.12

Такие записи создаются при проведении документа-регистратора, соответственно при отмене проведения документа эти записи будут уничтожаться автоматически.

Итак, всю теоретическую информацию, вкратце, Вы узнали. Самостоятельно создайте периодический регистр сведений (с периодичностью в пределах месяца) *Норма расхода*, с двумя измерениями: *Автомобиль* (тип *СправчоникСсылка.Автмобили*, ведущее) и *Сезон* (для измерения *Сезон* создайте перечисление *Сезон (Зима-Лето)*) и одним ресурсом *Норма* (тип число(10,2)). Добавьте его во все подсистемы и кроме полных прав дайте права роли *УчетчикТоплива* (диспетчеру – чтение/просмотр).

После того, как Вы научились создавать регистры сведений, и вспомнили, какие их виды существуют, научимся работать с ними.

Менеджер регистров

Для того чтобы работать непосредственно с регистром сведений: создать новую запись, удалить или редактировать имеющуюся, получать выборку или срез первых (последних) и т.п., - необходимо вызывать *Менеджер регистров*, а после уже можно проводить различные вышеперечисленные манипуляции с регистром сведений.

Менеджер регистров сведений создается так же, как и менеджер справочников и документов.

МенеджерНормаРасход = РегистрыСведений.НормаРасхода;

МенеджерОсновнойГараж= РегистрыСведений.ОсновнойГаражАвтомобиля;

Узнаем, как создать новую запись регистра сведений.

Менеджер записи

Для того, чтобы программным способом создать, редактировать или удалить конкретную запись независимого регистра сведений, необходимо использовать объект *РегистрСведенийМенеджерЗаписи.* С помощью данного объекта можно получить доступ к записи с необходимым набором полей. Создается менеджер записи с помощью функции менеджера регистров *СоздатьМенеджерЗаписи*.

Для того чтобы посмотреть, как работает данный метод, создайте новую обработку, на которой разместите три реквизита: *Автомобиль*, *Сезонность* и *Норма расхода* с соответствующими типами (как у данных регистра сведений **НормаРасхода**). А также команду *СоздатьЗапись*, для которой сделайте обработчики в клиентском и серверном контексте.

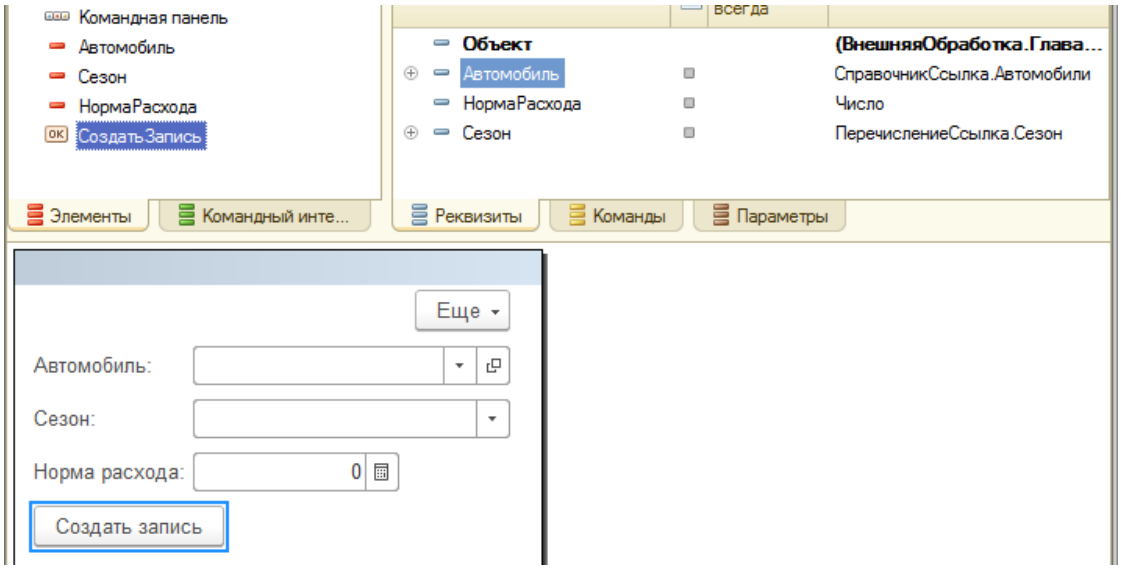


Рис. 10.1.13

Теперь в серверном обработчике команды создадим менеджер регистра сведений и менеджер записи.

```
&НаСервере
Процедура СоздатьЗаписьНаСервере()
    МенеджерНормаРасход = РегистрыСведений.НормаРасхода;
    МенеджерЗаписиНР    = МенеджерНормаРасход.СоздатьМенеджерЗаписи();
КонецПроцедуры

&НаКлиенте
Процедура СоздатьЗапись(Команда)
    СоздатьЗаписьНаСервере();
КонецПроцедуры
```

Листинг 10.1.1

Объект *МенеджерЗаписиНР*, который мы создали, имеет тип *РегистрСведенийМенеджерЗаписи*, он предназначен для чтения, редактирования и удаления конкретной записи. Мы можем обращаться к измерениям, ресурсам и реквизитам регистра сведений как к свойствам данного объекта.

Присвоим свойствам данного объекта значения реквизитов формы и запишем новую запись.

```
&НаСервере
Процедура СоздатьЗаписьНаСервере()
    МенеджерНормаРасход = РегистрыСведений.НормаРасхода;
    МенеджерЗаписиНР    = МенеджерНормаРасход.СоздатьМенеджерЗаписи();
    МенеджерЗаписиНР.Автомобиль = Автомобиль;
    МенеджерЗаписиНР.Норма = НормаРасхода;
    МенеджерЗаписиНР.Сезон = Сезон;
    МенеджерЗаписиНР.Период = ТекущаяДата();
    МенеджерЗаписиНР.Записать();
КонецПроцедуры
```

Листинг 10.1.2

Самостоятельно посмотрите, как работает Ваша обработка.

Теперь разберем этот объект (регистр сведений менеджер записей) более подробно. Как мы уже проходили, данный объект позволяет управлять записью регистра сведений и применим только для независимых регистров. Доступ к записи обеспечивается путем присвоения значений полям объекта, которые соответствуют измерениям, ресурсам и реквизитам регистра. В Вашем примере это поля *Автомобиль*, *Норма расхода* и *Период*.

Относительно периода замечу, что платформа самостоятельно изменит текущую дату на дату начала месяца (если было 21.12.2017, то запишется 01.12.2017). В данном примере мы не выясняем, есть ли уже запись с нужным набором ключевых полей, а просто записываем ее, поэтому если такая запись уже есть, то она перезапишется. Проверьте это.

Иногда это не нужно.

Как сделать так, чтобы *Норма расхода* не перезаписывалась при одинаковых измерениях и периоде? Для этого нам понадобятся два метода объекта *РегистрСведенийМенеджерЗаписи* - это метод *Прочитать* и метод *Выбран*.

Метод *Прочитать* считывает данные регистра по указанным измерениям и периоду, а метод *Выбран* возвращает *Истину*, если есть запись с указанными полями, и *Ложь,* если такой нет.

Добавьте на форму новую команду «Создать запись с проверкой», создайте клиентский и серверный обработчики этой команды. И в серверном обработчике мы будем проверять, есть ли запись по заданному набору полей, и если её нет, создадим её.

```
&НаСервере
Процедура СоздатьЗаписьСПроверкойНаСервере()
    МенеджерНормаРасход = РегистрыСведений.НормаРасхода;
    МенеджерЗаписиНР = МенеджерНормаРасход.СоздатьМенеджерЗаписи();

    МенеджерЗаписиНР.Автомобиль = Автомобиль;
    МенеджерЗаписиНР.Сезон = Сезон;
    МенеджерЗаписиНР.Период = ТекущаяДата();
    МенеджерЗаписиНР.Прочитать();
    Если Не МенеджерЗаписиНР.Выбран() тогда
        МенеджерЗаписиНР.Автомобиль = Автомобиль;
        МенеджерЗаписиНР.Норма = НормаРасхода;
        МенеджерЗаписиНР.Сезон = Сезон;
        МенеджерЗаписиНР.Период = ТекущаяДата();
        МенеджерЗаписиНР.Записать();
    КонецЕсли;
КонецПроцедуры

&НаКлиенте
Процедура СоздатьЗаписьСПроверкой(Команда)
    СоздатьЗаписьСПроверкойНаСервере();
КонецПроцедуры
```

Листинг 10.1.3

Прокомментируем данный код. Как и в прошлый раз, мы создаем объект *РегистрСведенийМенеджерЗаписи*.

Следующим шагом присваиваем значения ключевым полям и периоду. И применяем метод *Прочитать*. Данный метод считывает записи с регистра по указанным ключевым полям (измерениям) и периоду. Если есть записи с данным набором полей, то метод *Выбран* возвращает *Истину*, иначе – *Ложь*. В Вашем примере, если метод *Выбран* вернул значение *Ложь* (записей нет), то мы заново присваиваем значения измерений и ресурсов и записываем.

Проверьте, как работает новый способ записи.

Рассмотрим, как с помощью объекта *Менеджер записи* удалить запись с определенным набором ключевых полей.

Создайте новую команду на форме обработки с названием «Удалить запись», а также серверные и клиентские обработчики команды. Напишите следующий код в серверном обработчике:

```
&НаСервере
Процедура УдалитьЗаписьНаСервере()
    МенеджерНормаРасход = РегистрыСведений.НормаРасхода;
    МенеджерЗаписиНР = МенеджерНормаРасход.СоздатьМенеджерЗаписи();
    МенеджерЗаписиНР.Автомобиль = Автомобиль;
    МенеджерЗаписиНР.Сезон = Сезон;
    МенеджерЗаписиНР.Период = ТекущаяДата();
    МенеджерЗаписиНР.Прочитать();
    Если МенеджерЗаписиНР.Выбран() тогда
        МенеджерЗаписиНР.Автомобиль = Автомобиль;
        МенеджерЗаписиНР.Сезон = Сезон;
        МенеджерЗаписиНР.Период = ТекущаяДата();
        МенеджерЗаписиНР.Удалить();
    КонецЕсли;
КонецПроцедуры

&НаКлиенте
Процедура УдалитьЗапись(Команда)
    УдалитьЗаписьНаСервере();
КонецПроцедуры
```

Листинг 10.1.4

В данном коде мы также создаем объект *Регистр сведений Менеджер записи* и присваиваем измерениям значения с формы. Используя методы *Прочитать* и *Выбран,* определяем, есть ли запись с данным набором полей. И если есть, то снова присваиваем значения с полей и удаляем запись.

Проверьте работу вышеприведенного кода.

Итак, мы научились работать с определенной записью регистра сведений, используя объект *Регистр сведений Менеджер записи*. Но иногда возникает необходимость получить доступ к определенному набору записей регистра сведений и в этом наборе добавить, либо изменить, либо удалить какую-нибудь запись.

Для этого существует объект *Регистр сведений Набор записей*.

Набор записей

Для того чтобы начать знакомиться с данным объектом, решим следующую задачу: необходимо изменить норму расхода определенного автомобиля, помножив на нужный коэффициент.

Для этого создадим обработку, на форме которой создадим реквизиты *Автомобиль* (тип *СправочникСсылка.Автомобили*) и *Коэффициент* (тип *Число(10)*). А также команду «Пересчитать норму расхода», для которой создадим серверные и клиентские обработчики.

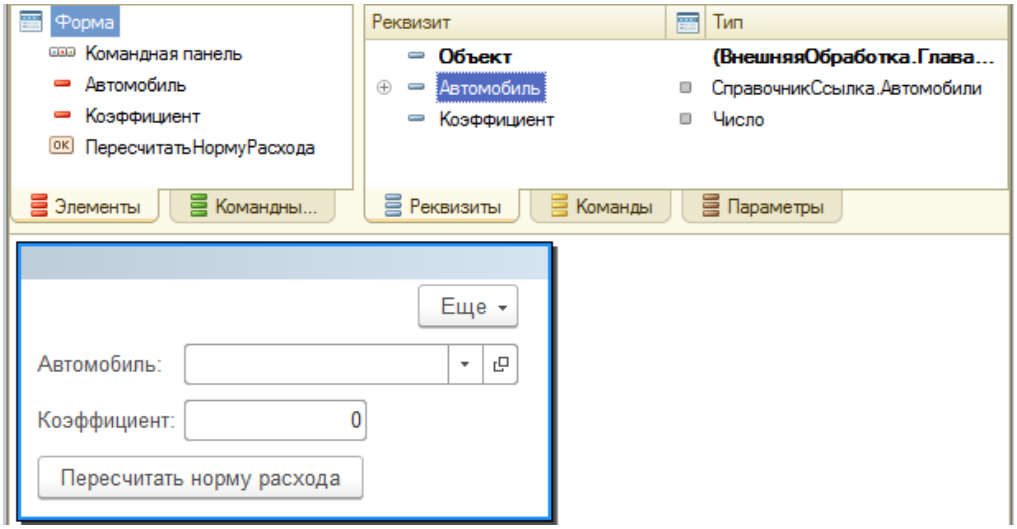


Рис. 10.1.14

В серверном обработчике команды напишем следующий код:

```
&НаКлиенте
Процедура ПересчитатьНормуРасхода(Команда)
    ПересчитатьНормуРасходаНаСервере();
КонецПроцедуры


&НаСервере
Процедура ПересчитатьНормуРасходаНаСервере()
    НормаРасхода = РегистрыСведений.НормаРасхода;
    НаборНорм = НормаРасхода.СоздатьНаборЗаписей();
    НаборНорм.Отбор.Автомобиль.Установить(Автомобиль);
    НаборНорм.Прочитать();
    Если НаборНорм.Количество() > 0 тогда
        Для Каждого НормаЗапись из НаборНорм цикл
            НормаЗапись.Норма = НормаЗапись.Норма*Коэффициент;
        КонецЦикла;
        НаборНорм.Записать();
    КонецЕсли;
КонецПроцедуры
```

Листинг 10.1.5

Разберем строки данного кода.

Первоначально мы создаем набор записей (который является отдельным объектом) при помощи метода *СоздатьНаборЗаписей* менеджера регистра сведений. Следующим шагом нам нужно сделать так, чтобы в наборе были записи, соответствующие только *нужному* нам автомобилю. Делается это с помощью свойства *Отбор* объекта *НаборЗаписей*.

Мы не будем вдаваться в подробности работы с отбором, единственно уточню, что в отборе как к свойству объекта можно обращаться ко всем измерениям регистра сведений и к периоду, если регистр периодический. Если регистр подчинен регистратору, то отбор возможен только по регистратору.

Когда мы обращаемся к свойству объекта *Отбор* посредством названия измерения, то мы получаем объект *Элемент отбора*, у данного объекта только один метод - *Установить*. Данный метод устанавливает значение отбора.

Можно производить отбор по одному полю, а можно и по нескольким.

В случае отбора по нескольким полям логическая связь между полями будет осуществляться с помощью булевой операции *«И»*.

Имейте в виду, что отбор в регистрах сведений можно устанавливать только на равенство!

После того, как мы установили отбор, нам необходимо извлечь данные из базы в объект *Набор записей*, осуществляется это с помощью метода *Прочитать*. Этот метод считывает записи из базы данных по установленному отбору и записывает их в объект *НаборЗаписей*.

Объект *Набор записей* является коллекцией объектов *Запись регистров сведений*. Не путать с объектом *Менеджер записей регистров сведений*: объект *Запись регистров сведений* не создается самостоятельно, в отличие от менеджера записей, а доступ к нему предоставляется другими объектами, в нашем случае это *Набор записей*.

Поэтому после того, как мы с помощью метода *Количество* проверили, есть ли в принципе записи по данному отбору, мы обходим данную коллекцию с помощью цикла: *Для каждого…Цикл*. Где переменная *НормаЗапись* является объектом *Запись регистров сведений*. С помощью этого объекта мы получаем доступ ко всем полям записи регистра сведений и можем их изменять на свое усмотрение. После того, как мы поменяли все нужные нам записи, необходимо записать набор. Обращаю внимание, что записывается именно набор в целом, а не конкретная запись по отдельности.

Метод *Записать* объекта *Набор записей* имеет следующий синтаксис.

Записать(<Замещать>)

Параметр *Замещать* должен иметь тип булево, и если установлено значение *Истина*, то записи по данному отбору будут удалены и вместо них записаны новые, а если *Ложь*, то будут добавлены новые записи. По умолчанию установлено значение *Истина*.

Посмотрите, как работает Ваша обработка. Создайте с помощью предыдущей обработки две записи с одним автомобилем (нормы расхода для зимы и лета соответственно).

Как Вы увидите, написанный Вами код изменяет значение поля *Норма расхода* сразу по нескольким записям.

Продолжим изучать наборы записей.

С помощью данного объекта можно не только изменять имеющиеся записи, но и добавлять новые. Это удобно использовать, когда необходимо добавить сразу несколько записей одновременно. Алгоритм добавления нескольких записей с помощью объекта *Набор записей* будет работать гораздо быстрее, чем с помощью объекта *Менеджер записи*.

Чтобы продемонстрировать это, создайте обработку, на форме которой создайте реквизит *СписокАвтомобилей* с типом *СписокЗначений* (тип значения *СправочникСсылка.Автомобили*), в который будем добавлять нужные нам автомобили, и реквизиты *Сезон* и *Норма расхода*, а также команду «Заполнить нормы расхода». Реквизит *СписокАвтомобилей* разместим на форме в виде таблицы, добавив всего одну колонку, которая соответствует реквизиту списка *Значение*.

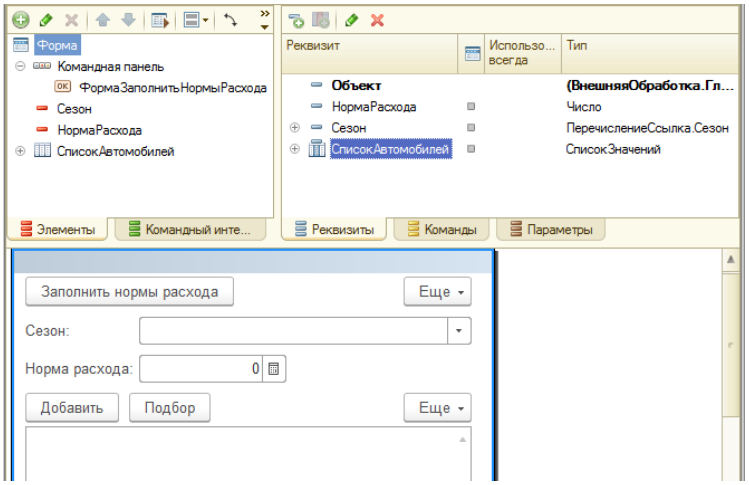


Рис. 10.1.15

При выполнении команды будем создавать набор записей регистра сведений *Норма расхода*, где каждая запись будет соответствовать автомобилю из списка.

Чтобы нельзя было создать несколько записей списка с одинаковыми автомобилями, создадим обработчик события «ОбработкаВыбора» единственной колонки таблицы (соответствует реквизиту списка «Значение»). Это событие вызывается после выбора значения в список, но до помещения выбранного значения в элемент управления.

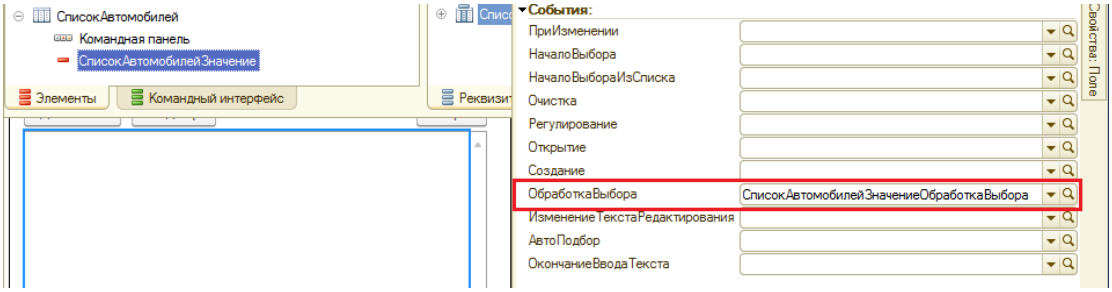


Рис. 10.1.16

Создадим клиентский обработчик команды, в котором напишем следующий код:

```
&НаКлиенте
Процедура СписокАвтомобилейЗначениеОбработкаВыбора(Элемент,
                                                    ВыбранноеЗначение,
                                                    СтандартнаяОбработка)
    Если СписокАвтомобилей.НайтиПоЗначению(ВыбранноеЗначение) <> Неопределено
Тогда
        СтандартнаяОбработка = Ложь;
    КонецЕсли;
КонецПроцедуры
```

Листинг 10.1.6

Данный код не требует дополнительных пояснений, так как работу со списком значений мы уже проходили.

После того, как мы провели всю подготовительную работу, напишем основной код в серверном обработчике команды «Заполнить нормы расхода».

```
&НаСервере
Процедура ЗаполнитьНормыРасходаНаСервере()
    Если СписокАвтомобилей.Количество() = 0 тогда
        Возврат;
    КонецЕсли;
    НаборЗаписей = РегистрыСведений.НормаРасхода.СоздатьНаборЗаписей();
    Для Каждого ЭлСписка из СписокАвтомобилей цикл
        Если Не ЗначениеЗаполнено(ЭлСписка.Значение) Тогда
            Продолжить;
        КонецЕсли;
        Автомобиль = ЭлСписка.Значение;
        НоваяЗапись = НаборЗаписей.Добавить();
        НоваяЗапись.Автомобиль = Автомобиль;
        НоваяЗапись.Норма = НормаРасхода;
        НоваяЗапись.Сезон = Сезон;
        НоваяЗапись.Период = ТекущаяДата();
    КонецЦикла;
    НаборЗаписей.Записать(Ложь);
КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьНормыРасхода(Команда)
    ЗаполнитьНормыРасходаНаСервере();
КонецПроцедуры
```

Листинг 10.1.7

Разберем вышеприведенный код. Сначала мы проверяем, есть ли элементы в списке, и если их нет, то выходим из процедуры. Следующим шагом получаем набор записей. Потом мы обходим список значений *Автомобили* циклом. Внутри цикла с помощью функции *Добавить* объекта *Регистр сведений Набор записей* создается переменная *НоваяЗапись* имеющая тип *Регистр сведений Запись*.

Свойствами данного объекта являются измерения, реквизиты и ресурсы регистра сведений. Поэтому обращаемся к ним напрямую и присваиваем нужные значения.

После того как обошли цикл, сохраняем набор записей.

Причем параметр *Замещать* мы ставим в *Ложь*, чтобы не удалялись старые записи. У данного решения есть один существенный недостаток – если уже есть запись с такими ключевыми полями, то будет сгенерирована ошибка.

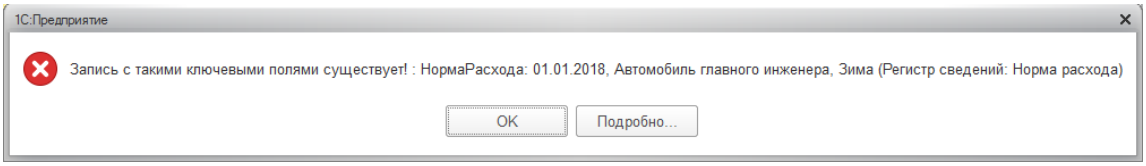


Рис. 10.1.17

Поэтому групповая запись в регистр сведений без использования отборов осуществима только для начального заполнения пустого регистра. В иных случаях необходимо работать с отборами.

Создайте новую команду «Заполнить нормы расхода с проверкой», в серверном обработчике которой напишите следующий код:

```
&НаСервере
Процедура ЗаполнитьНормыРасходаСПроверкойНаСервере()
    Если СписокАвтомобилей.Количество() = 0 тогда
        Возврат;
    КонецЕсли;
    НаборЗаписей = РегистрыСведений.НормаРасхода.СоздатьНаборЗаписей();
    Для Каждого ЭлСписка из СписокАвтомобилей цикл
        Автомобиль = ЭлСписка.Значение;
        НаборЗаписей.Отбор.Автомобиль.Установить(Автомобиль);
        НаборЗаписей.Отбор.Сезон.Установить(Сезон);
        НаборЗаписей.Отбор.Период.Установить(НачалоМесяца(ТекущаяДата()));
        НаборЗаписей.Прочитать();
        Если НаборЗаписей.Количество() = 0 тогда
            НоваяЗапись = НаборЗаписей.Добавить();
            НоваяЗапись.Автомобиль = Автомобиль;
            НоваяЗапись.Норма = НормаРасхода;
            НоваяЗапись.Сезон = Сезон;
            НоваяЗапись.Период = ТекущаяДата();
            НаборЗаписей.Записать(Истина);
        КонецЕсли;
    КонецЦикла;
КонецПроцедуры
```

Листинг 10.1.8

В этом коде мы с помощью отборов получаем определенный набор записи, который должен соответствовать установленным отборам. И если данных по установленным отборам нет, то создаем новую запись.

Имейте ввиду, что после того, как вы примените метод *Прочитать* к набору записей, то состав набора изменится согласно установленному отбору.

С помощью набора записей можно не только создавать новые записи, но и удалять имеющиеся. Дополните в Вашу форму новую команду «Удалить набор записей», при выполнении которой будут удаляться все записи, которые соответствуют автомобилям из списка, сезону и текущему месяцу.

В серверном обработчике команды напишите следующий код:

```
&НаСервере
Процедура УдалитьНаборЗаписейНаСервере()
    Если СписокАвтомобилей.Количество() = 0 тогда
        Возврат;
    КонецЕсли;
    НаборЗаписей = РегистрыСведений.НормаРасхода.СоздатьНаборЗаписей();
    Для Каждого ЭлСписка из СписокАвтомобилей цикл
        Автомобиль = ЭлСписка.Значение;
        НаборЗаписей.Отбор.Автомобиль.Установить(Автомобиль);
        НаборЗаписей.Отбор.Сезон.Установить(Сезон);
        НаборЗаписей.Отбор.Период.Установить(НачалоМесяца(ТекущаяДата()));
        НаборЗаписей.Прочитать();
        НаборЗаписей.Очистить();
        НаборЗаписей.Записать();
    КонецЦикла;
КонецПроцедуры
```

Листинг 10.1.9

Разберем этот код. Как и в предыдущем алгоритме, мы проходим по списку значений, и внутри цикла производим отбор по автомобилю, после чего считываем данные из базы, очищаем считанный набор и в конце записываем уже пустой набор данных. Также можно удалить конкретный элемент набора записей. Изменим наш код: будем с помощью отборов считывать записи по всем автомобилям из списка (со всеми сезонами и периодами), а удалим только те записи, у которых значение измерения *Сезон* равно реквизиту *Сезон формы*.

```
&НаСервере
Процедура УдалитьНаборЗаписей2НаСервере()
    Если СписокАвтомобилей.Количество() = 0 тогда
        Возврат;
    КонецЕсли;
    НаборЗаписей = РегистрыСведений.НормаРасхода.СоздатьНаборЗаписей();
    Для Каждого ЭлСписка из СписокАвтомобилей цикл
        Автомобиль = ЭлСписка.Значение;
        НаборЗаписей.Отбор.Автомобиль.Установить(Автомобиль);
        НаборЗаписей.Прочитать();
        Если НаборЗаписей.Количество() = 0 Тогда
            Продолжить;
        КонецЕсли;
        МассивНаУдаление = Новый Массив;
        Для Каждого Запись из НаборЗаписей цикл
            Если Запись.Сезон = Сезон Тогда
                МассивНаУдаление.Добавить(Запись);
            КонецЕсли;
        КонецЦикла;
        Если МассивНаУдаление.Количество() > 0 Тогда
            Для н = 0 по МассивНаУдаление.ВГраница() цикл
                НаборЗаписей.Удалить(МассивНаУдаление[н]);
            КонецЦикла;
            НаборЗаписей.Записать();
        КонецЕсли;
    КонецЦикла;
КонецПроцедуры
```

Листинг 10.1.10

В том случае, если есть нужные записи, мы их записываем в массив, который создаем внутри цикла. После того как массив сформирован, мы обходим его, и удаляем каждый элемент данного массива, который представляет собой запись регистра сведений, с помощью метода *Удалить* объекта *Набор записей*.

Посмотрите, как работает Ваша обработка.

Итак, мы с Вами научились создавать, редактировать и удалять записи независимых регистров сведений. Теперь научимся работать с регистрами сведений, подчиненных регистратору. Рекомендую все манипуляции с подчиненными регистрами сведений проводить только посредством документов, то есть если необходимо поменять значение какого-нибудь ресурса в регистре, то необходимо сначала поменять значение соответствующего реквизита в документе и после чего перепровести данный документ.

В документе *Установка цен на топливо*, в процедуре *Обработка проведения*, уже создан код, в результате выполнения которого будет создаваться запись в регистре сведений *ЦеныНаТопливо*. Он создан, когда мы в 4-й главе при помощи конструктора создавали движения документа.

Документ УстановкаЦенНаТопливо: Модуль объекта

```
Процедура ОбработкаПроведения(Отказ, Режим)
    // регистр ЦеныНаТоплива
    Движения.ЦеныНаТоплива.Записывать = Истина;
    Для Каждого ТекСтрокаЦеныТоплива Из ЦеныТоплива Цикл
        Движение = Движения.ЦеныНаТоплива.Добавить();
        Движение.Период = Дата;
        Движение.ТипТоплива = ТекСтрокаЦеныТоплива.ТипТоплива;
        Движение.ПоставщикТоплива = Поставщик;
        Движение.Цена = ТекСтрокаЦеныТоплива.Цена;
    КонецЦикла;
КонецПроцедуры
```

Рис. 10.1.18

В этом коде мы обращаемся к набору записей регистра сведений *ЦеныНаТоплива* через свойство документа *Движения*.

Мы присваиваем значение *Истина* свойству *Записывать* объекта *Регистр сведений Набор записей*, для того чтобы при проведении документа создавались записи в регистре сведений.

Обращаю Ваше внимание, что свойство *Движения.ЦеныНаТоплива* имеет тип *Регистр сведений Набор записей*, поэтому для данного свойства можно применять все методы объекта *Набор записей*.

В цикле по табличной части документа мы создаем новую запись, используя метод *Добавить* объекта *Регистр сведений Набор записей*. И присваиваем значение измерениям и ресурсам.

Как Вы видите, в данном случае метод *Записать* объекта *Набор записей* не используется, это за нас сделает платформа 1С. Достаточно установить свойство *Записывать* в значение *Истина*.

Резюме

В этой части мы научились работать с регистрами сведений: создавать, редактировать и удалять записи. Вы поняли, что создавать записи можно как с помощью менеджера записи, так и с помощью набора записей.

Менеджер записи подходит, когда необходимо создать одну или несколько записей в регистр, без особого контроля.

В то же время набор записей более универсальный инструмент. Благодаря свойству *Отбор* данного объекта можно осуществлять любые выборки по любым признакам, что позволит Вам исключить дублирование вновь созданных записей регистров сведений, также Вы сможете редактировать и удалять имеющиеся записи.

Часть 2. Регистры сведений. Получение данных

Во второй части десятой главы мы научимся получать данные из регистров сведений. Сделать это можно двумя способами: используя методы менеджера регистров и используя язык запросов платформы 1С.

Начнем с методов менеджера регистров.

Выбрать

Первый метод, который мы с Вами разберем, это метод *Выбрать*. Данный метод формирует выборку записей регистров сведений.

Рассмотрим синтаксис этого метода.

Первый вариант синтаксиса, для периодического регистра сведений.

Выбрать(<НачалоИнтервала>, <КонецИнтервала>, <Отбор>, <Порядок>)

Данный метод по своему синтаксису похож на метод *Выбрать* для документов, который мы проходили в восьмой главе. Возвращает он объект *Выборка регистров сведений*.

Все параметры метода *Выбрать* являются необязательными, ниже мы разберем каждый из них.

Первый и второй параметр задают начало и конец интервала времени, за который будут выбираться записи регистра сведений.

Если не будет указан параметр *НачалоИнтервала*, то выборка будет начинаться с самой первой по времени записи. А если не указан параметр *КонецИнтервала*, то выборка будет оканчиваться самой последней по времени записью.

Третий параметр - это структура, в которой в качестве ключа идет название поля, по которому будет осуществляться отбор, а в качестве значения структуры – значение отбора по этому полю. Отбор может осуществляться только по измерениям и реквизитам, для которых установлен признак *Индексировать*, либо признак *Ведущие*.

Посмотрите, где это устанавливается.

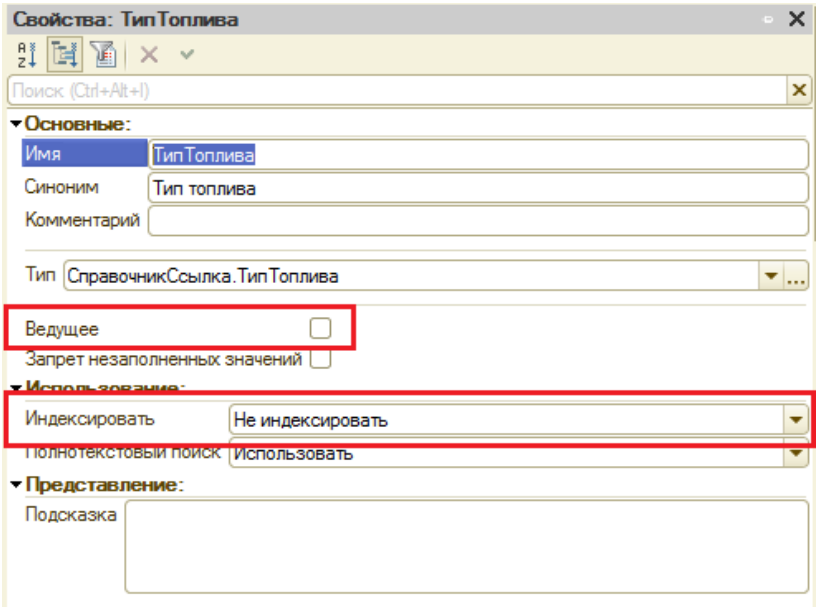


Рис. 10.2.1

Обращаю Ваше внимание на то, что структура может содержать только один элемент. Если необходимо осуществить выборку с отбором по двум или более полям, то необходимо использовать язык запросов.

И последний параметр - это *Порядок*. Данный параметр имеет тип *Строка* и может содержать название измерения или реквизита, по которому необходимо осуществить упорядочивание выборки. Обратите внимание, что у измерения или реквизита также должен быть установлен признак *Индексировать*.

Название полей можно применять с уже знакомыми Вам словами *Возр* и *Убыв*.

Это значит, что упорядочивание будет проходить либо по возрастанию, либо по убыванию.

Синтаксис метода *Выбрать* для непериодического регистра точно такой же, только без параметров начала и конца интервала.

Выбрать(<Отбор>, <Порядок>)

Параметры *Отбор* и *Порядок* имеют точно такой же функциональный смысл, как и в предыдущем варианте синтаксиса.

Теперь для закрепления решим небольшую задачку: создадим обработку, форму, на форме создадим реквизит *ПоставщикТоплива* с соответствующим типом и реквизит *ЦеныНаТопливо*, тип которого будет *ТаблицаЗначений* с колонками *Период, ТипТоплива* и *Цена* с соответствующими типами.

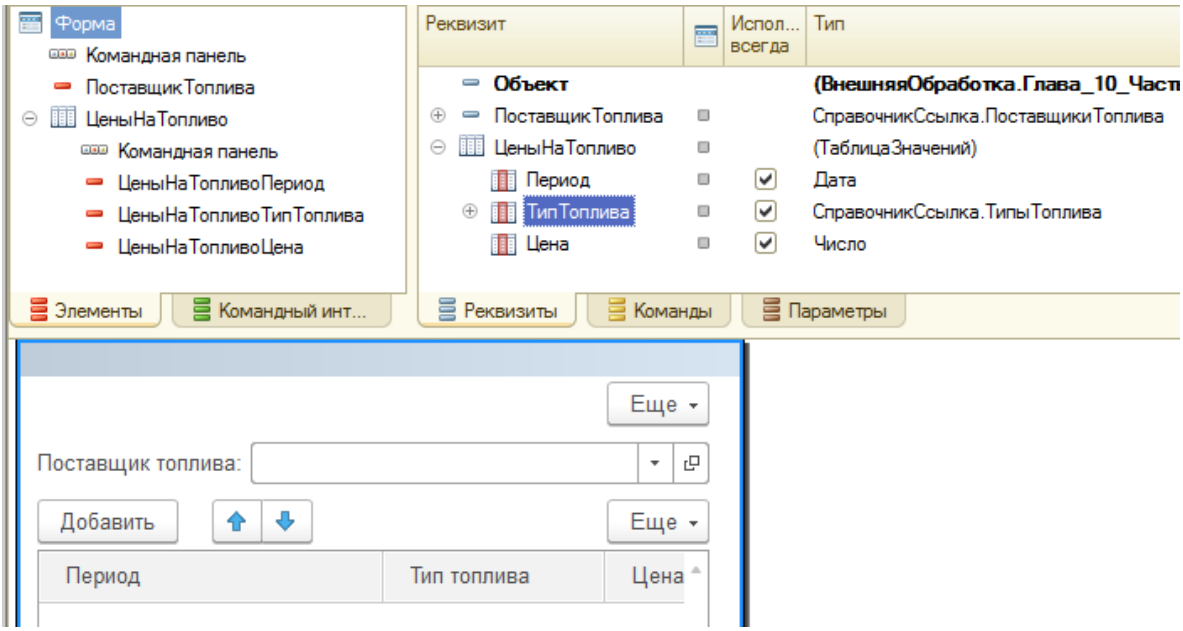


Рис. 10.2.2

При выборе поставщика будем заполнять таблицу данными из регистра сведений *Цены на топливо* для выбранного поставщика, которые соответствуют *текущему* году.

Для того, чтобы отбор работал, нам необходимо для измерения *ПоставщикТоплива* у регистра сведений *ЦеныНаТопливо* установить признак *Индексировать*.

В модуле формы обработки напишите серверную процедуру, которая осуществляет выборку из регистра и заполняет таблицу значений.

```
&НаСервере
Процедура ПолучитьВыборку(ДатаТекущая)
    ЦеныНаТопливо.Очистить();
    МенеджерЦенНаТопливо = РегистрыСведений.ЦеныНаТоплива;
    Отбор = Новый Структура;
    Отбор.Вставить("ПоставщикТоплива",ПоставщикТоплива);
    Выборка = МенеджерЦенНаТопливо.Выбрать(НачалоГода(ДатаТекущая),
                                           КонецГода(ДатаТекущая),
                                           Отбор);
    Пока Выборка.Следующий() цикл
        НоваяСтрока = ЦеныНаТопливо.Добавить();
        НоваяСтрока.Период      = Выборка.Период;
        НоваяСтрока.ТипТоплива  = Выборка.ТипТоплива;
        НоваяСтрока.Цена        = Выборка.Цена;
    КонецЦикла;
КонецПроцедуры // ПолучитьВыборку(ДатаТекущая)
```

Листинг 10.2.1

Прокомментируем данный код. В Вашу процедуру мы передаем один параметр - это текущая дата. Хочу Вам посоветовать не использовать системную функцию 1С *ТекущаяДата()* внутри собственных процедур или функций, а передавать ее в качестве параметра, это поможет в дальнейшем избежать множества проблем.

Первым шагом мы очищаем нашу таблицу значений. После создаем менеджер регистра сведений, новую структуру, которую называем *Отбор*, и создаем у этой структуры единственный элемент с ключом *ПоставщикТоплива*, и со значением, в которое запишем одноименный реквизит формы. *Обратите внимание: название ключа должно совпадать с названием изменения регистра сведений.*

Следующим шагом создаем выборку при помощи метода *Выбрать* менеджера регистра сведений. У данного метода три параметра: начало и конец интервала и структура, по которой будет осуществляться отбор. В конце мы идем по выборке и заполняем нашу таблицу значений.

После того, как мы создали процедуру в модуле формы, будем ее использовать в событии *ПриИзменении* элемента формы «ПолеВвода», который связан с реквизитом *ПоставщикТоплива*.

Зайдите с палитру свойств этого элемента и создайте обработчик события «ПриИзменении».

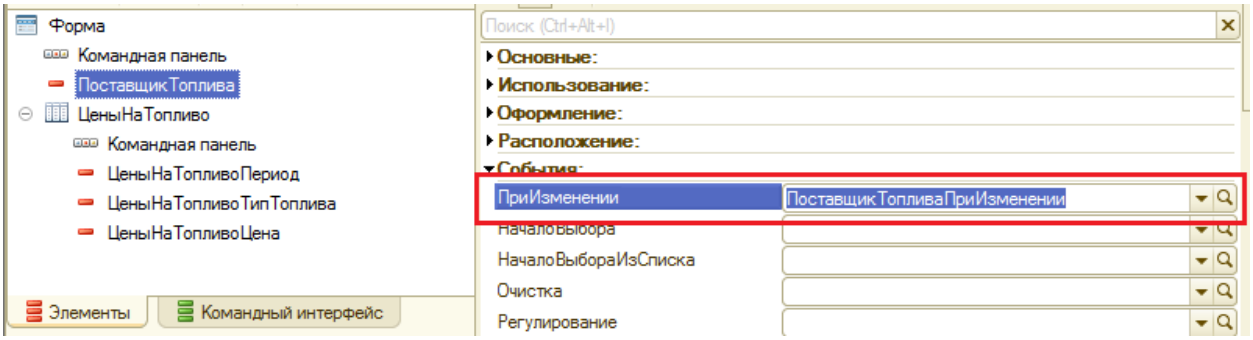


Рис. 10.2.3

В обработчике события и будем использовать процедуру *ПолучитьВыборку*.

```
&НаКлиенте
Процедура ПоставщикТопливаПриИзменении(Элемент)
    ПолучитьВыборку(ТекущаяДата());
КонецПроцедуры
```

Листинг 10.2.2

Теперь сохраните обработку и посмотрите, как она работает.

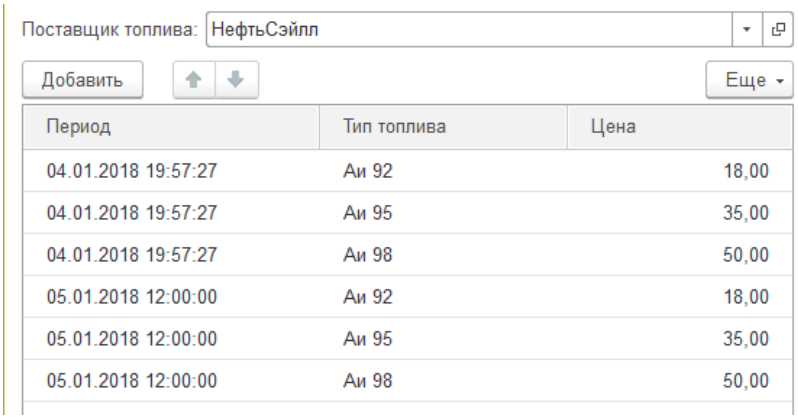


Рис. 10.2.4

Как видите, осуществляется выборка записей с нужными нам автомобилями за текущий год. Самостоятельно подправьте обработку, чтобы можно было указывать дату пользователю.

Рассмотрим еще два метода получения данных, которые будут новыми для Вас. Это *Срез первых* и *Срез последних* записей.

«Срез последних»

Метод менеджера регистра сведений *СрезПоследних* получает наиболее поздние записи регистра сведений на указанную дату. Как Вы уже поняли, данный метод применим только для **периодических** регистров сведений.

Метод имеет следующий синтаксис:

СрезПоследних(<КонецПериода>, <Отбор>)

Первый параметр – дата, на которую берется срез последних записей регистра.

Второй параметр - это отбор, данный параметр аналогичен отбору в выборках.

В отличие от выборки, данный метод возвращает *таблицу значений,* поля которой соответствуют названиям измерений и ресурсов регистра и имеют соответствующие типы.

Возможно, у Вас возникнет вопрос: «Чем данный метод отличается от простой выборки?» Продемонстрируем ответ на этот вопрос на примере: будем вводить дату и осуществлять выборку из регистра *ЦеныНаТопливо* с конечной датой, равной введенной, и срез последних на введенную дату.

Создайте обработку, форму, на которой разместите реквизит *ДатаСреза* с типом значения *Дата* (состав *Дата и время*) и две таблицы значений: *Выборка* и *СрезПоследних.* Колонки этих табличных полей будут соответствовать названиям измерений и ресурсов регистра сведений *ЦеныНаТопливо.* А также команду формы «Заполнить таблицы», которую разместите в командной панели формы.

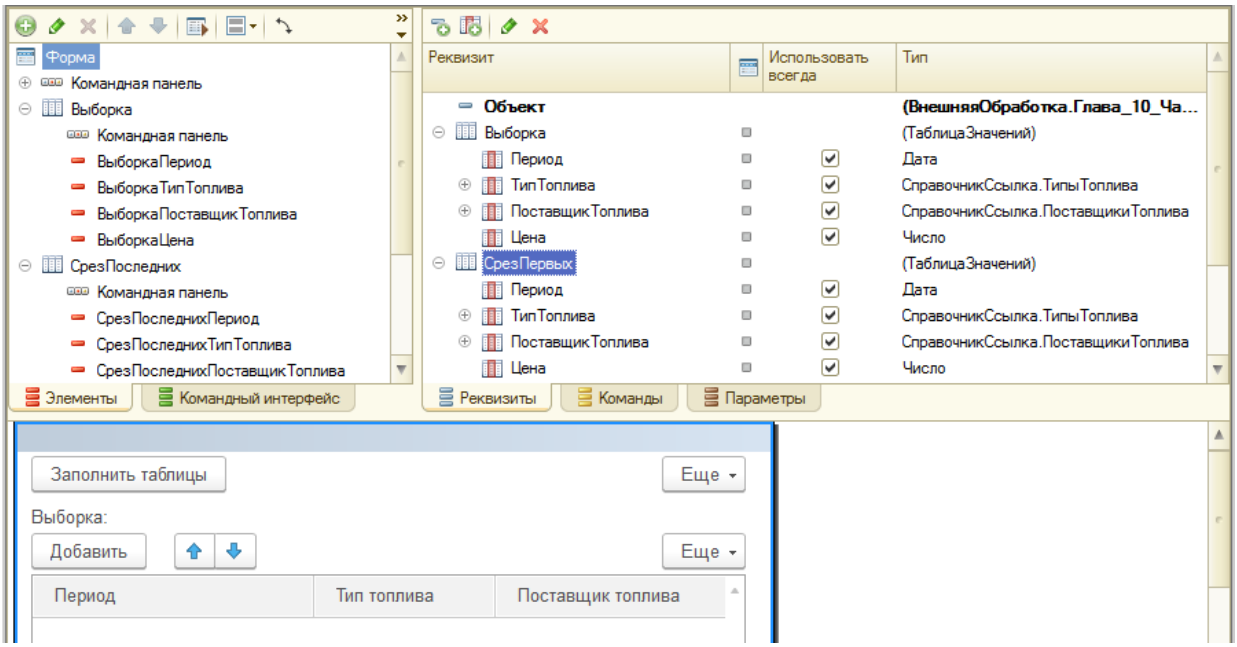


Рис. 10.2.5

Внутри серверной процедуры обработчика команды напишите следующий код:

```
&НаСервере
Процедура ЗаполнитьТаблицыНаСервере()
    Выборка.Очистить();
    СрезПоследних.Очистить();
    РегЦеныНаТопливо = РегистрыСведений.ЦеныНаТоплива;
    ВыборкаИзРегистра = РегЦеныНаТопливо.Выбрать(,ДатаСреза);
    Пока ВыборкаИзРегистра.Следующий() цикл
        НоваяСтрока = Выборка.Добавить();
        ЗаполнитьЗначенияСвойств(НоваяСтрока,ВыборкаИзРегистра);
    КонецЦикла;
    ТЗСрезПоследних = РегЦеныНаТопливо.СрезПоследних(ДатаСреза);
    СрезПоследних.Загрузить(ТЗСрезПоследних);
КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьТаблицы(Команда)
    ЗаполнитьТаблицыНаСервере();
КонецПроцедуры
```

Листинг 10.2.3

В данном коде мы первоначально получаем выборку из регистра, причем датой окончания выборки ставим дату среза, и затем заполняем таблицу значений *ТаблВыборка*. Следующим шагом мы получаем срез последних из регистра и результат сразу загружаем в реквизит формы *СрезПоследних*.

Сохраните обработку и посмотрите, что у Вас получится.

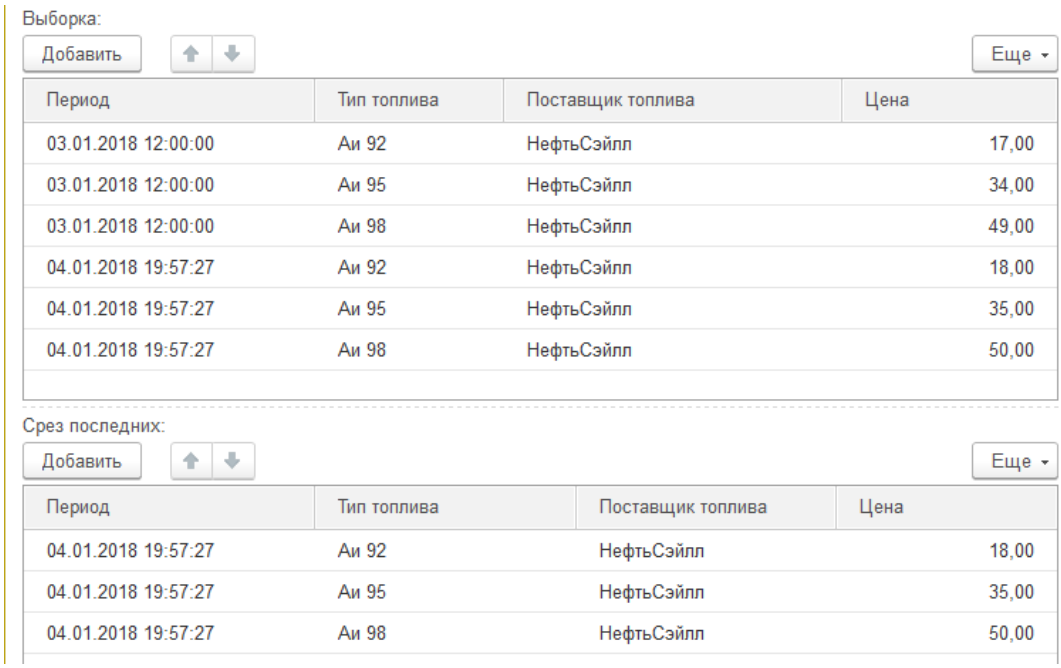


Рис. 10.2.6

Как видите, в случае выборки вывелись все записи до *Даты среза*. Но в случае *Среза Последних* мы имеем только одну запись, наиболее близкую к дате среза. Т.е. смысл метода *Срез Последних* в том, что из имеющегося множества записей с одинаковым набором измерений, разбросанных по датам, он выбирает ту запись, дата которой меньше даты среза и наиболее близко к ней расположена. Если в регистре будут записи, период которых более поздний, чем дата среза, то они не попадут в результат. Метод вернет все записи, с любыми комбинациями измерений, дата которых меньше даты среза и наиболее близка к ней.

Поработайте с данной обработкой и посмотрите, какие записи будет возвращать метод, в зависимости от того, какая дата указана в поле ввода.

«Срез первых»

В отличие от предыдущего метода, *СрезПервых* возвращает наиболее ранние записи регистра. Данный метод имеет синтаксис, аналогичный синтаксису предыдущего метода:

СрезПервых(<НачалоПериода>, <Отбор>)

Где первый параметр - это начало периода, начиная с которого будут выбираться записи регистра сведений. Второй параметр – «Отбор» - выполняет ту же функцию, что и в выборках.

Данный метод также возвращает таблицу значений.

Для наглядности доработаем нашу предыдущую обработку: создадим на форме третью таблицу *СрезПервых* и будем выводить срез первых по *Дате среза* в эту третью таблицу.

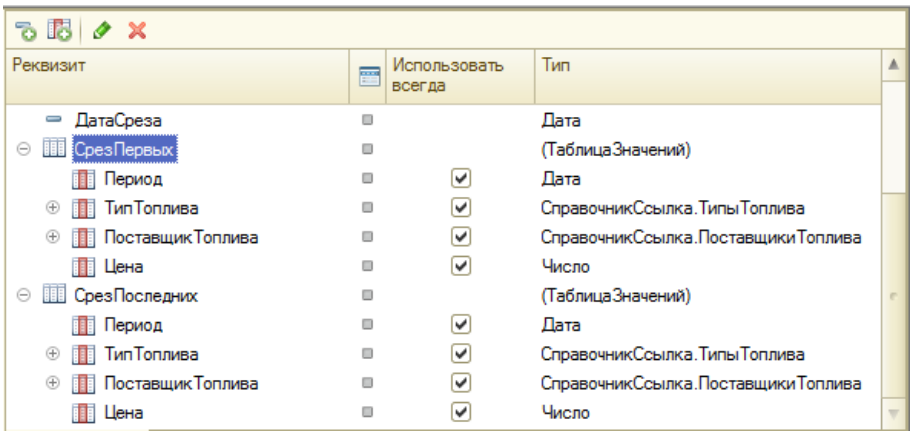


Рис. 10.2.7

Допишите код в серверном обработчике команды.

```
ТЗСрезПервых = РегЦеныНаТопливо.СрезПервых(ДатаСреза);
СрезПервых.Загрузить(ТЗСрезПервых);
```

Листинг 10.2.4

Теперь сохраните обработку и перезапустите ее.

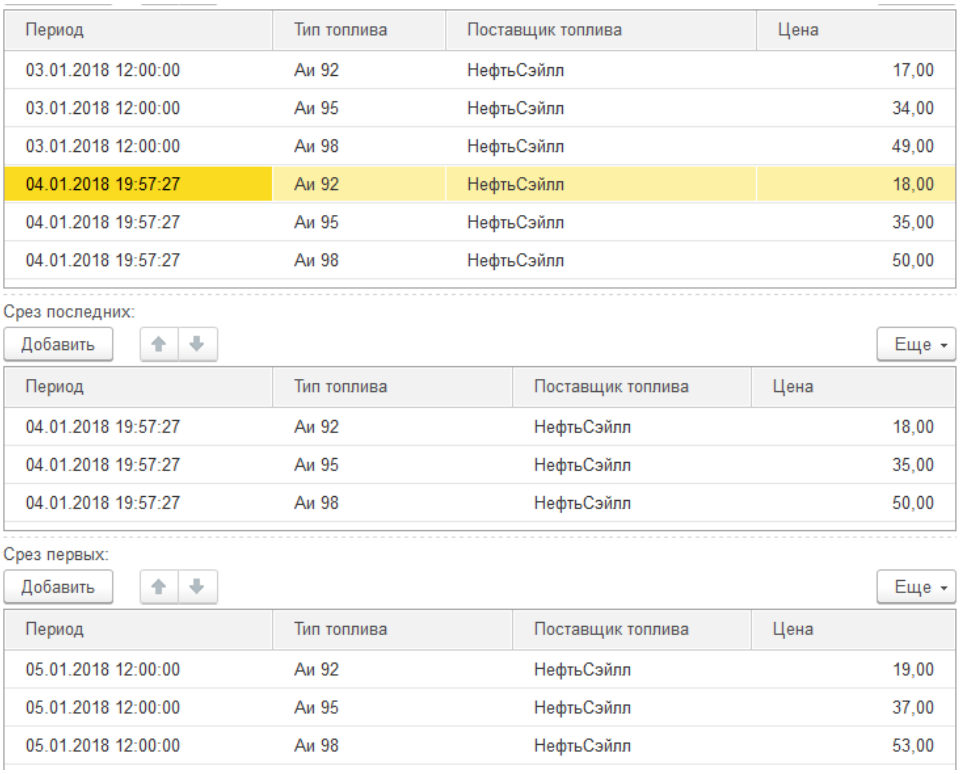


Рис. 10.2.8

Как Вы видите из выполнения обработки, метод возвращает записи, которые следуют после даты среза. Причем для определенной комбинации измерений возвращается только одна запись, период которой позже даты среза и наиболее близок к ней. Все последующие записи отбрасываются.

То есть данный метод - полная противоположность методу *СрезПоследних*.

Поэкспериментируйте с новым методом самостоятельно.

Выборка, «срез последних» и «срез первых» в запросе

Теперь разберем, как получать данные из регистров сведений в запросах. Для этого можно использовать виртуальные таблицы *СрезПоследних* и *СрезПервых*. Они применимы только к *периодическим* регистрам сведений.

Виртуальные таблицы языка запросов это новое для Вас понятие. Виртуальная таблица не существует в базе данных, она создается во время выполнения запроса, а потом уничтожается. Разработчик может самостоятельно создавать различные виртуальные таблицы по правилам, но их перечень всегда ограничен платформой 1С. Например, для *периодических* регистров сведений можно создать две виртуальные таблицы, это *СрезПоследних* и *СрезПервых*, которые возвращают наиболее поздние и наиболее ранние записи регистров сведений, по аналогии с одноименными методами менеджера регистра сведений.

Рассмотрим, как создаются запросы с данными таблицами.

Самостоятельно создайте обработку с формой, на которой создайте реквизит *ДанныеЗапроса* с типом *ТаблицаЗначений*, колонки этой таблицы значений совпадают с измерениями и ресурсами регистра сведений *НормаРасхода*. А также реквизит «ДатаСреза» с типом «Дата» (состав *Дата и время*) В командной панели формы создайте подменю «Выполнить запрос», в которое будем помещать различные команды формы.

Создайте первую команду формы «Простая выборка», для этой команды создайте серверные и клиентские обработчики. В серверном обработчике создадим запрос, который возвращает простую выборку регистра сведений. Создайте пустой запрос, зайдите в конструктор запросов, найдите регистр сведений *НормаРасхода* и выберите его. Выберите поля: *Период, Автомобиль, Сезон* и *Норма*.

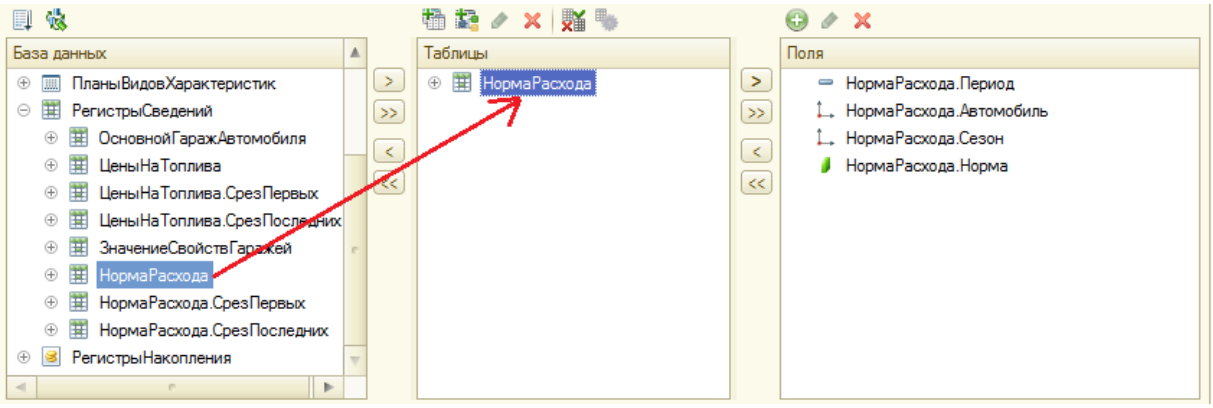


Рис. 10.2.9

Теперь задайте условия, чтобы выбирались записи с периодом меньше или равным дате среза.

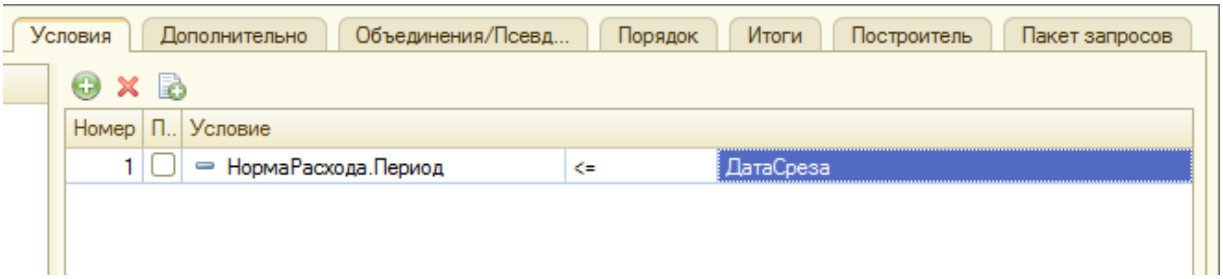


Рис. 10.2.10

Все, Ваш запрос создан, и он имеет следующий вид:

```
&НаСервере
Процедура ПростаяВыборкаНаСервере()
    Запрос = Новый Запрос;
    Запрос.Текст = "ВЫБРАТЬ
                   |    НормаРасхода.Период КАК Период,
                   |    НормаРасхода.Автомобиль КАК Автомобиль,
                   |    НормаРасхода.Сезон КАК Сезон,
                   |    НормаРасхода.Норма КАК Норма
                   |ИЗ
                   |    РегистрСведений.НормаРасхода КАК НормаРасхода
                   |ГДЕ
                   |    НормаРасхода.Период <= &ДатаСреза";

КонецПроцедуры
```

Листинг 10.2.5

Доделайте заполнение таблицы самостоятельно. У Вас должен получиться следующий результат:

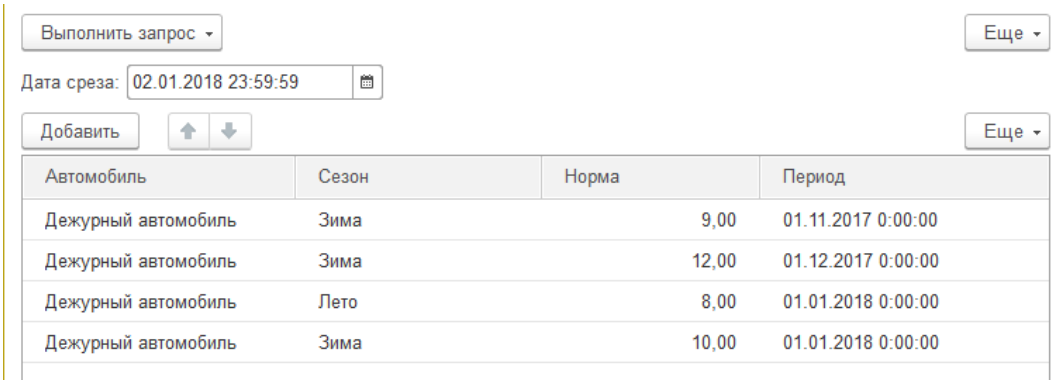


Рис. 10.2.11

Данный запрос возвращает те же данные, что и выборка. Вы спросите: «В чем удобство запроса?» А удобство в том, что Вы практически не ограничены по отбору и сортировке, в отличие от метода менеджера регистра, и можете осуществлять отбор и сортировку даже по неиндексируемым полям и по ресурсам.

Например, можно сделать так:

```
ВЫБРАТЬ
    НормаРасхода.Период КАК Период,
    НормаРасхода.Автомобиль КАК Автомобиль,
    НормаРасхода.Сезон КАК Сезон,
    НормаРасхода.Норма КАК Норма
ИЗ
    РегистрСведений.НормаРасхода КАК НормаРасхода
ГДЕ
    НормаРасхода.Период МЕЖДУ &ДатаНачала И &ДатаОкончания
    И НормаРасхода.Сезон = &Сезон
    И НормаРасхода.Норма >= &Норма
```

Листинг 10.2.6

То есть мы ввели отбор по ресурсу и измерению, и это будет нормально работать. Поэтому мой совет: если Вам надо получить быстро простую выборку из регистра сведений без лишних отборов и сортировок, используйте метод менеджера запроса. А если появляются дополнительные требования к Вашей выборке, то работайте с языком запросов.

Теперь разберем работу с виртуальными таблицами *СрезПоследних* и *СрезПервых*.

Зайдите снова в конструктор запроса и раскройте регистры сведений. Как Вы видите, помимо самих регистров есть таблицы, название которых состоит из названия регистра, соединенного посредством точки со словосочетанием *«срез последних»* или *«срез первых»* (рис. 10.2.12).

Это виртуальные таблицы, как уже было сказано, они нигде не хранятся, а вычисляются при выполнении запроса.

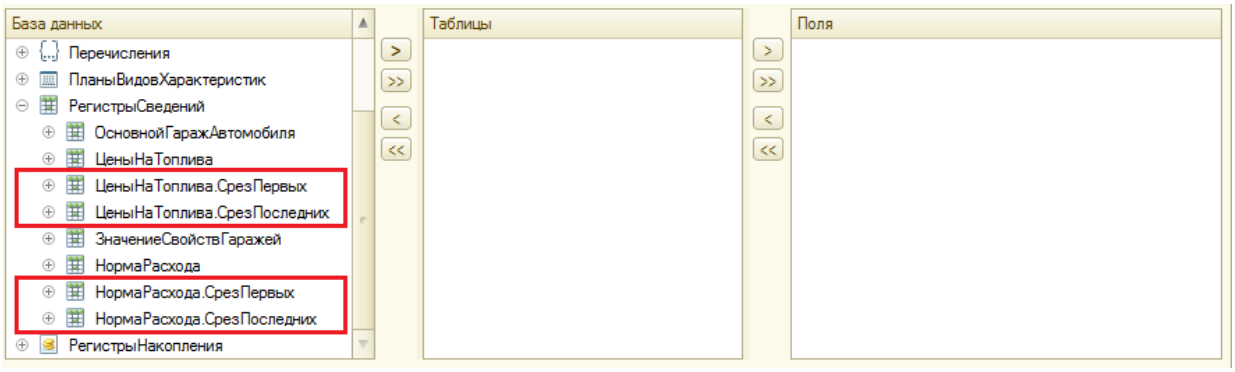


Рис. 10.2.12

Создайте новую команду «Срез последних», создайте обработчики команды в клиентском и серверном контексте. В серверном обработчике создайте пустой запрос и откройте конструктор запроса. В конструкторе выберете *«срез последних»* для регистра сведений *НормыРасхода*.

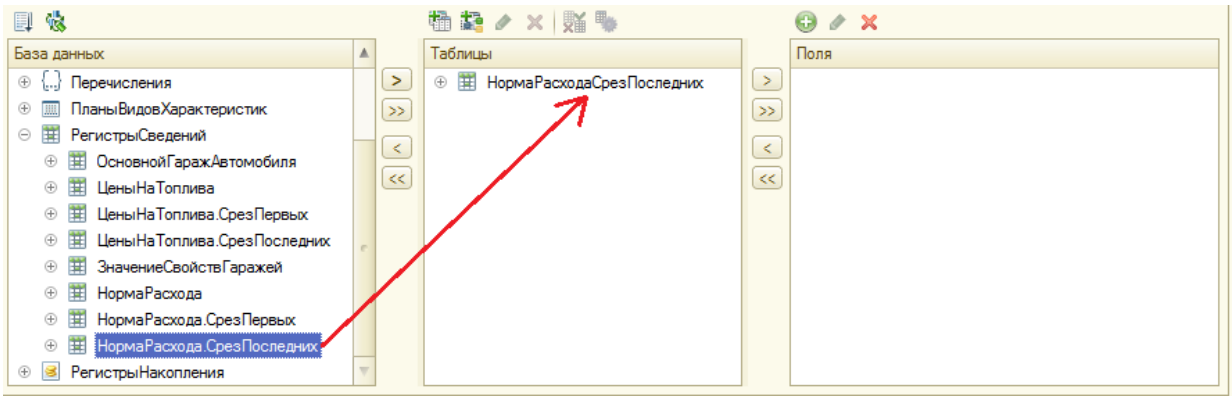


Рис. 10.2.13

При выделении виртуальной таблицы становится активной кнопка *«Параметры виртуальной таблицы»*.

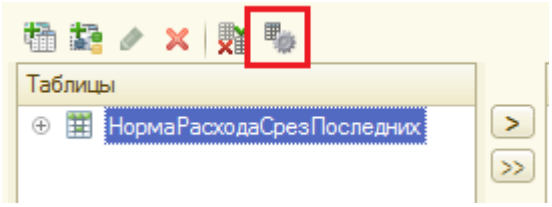


Рис. 10.2.14

Нажимаем на данную кнопку и открываются параметры виртуальной таблицы *«Срез последних»*.

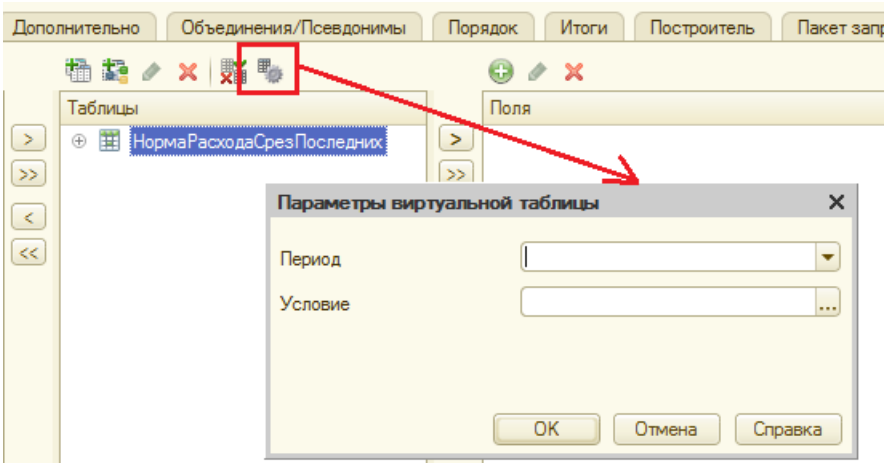


Рис. 10.2.15

Открылась форма параметром виртуальной таблицы, в которой можно установить период среза и условия. В поле *Период* необходимо указывать параметр, в который будет передана какая-нибудь дата.

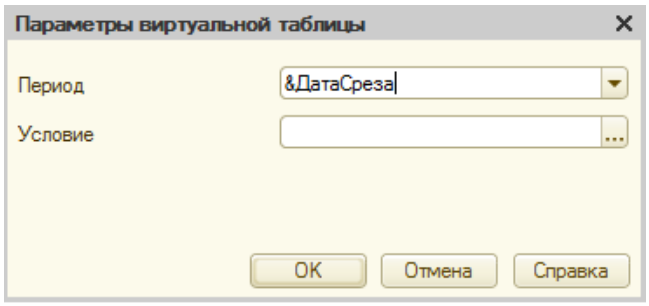


Рис. 10.2.16

В *Условие* можно задать ограничение на выборку. Например, мы можем сделать срез последних по всем нормам расхода, у которых сезон – «Лето». Причем в качестве полей для условия можно выбирать и измерения, и ресурсы, и реквизиты. Также можно выполнять любую логическую связку: *И, ИЛИ, НЕ*.

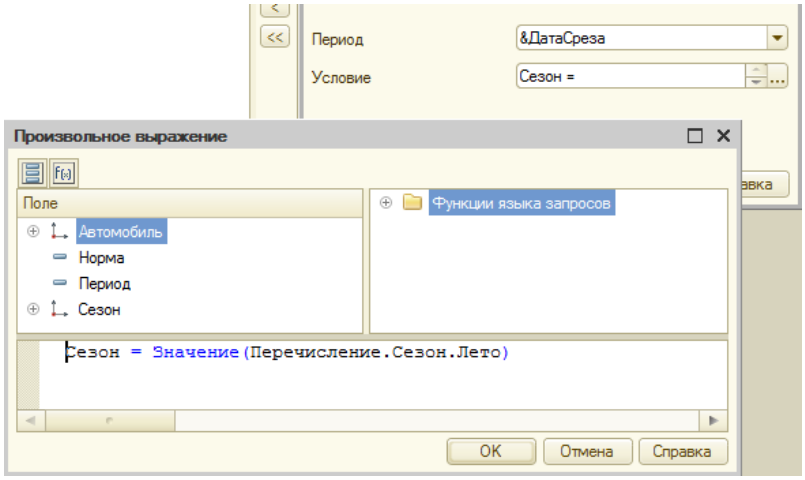


Рис. 10.2.17

Обратите на это внимание, что ограничение на выборку «среза последних» нужно делать именно посредством условия виртуальной таблицы. Ограничивать виртуальную таблицу через условия запроса не оптимально с точки зрения производительности!

После того, как мы задали условия виртуальной таблицы, выберем все поля виртуальной таблицы и сохраним запрос.

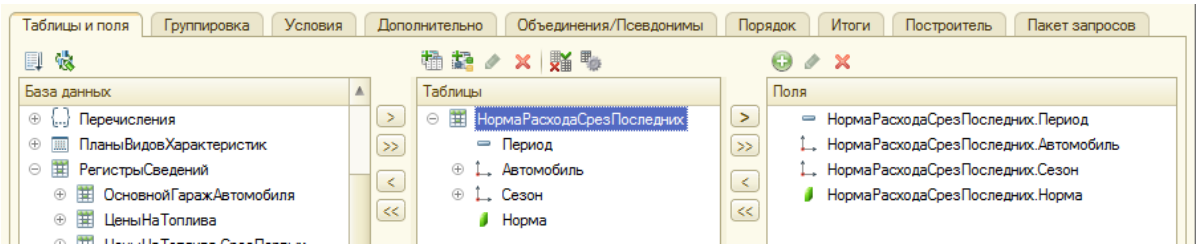


Рис. 10.2.18

В результате у Вас получился следующий запрос:

```
ВЫБРАТЬ
    НормаРасходаСрезПоследних.Период КАК Период,
    НормаРасходаСрезПоследних.Автомобиль КАК Автомобиль,
    НормаРасходаСрезПоследних.Сезон КАК Сезон,
    НормаРасходаСрезПоследних.Норма КАК Норма
ИЗ
    РегистрСведений.НормаРасхода.СрезПоследних(&ДатаСреза,
                                                Сезон =
ЗНАЧЕНИЕ(Перечисление.Сезон.Лето)
                                               ) КАК
НормаРасходаСрезПоследних
```

Листинг 10.2.7

Создайте самостоятельно обработку этого запроса и выведите данные в таблицу формы. Должен получиться подобный результат:

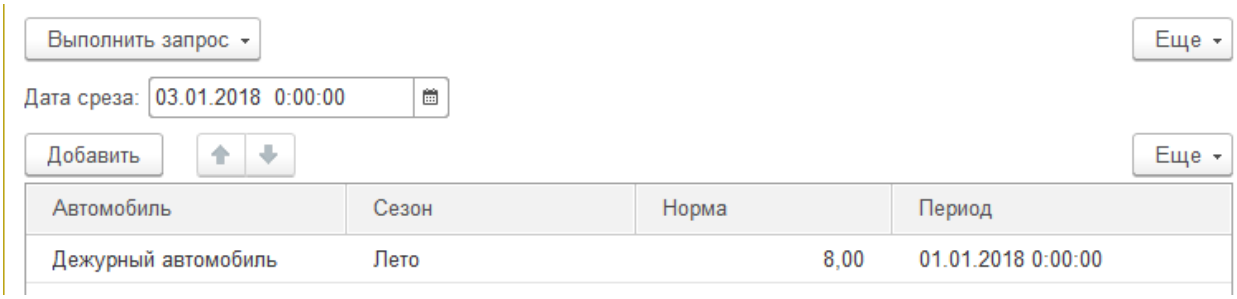


Рис. 10.2.19

Виртуальные таблицы *СрезПоследних* и *СрезПервых* возвращают тот же результат, что и одноименные методы менеджера регистра сведений. Только они более удобны в плане отборов, сортировки и т.п.

Теперь небольшое задание. Создайте новую команду «Срез первых», в которой при помощи виртуальной таблицы «СрезПервых» получайте «срез первых» регистра сведений «НормаРасхода» для сезона *Зима*.

Резюме

На этом мы закончим изучать способы получения данных из регистров сведений. Повторюсь: если необходимо получить простую выборку или простой срез, то используйте методы менеджера регистра сведений, в более сложных случаях используйте виртуальные таблицы «СрезПервых» и «СрезПоследних» языка запросов платформы 1С. Если имеется необходимость поставить отбор на срез первых или срез последних, то в целях оптимизации производительности необходимо устанавливать в условиях виртуальной таблицы.

Часть 3. Регистры накопления. Работа с записями

Освежим и углубим информацию о регистрах накопления, которую Вы получили в четвертой главе.

Как Вы помните (если не помните, то освежите свою память, пролистав четвертую главу), регистр накопления предназначен для хранения данных о движении материальных величин. Это может быть прибытие и выбытие материала, автомобилей и т.п. или просто движение денежных средств в кассе предприятия.

Регистр накопления состоит из измерений, ресурсов и реквизитов. По сути, он представляет N-мерную систему координат, в узлах которой хранятся данные. Оси такой системы координат являются измерениями, а данные, которые хранятся в узлах, – ресурсы. Помимо ресурсов, в узлах могут храниться реквизиты, которые несут на себе роль вспомогательных данных. Основной задачей регистров накопления является суммирование ресурсов. Разработчик может получить нужную сумму по любому измерению регистра накопления. Как это осуществить, мы узнаем в следующей части текущей главы.

Существует два вида регистров накопления в платформе 1С: это *Регистр остатков* и *Оборотный регистр*. Для регистров остатков мы можем получить остатки по данному регистру на заданный момент времени. У оборотных регистров мы можем только посмотреть, какой был оборот материальных величин за указанный промежуток времени. Мы уже создали в нашей прикладной задаче два регистра накопления. Это регистры *ПробегАвтомобиля* и *ТопливоВАвтомобилях*.

Регистр *ПробегАвтомобиля* является *оборотным* регистром.

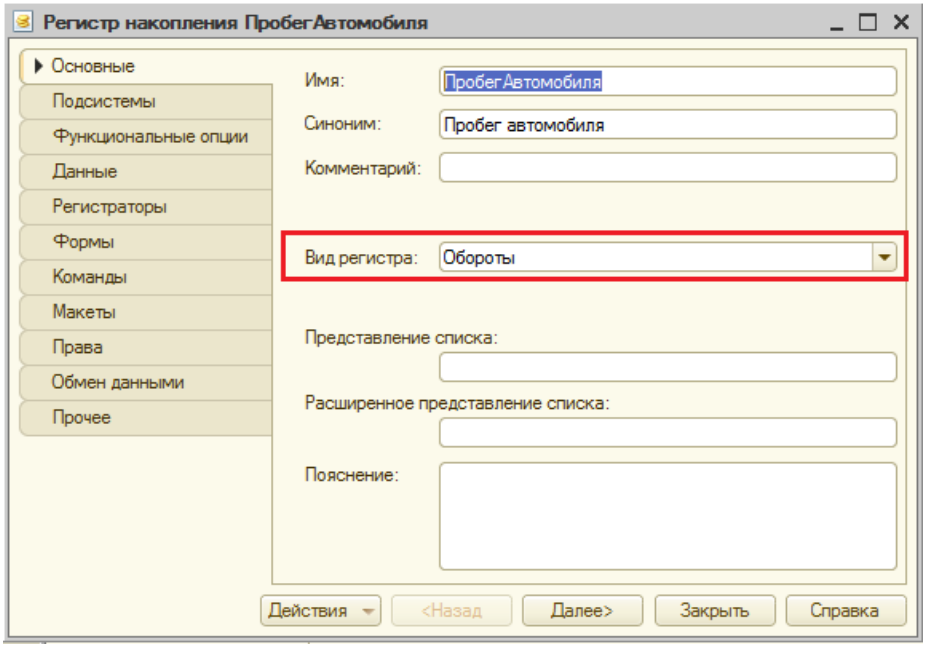


Рис. 10.3.1

Имеет одно измерение *Автомобиль* (тип *СправочникСсылка.Автомобили*) и один ресурс *Пробег* (все ресурсы имеют тип «Число», можно только изменять параметры числа).

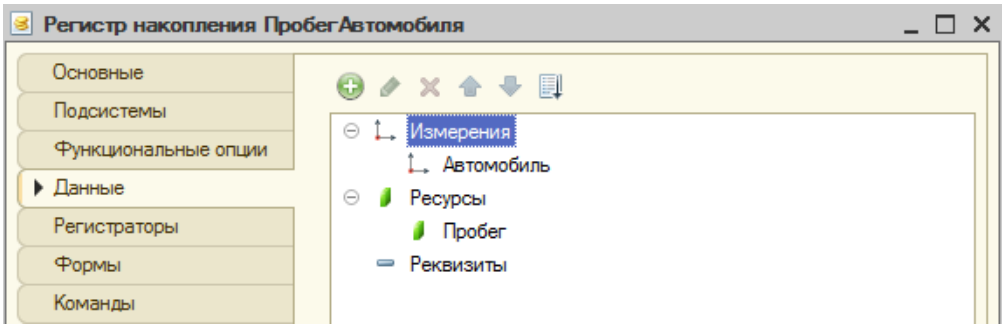


Рис. 10.3.2

Регистратором регистра *ПробегАвтомобиля* является документ *Прибытие в гараж*.

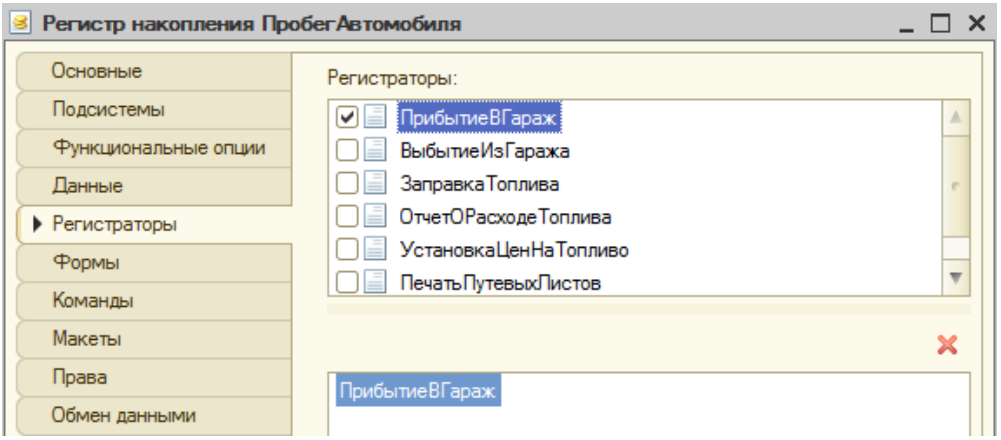


Рис. 10.3.3

Регистр *ТопливоВАвтомобилях* является регистром остатков.

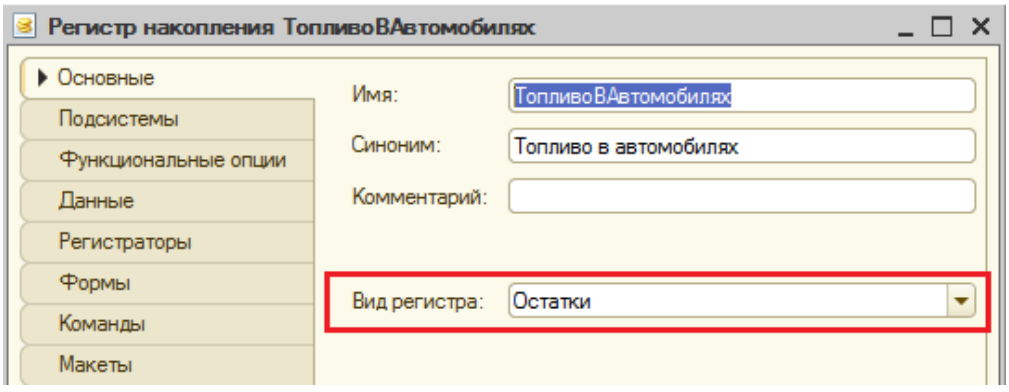


Рис. 10.3.4

Имеет два измерения - *Автомобиль* (тип *СправочникСсылка.Автомобили*), *ТипТоплива* (тип *СправочникСсылка.ТипыТоплива*) - и один ресурс *Количество*.

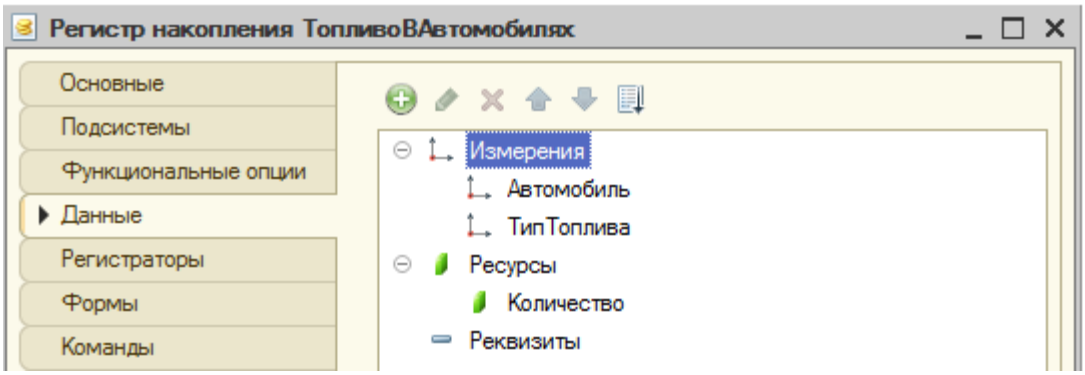


Рис. 10.3.5

Регистратором этого регистра являются документы *Заправка топлива* и *Отчет о расходе топлива*.

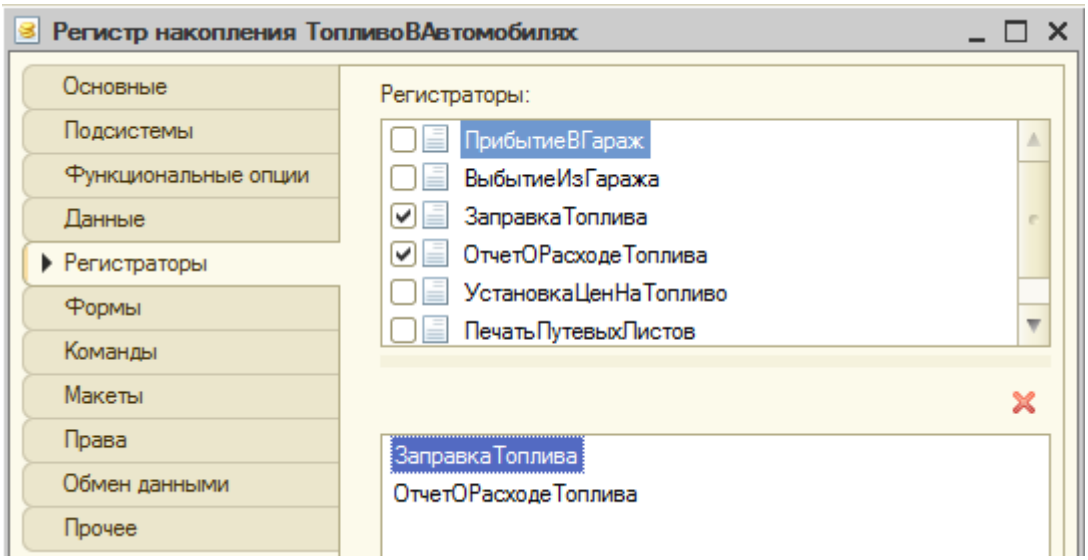


Рис. 10.3.6

Создавать регистры накопления мы с Вами научились в четвертой главе, теперь осталось научиться работать с ними, используя объекты языка программирования 1С.

И первый объект, который мы изучим, это *Менеджер регистра накопления*.

Менеджер регистра накопления

Менеджер регистра накопления позволяет управлять данными конкретного регистра накопления: получать остатки и обороты, получать выборку, создавать набор данных и прочее.

Создать менеджер несложно:

```
МенеджерПробегАвтомобиля = РегистрыНакопления.ПробегАвтомобиля;
```

Теперь переменная *МенеджерПробегАвтомобиля* будет иметь тип *Менеджер регистра накопления Пробег Автомобиля*.

Набор записей

Набор записей регистров накопления выполняет те же функции, что и *Набор записей регистров сведений*, а именно: считывание записей из базы данных по определенному регистратору, а также позволяет добавлять, изменять и удалять записи в наборе.

Поскольку методы *Набора записей регистров накоплений* совпадают с методами *Набора записей регистров сведений* по названию, по смыслу и по работе, то мы не будем подробно на них останавливаться. Единственно, что если в случае с регистрами сведений отбор можно было устанавливать как по измерениям (для независимых регистров), так и по регистратору (для зависимых), то в случае с регистрами накоплений (которые не могут быть независимыми) отбор можно установить только по регистратору.

Также обращаю Ваше внимание, что *Набор записей регистров накоплений* работает только с отбором по регистратору. Т.е. для того, чтобы создать набор записей нужно всегда устанавливать отбор по регистратору.

Сейчас мы создадим обработку, которая перебирает все проведенные документы *ПрибытиеВГараж* и выполняет запись пробега автомобиля для регистра накопления *ПробегАвтомобиля*. При этом будем проверять, есть ли уже запись по данному регистратору или нет.

Обычно, чтобы заполнить регистр накопления, нужно просто перепровести документ. На возможна ситуация, когда нужно сделать запись в регистре не проводя документ. Такая ситуация редко, но может встретиться. Например, когда регистр накопления был создан после разработки какого-то документа, причем экземпляры этого документа пользователь уже успел создать. В этом случае перепроводить все документы не всегда имеет смысл (может быть закрыт период, есть вероятность, что другие регистры «поплывут» и т.д.). Поэтому есть смысл сделать запись в регистре по какому-то документу.

Запись напрямую в регистр накопления на практике следует использовать очень аккуратно и осторожно! Поскольку, как правило, редко бывает простая запись, а при этом используются различные алгоритмы (проверок, заполнений и т.п.).

Создайте обработку, форму обработки, разместите на форме реквизит *Период* с типом *СтандартныйПериод*, а также команду «Заполнить регистр» для этой команды создайте обработчики на сервере и на клиенте. В серверном обработчике создайте объект *запрос* и получите выборку всех проведенных документов *ПрибытиеВГараж* за указанный период (выберете поля *Ссылка* и *Пробег*).

```
&НаКлиенте
Процедура ЗаполнитьРегистр(Команда)
    ЗаполнитьРегистрНаСервере();
КонецПроцедуры
&НаСервере
Процедура ЗаполнитьРегистрНаСервере()
    Запрос = Новый Запрос;
    Запрос.Текст = "ВЫБРАТЬ
                   |    ПрибытиеВГараж.Ссылка КАК Регистратор,
                   |    ПрибытиеВГараж.Пробег КАК Пробег,
                   |    ПрибытиеВГараж.Дата КАК Период,
                   |    ПрибытиеВГараж.Автомобиль КАК Автомобиль
                   |ИЗ
                   |    Документ.ПрибытиеВГараж КАК ПрибытиеВГараж
                   |ГДЕ
                   |    ПрибытиеВГараж.Проведен
                   |    И ПрибытиеВГараж.Дата МЕЖДУ &ДатаНачала И
&ДатаОкончания";
    Запрос.УстановитьПараметр("ДатаНачала",Период.ДатаНачала);
    Запрос.УстановитьПараметр("ДатаОкончания",Период.ДатаОкончания);
    Результат = Запрос.Выполнить();
    Если Результат.Пустой() Тогда
        Сообщить("Нет данных по указанному периоду");
        Возврат;
    КонецЕсли;
КонецПроцедуры
```

Листинг 10.3.1

Следующим шагом перед циклом выборки создайте менеджер регистра накопления *ПробегАвтомобиля* и набор записей с помощью метода менеджера *СоздатьНаборЗаписей.*

```
МенеджерПробегАвто = РегистрыНакопления.ПробегАвтомобиля;
НЗПробегАвто = МенеджерПробегАвто.СоздатьНаборЗаписей();
```

Листинг 10.3.2

Установите внутри цикла выборки отбор по регистратору и прочитайте набор данных из базы.

```
Пока Выборка.Следующий() Цикл
    Регистратор = Выборка.Регистратор;
    НЗПробегАвто.Отбор.Регистратор.Установить(Регистратор);
КонецЦикла;
```

Листинг 10.3.3

Теперь проверим, есть ли в полученном наборе данных информация, и если нет, то создадим новую запись и заполним поля этой записи (это все делаем внутри цикла по выборке).

```
Если НЗПробегАвто.Количество() = 0 Тогда
    НоваяЗапись = НЗПробегАвто.Добавить();
    ЗаполнитьЗначенияСвойств(НоваяЗапись,Выборка);
КонецЕсли;
```

Листинг 10.3.4

Обратите внимание, я специально полям запроса присвоил такие же псевдонимы, как и у названий измерений и ресурсов, чтобы можно было использовать процедуру *ЗаполнитьЗначенияСвойств*.

Поскольку мы имеем дело с регистром накоплений, то сохраняем набор записей при каждом обходе цикла. Так необходимо делать и с регистрами сведений, если они имеют регистратор. Сохраняем регистр внутри условия, где проверяем отсутствие записей.

```
НЗПробегАвто.Записать();
Сообщить("Записали движения для документа
        |" + Выборка.Регистратор);
```

Листинг 10.3.5

Теперь сохраните обработку, запустите и проверьте, появились ли движения у документов *ПрибытиеВГараж* по регистру *ПробегАвтомобиля*.

Если все сделано правильно, записи все должны создаться как надо, но некоторые с пустым значением ресурса «Пробег». Это означает, что не указан пробег в документе. Самостоятельно подправьте обработку так, чтобы такие записи не создавались (измените запрос), а мы сделаем команду, которая подчистит записи с пустым пробегом.

Создайте команду «Очистить пустые записи регистра», разместите её на форме и создайте обработчики команды в серверном и клиентском контексте. Внутри серверной процедуры создайте менеджер регистра накопления *ПробегАвтомобидля* и набор записей данного регистра.

```
&НаСервере
Процедура ОчиститьПустыеЗаписиРегистраНаСервере()
    МенеджерПробегАвто = РегистрыНакопления.ПробегАвтомобиля;
    НЗПробегАвто = МенеджерПробегАвто.СоздатьНаборЗаписей();
КонецПроцедуры


&НаКлиенте
Процедура ОчиститьПустыеЗаписиРегистра(Команда)
    ОчиститьПустыеЗаписиРегистраНаСервере();
КонецПроцедуры
```

Листинг 10.3.6

Поскольку набор записей регистров накоплений работает только с отбором по регистратору, нам необходимо выбрать все документы *Прибытие в гараж* за указанный период. Я сделаю это с помощью метода *Выбрать* менеджера документов *Прибытие в гараж*, а Вы сделайте при помощи запроса.

```
ВыборкаДокументов = Документы.ПрибытиеВГараж.Выбрать(Период.ДатаНачала,
                                                      Период.ДатаОкончания);
Пока ВыборкаДокументов.Следующий() Цикл

КонецЦикла;
```

Листинг 10.3.7

После этого установите внутри цикла для набора записей отбор по регистратору и прочитайте набор.

```
Пока ВыборкаДокументов.Следующий() Цикл
    НЗПробегАвто.Отбор.Регистратор.Установить(ВыборкаДокументов.Ссылка);
    НЗПробегАвто.Прочитать();
КонецЦикла;
```

Листинг 10.3.8

Теперь, в том случае, если есть данные для выбранного регистратора, то получим соответствующую запись.

```
НЗПробегАвто.Прочитать();
Если НЗПробегАвто.Количество() <> 0 Тогда
    ЗаписьНабора = НЗПробегАвто.Получить(0);
КонецЕсли;
```

Листинг 10.3.9

Мы получили запись с помощь метода *Получить*, этот метод возвращает объект *Запись регистра сведений*. Вводимый параметр это индекс записи. Поскольку у нас предполагается, что у регистратора только одна запись, то мы просто ввели первый индекс. Если же записей было бы несколько, то необходимо было бы организовать обход по набору.

После того как Вы получили запись, проверим, равен ли нулю *Расход*, и если да, то очистим набор и сохраним набор записей.

```
Если ЗаписьНабора.Пробег = 0 Тогда
    НЗПробегАвто.Очистить();
    НЗПробегАвто.Записать();
КонецЕсли;
```

Листинг 10.3.10

Сохраните обработку и посмотрите, как она работает.

Теперь оптимизируйте обработку: выведите с помощью запроса только те документы, у автомобилей которых пустая норма расхода.

Проведение документа

Мы научились создавать записи регистра накоплений независимо, т.е. вне документа, а теперь научимся создавать их при проведении документа.

Для этого зайдите в модуль объекта документа *Прибытие в гараж* и раскройте процедуру *Обработка проведения*.

В данной процедуре уже есть код, который мы создали при помощи конструктора движений в четвертой главе.

```
Процедура ОбработкаПроведения(Отказ, Режим)
    // регистр ПробегАвтомобиля
    Движения.ПробегАвтомобиля.Записывать = Истина;
    Движение = Движения.ПробегАвтомобиля.Добавить();
    Движение.Период = Дата;
    Движение.Автомобиль = Автомобиль;
    Движение.Пробег = Пробег;
КонецПроцедуры
```

Листинг 10.3.11

В этом коде мы получаем набор записей регистра накопления *ПробегАвтомобиля* при помощи свойства документа *Движения*. В этом свойстве хранятся наборы записей по всем регистрам, по которым документ осуществляет движения. После этого мы свойству *Записывать* набора записей присваиваем значение *Истина*, это значит, что после выполнения кода в процедуре *ОбработкаПроведения* наш набор записей запишется и нет смысла записывать его отдельно при помощи метода *Записать*. Также нет смысла отдельно очищать набор записей, поскольку у документа мы установили в свойство «Удаление движений» значение «Удалять автоматически при отмене проведения».

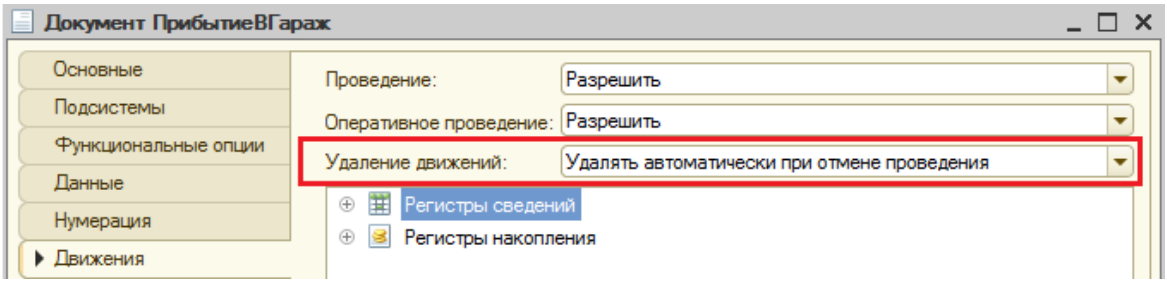


Рис. 10.3.7

Поэтому при отмене проведения все данные очистятся, а при перепроведении – перезапишутся. Хотя некоторые программисты перестраховываются и принудительно очищают набор записей в любом случае.

```
Движения.ПробегАвтомобиля.Очистить();
```

Листинг 10.3.12

Очистите код внутри процедуры проведения, и сейчас мы самостоятельно напишем проведения по регистру накопления *ПробегАвтомобиля*, чтобы у Вас закрепилось понимание. Но перед этим сделаем запрос, в котором получим данные по текущему документу, причем поставим условия, чтобы реквизит *Пробег* был заполнен.

```
Процедура ОбработкаПроведения(Отказ, Режим)
    Запрос = Новый Запрос;
    Запрос.Текст = "ВЫБРАТЬ
                   |    ПрибытиеВГараж.Дата КАК Период,
                   |    ПрибытиеВГараж.Автомобиль КАК Автомобиль,
                   |    ПрибытиеВГараж.Пробег КАК Пробег
                   |ИЗ
```

```
                    |    Документ.ПрибытиеВГараж КАК ПрибытиеВГараж
                    |ГДЕ
                    |    ПрибытиеВГараж.Пробег > 0
                    |    И ПрибытиеВГараж.Ссылка = &Ссылка";
    Запрос.УстановитьПараметр("Ссылка",Ссылка);
    Результат = Запрос.Выполнить();
    Если Результат.Пустой() Тогда
        Сообщить("Не заполнен пробег! Проведение не возможно!")
        Отказ = Истина;
    КонецЕсли;
    Выборка = Результат.Выбрать();
    Выборка.Следующий()
    //....
КонецПроцедуры
```

Листинг 10.3.13

В этом запросе мы обращаемся напрямую к проводимому документу, и если у этого документа реквизит *Пробег*, то присваиваем параметру *Отказ* значение *Истина*. Это значит, что документ не будет проведен, а пользователю выйдет соответствующее сообщение.

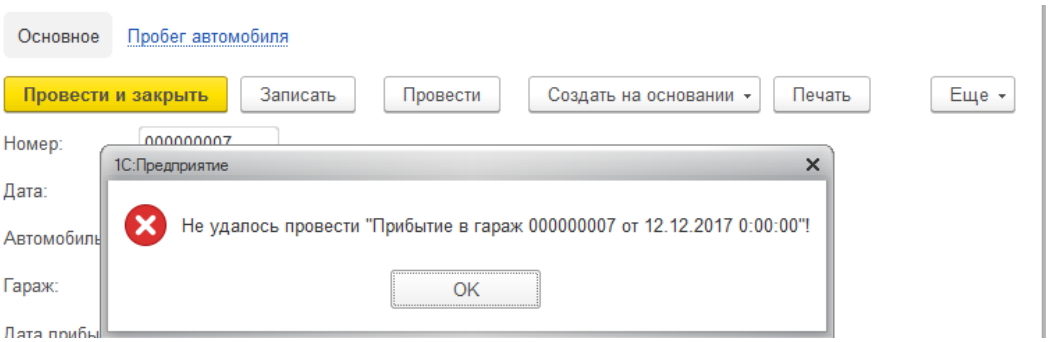


Рис. 10.3.8

Мы получим только один элемент выборки из запроса, поскольку запрос вернет одну запись и нет смысла обходить эту выборку целиком.

Получим набор записей регистра накопления *ПробегАвтомобиля*, очистим его и установим в свойство *Записывать* значение *Истина*. В этом случае после выполнения процедуры *ОбработкаПроведения* нужные записи в регистре будут созданы автоматически и нет смысла их отдельно записывать.

```
// регистр ПробегАвтомобиля
НЗПробегАвтомобиля = Движения.ПробегАвтомобиля;
НЗПробегАвтомобиля.Очистить();
НЗПробегАвтомобиля.Записывать = Истина;
```

Листинг 10.3.14

Осталось создать новую запись регистра и заполнить её поля данными из запроса.

```
НовЗапись = НЗПробегАвтомобиля.Добавить();
ЗаполнитьЗначенияСвойств(НовЗапись,Выборка);
```

Листинг 10.3.15

Обратите внимание, что мы не заполнили свойство записи *Регистратор*. В этом нет смысла, платформа сама автоматически присвоит записи нужный регистратор (текущий документ).

Проверьте, как будет проводиться документ после наших изменений.

Резюме

На этом мы закончим изучение регистров накопления. Мы научились создавать наборы данных, добавлять в них строки и очищать. Рекомендую не злоупотреблять данными знаниями, а применять их только в исключительных случаях. Все записи в регистрах накопления должны создаваться только документами. Как добавлять записи при проведении документа - Вы знаете. В следующей части мы рассмотрим, как получать информацию из регистров накопления.

Часть 4. Регистры накопления. Получение данных

В заключительной части этой главы мы изучим, каким образом в программе 1С можно получать данные из регистров накопления. Точно так же, как и в случае с регистрами сведений, сделать это возможно двумя способами: используя методы менеджера регистров накопления и используя язык запросов.

Первоначально рассмотрим методы менеджера регистров накопления, это четыре метода: *Выбрать*, *Выбрать по регистратору*, *Обороты* и *Остатки*.

Выбрать

Метод *Выбрать* формирует выборку записей регистра и возвращает объект *Выборка регистров накопления*.

Он имеет следующий синтаксис:

Выбрать(<НачалоИнтервала>, <КонецИнтервала>, <Отбор>, <Порядок>)

Как Вы уже поняли, он аналогичен методу *Выбрать* для регистров сведений. Также он имеет четыре параметра.

Начало интервала – дата, с которой начинается выборка.

Конец интервала – дата, которой оканчивается выборка.

Отбор – структура, которая может содержать только один элемент, по значению которого будет осуществлен отбор в выборке. Название ключа структуры должно совпадать с названием измерения (или реквизита), по которому будет вестись отбор. И у данного измерения признак *Индексирование* должен быть установлен в значение *Индексировать*.

Порядок – строка с названием поля, по которому будет отсортирована выборка.

Продемонстрируем работу данного метода: выведем в таблицу значений все записи регистра накопления *Топливо в автомобилях* по указанному на форме автомобилю. Для этого необходимо у измерения *Автомобиль* установить свойство *Индексировать*.

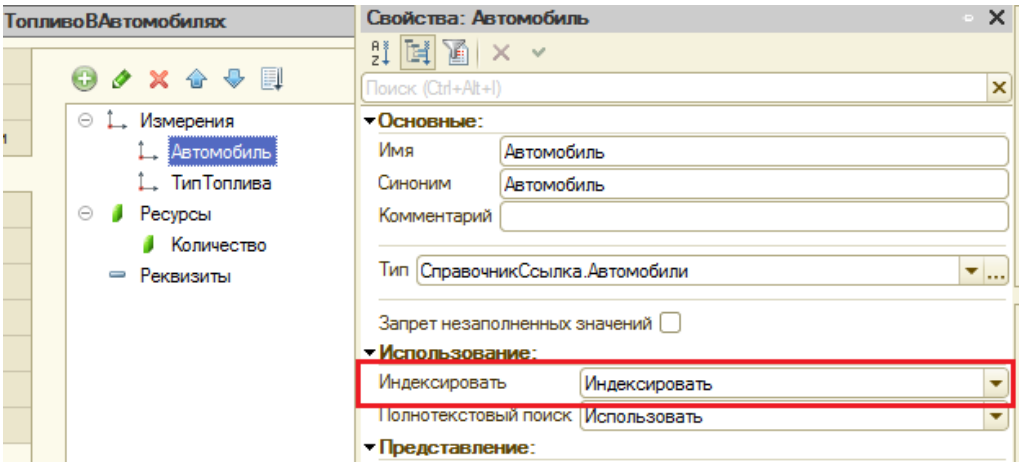


Рис. 10.4.1

Создайте обработку, форму обработки, на которой разместите два реквизита: реквизит *Автомобиль* с соответствующим типом и таблицу значений – *Топливо в автомобилях*. У таблицы значений создайте следующие колонки: *ПриходРасход* (тип значения выберите из системного перечисления *Вид движения накопления*, см. рис. 10.4.2), *ТипТоплива*, *Регистратор* (у этой колонки тип сделайте составным: *ДокументСсылка.ЗаправкаТоплива и ДокументСсылка.ОтчетОРасходеТоплива*) и *Количество*. А также команду «Заполнить таблицу», которую разместите в командной панели таблицы формы.

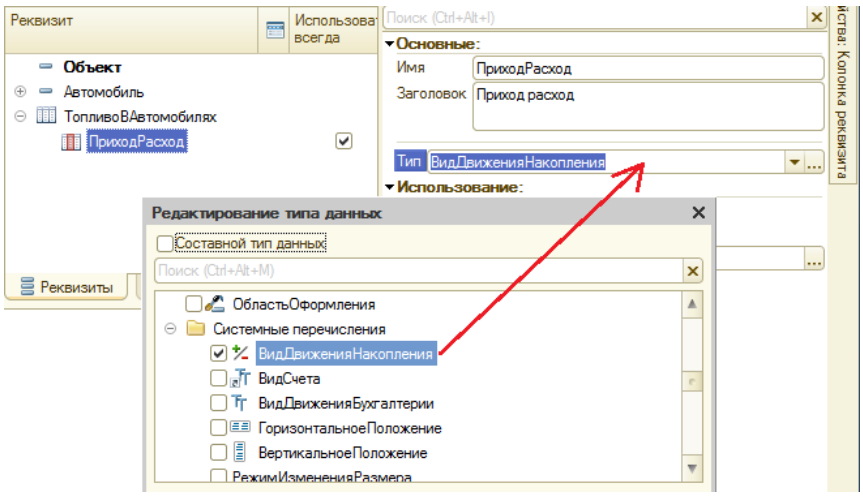


Рис. 10.4.2

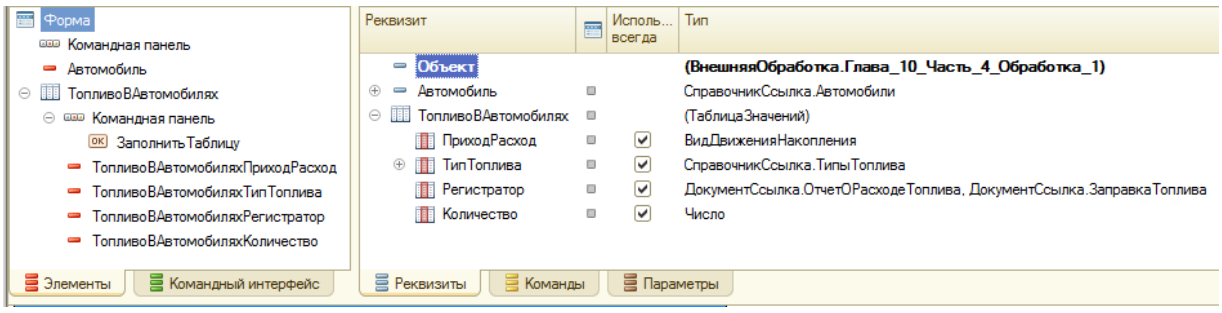


Рис. 10.4.3

В серверном обработчике команды напишем код, который будет осуществлять выборку с отбором по гаражу и заполнять табличное поле. В данной выборке у Вас не будет ограничений по датам.

```
&НаСервере
Процедура ЗаполнитьТаблицуНаСервере()
    ТопливоВАвтомобилях.Очистить();
    Отбор = Новый Структура;
    Отбор.Вставить("Автомобиль",Автомобиль);
    Выборка = РегистрыНакопления.ТопливоВАвтомобилях.Выбрать(,,Отбор);
    Пока Выборка.Следующий() цикл
        НоваяСтрока = ТопливоВАвтомобилях.Добавить();
        НоваяСтрока.ПриходРасход = Выборка.ВидДвижения;
        ЗаполнитьЗначенияСвойств(НоваяСтрока,Выборка);
    КонецЦикла;
КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьТаблицу(Команда)
    Если Не ЗначениеЗаполнено(Автомобиль)Тогда
        Сообщить("Не выбран автомобиль");
        Возврат
    КонецЕсли;
    ЗаполнитьТаблицуНаСервере();
КонецПроцедуры
```

Листинг 10.4.1

В этом коде мы получили выборку регистра накопления *Топливо в автомобилях*, с установленным отбором по гаражу. Затем в цикле выборки мы создаем новые строки таблицы значений *ТопливоВАвтомобилях* и заполняем их.

Посмотрите, как работает Ваша обработка.

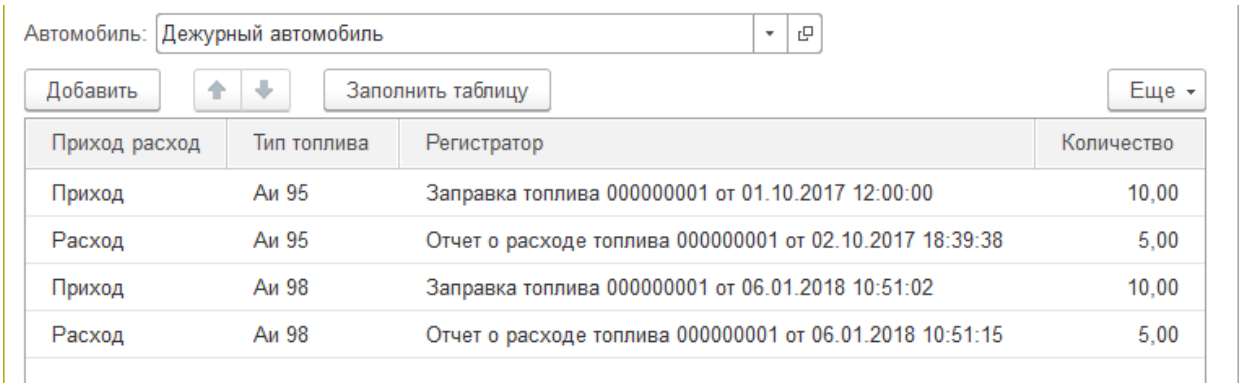


Рис. 10.4.4

Самостоятельно сделайте следующие доработки: разместите на форме стандартный период и из него подставляйте даты в соответствующие параметры метода *Выбрать*. Если дата пустая, то нужный параметр должен отсутствовать. В колонке *ПриходРасход* таблицы значений поменяйте тип значения на *строку*. Если вид движения *Приход*, то ставьте плюс, иначе - минус.

Выбрать по регистратору

Рассмотрим следующий метод - *Выбрать по регистратору*. Это простой метод, имеет всего один параметр – ссылка на документ, который может быть регистратором какой-нибудь записи регистра накопления.

Для его демонстрации изменим предыдущую Вашу форму: добавим реквизит *Регистратор*, с составным типом (Документы: *Прибытие в гараж* и *Выбытие из гаража*, см. рис. 10.4.5). Доработаем процедуру-обработчик команды: если реквизит «Регистратор» заполнен, то осуществляем выборку по регистратору, а иначе обычную выборку. Отбор по автомобилю должен работать только в случае обычной выборки. Для учебных целей я для этого сделаю отдельную команду «Заполнить таблицу по регистратору».

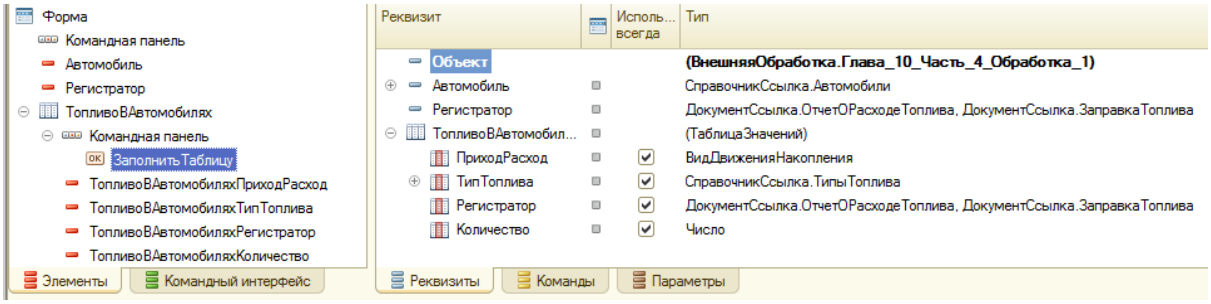


Рис. 10.4.5

```
&НаСервере
Процедура ЗаполнитьТаблицуПоРегистраторуНаСервере()
    ТопливоВАвтомобилях.Очистить();
    Если ЗначениеЗаполнено(Регистратор) тогда
        Выборка =
РегистрыНакопления.ТопливоВАвтомобилях.ВыбратьПоРегистратору(Регистратор);
    иначе
        Если Не ЗначениеЗаполнено(Автомобиль) тогда
            Возврат;
        КонецЕсли;
        Отбор = Новый Структура;
        Отбор.Вставить("Автомобиль",Автомобиль);
        Выборка = РегистрыНакопления.ТопливоВАвтомобилях.Выбрать(,,Отбор);
    КонецЕсли;
    Пока Выборка.Следующий() цикл
        НоваяСтрока = ТопливоВАвтомобилях.Добавить();
        ЗаполнитьЗначенияСвойств(НоваяСтрока,Выборка);
        НоваяСтрока.ПриходРасход = Выборка.ВидДвижения;
    КонецЦикла;
КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьТаблицуПоРегистратору(Команда)
    ЗаполнитьТаблицуПоРегистраторуНаСервере();
КонецПроцедуры
```

Листинг 10.4.2

В данном случае, если *Регистратор* не пустой, то выборка будет идти по регистратору, причем в качестве параметра будет указываться тот самый *Регистратор*. А иначе все как было раньше. Проверьте, как работает данный метод.

Только что мы рассмотрели общие методы, которые также есть и в регистрах сведений. Ниже мы будем рассматривать те методы, которые присущи только регистрам накопления. И начнем мы с метода *Обороты*.

Обороты

Данный метод позволяет получить обороты регистра накопления за заданный период.

Что понимается под словом *«обороты»*? Например, у нас есть регистр *Пробег автомобиля*, и нам необходимо узнать, какой пробег был у конкретного автомобиля, скажем, за месяц или за неделю. Для этого необходимо просуммировать все записи регистра накопления по данному автомобилю за нужный период. Эта сумма и будет *оборотом* регистра накопления по данному автомобилю. Можно получить *общий оборот* – по всем автомобилям, по которым прошли движения. Или оборот по конкретному измерению. Обороты можно получать как по оборотным регистрам, так и по регистрам остатков. В первом случае будет выводиться сумма оборота по какому-нибудь измерению, а во втором случае - общая сумма прихода и расхода.

Закончим с теорией, рассмотрим данный метод на практике. И первым делом изучим его синтаксис.

Обороты(<НачалоПериода>, <КонецПериода>, <Отбор>, <Измерения>, <Ресурсы>)

Первые два параметра, *НачалоПериода* и *КонецПериода*, не должны вызвать у Вас вопросов.

Отбор - это структура, которая содержит значение измерения, по которому должен осуществиться отбор. Если он указан, то обороты будут подсчитаны только для указанного измерения. Например, это могут быть обороты по конкретному типу топлива (для регистра *ТопливоВАвтомобилях*).

Измерения - это строка, в которой перечислены измерения через запятую, по которым следует разворачивать обороты. Например, можно указать *ТипТоплива*, тогда будут подсчитаны обороты только по гаражам.

Ресурсы – это строка, в которой перечислены через запятую ресурсы, по которым будут получены обороты.

Данный метод возвращает таблицу значений, которая содержит колонки с измерениями, указанными в параметре *Измерения*. Названия этих колонок совпадают с названиями измерений. И с колонками ресурсов, которые указаны в параметре *Ресурсы*.

Для оборотных регистров будет одна колонка на ресурс, с названием ресурса. А для регистров остатков будет по две колонки на ресурс, которые будут иметь следующие названия:

"<Имя ресурса>Приход" и "<Имя ресурса>Расход"

Теперь создайте обработку, на форме которой разместите два реквизита с типом *ТаблицаЗначений*: в первой таблице будем получать обороты по *Автомобилю* регистра накопления *ТопливоВАвтомобилях* (в таблице создадим три колонки *ТипТоплива*, *Прибыло*, *Убыло*). Во второй таблице будем получать пробег указанного автомобиля из регистра накопления «ПробегАвтомобиля» (в таблице создадим две колонки *Автомобиль*, *Пробег*). А также создадим реквизит с типом *СправочникСсылка.Автомобили* и реквизит с типом *СтандартныйПериод.*

На созданной форме добавим элемент «Группа» – страницы (со страницами *Расход Топлива* и *Пробег*) и разместим две таблицы значений в разных страницах.

А также создадим две команды: «Заполнить расход» и «Заполнить пробег», которые разместим в командных панелях соответствующих таблиц формы.

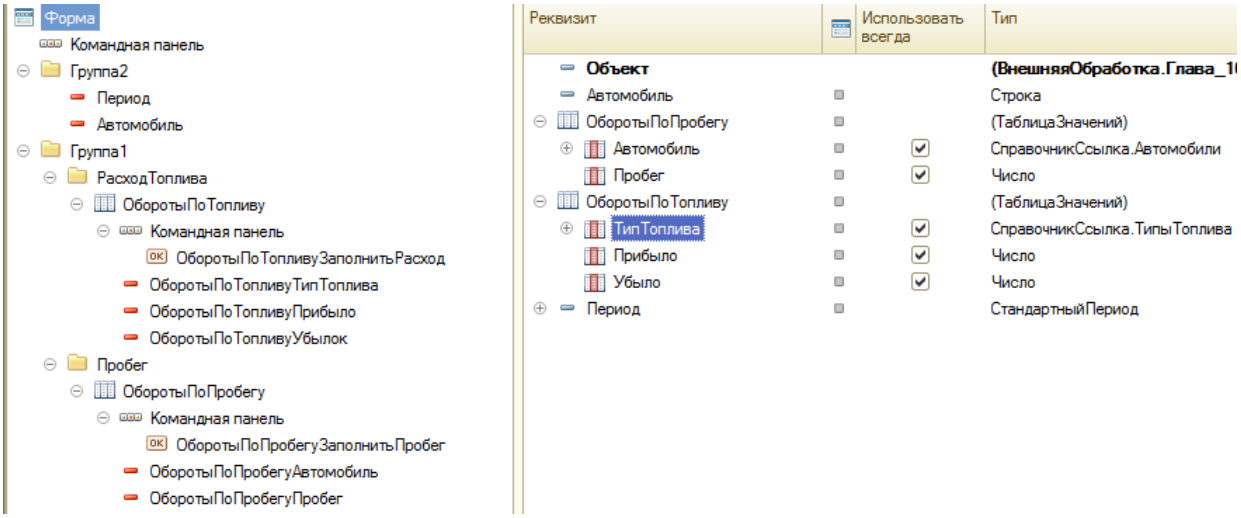


Рис. 10.4.6

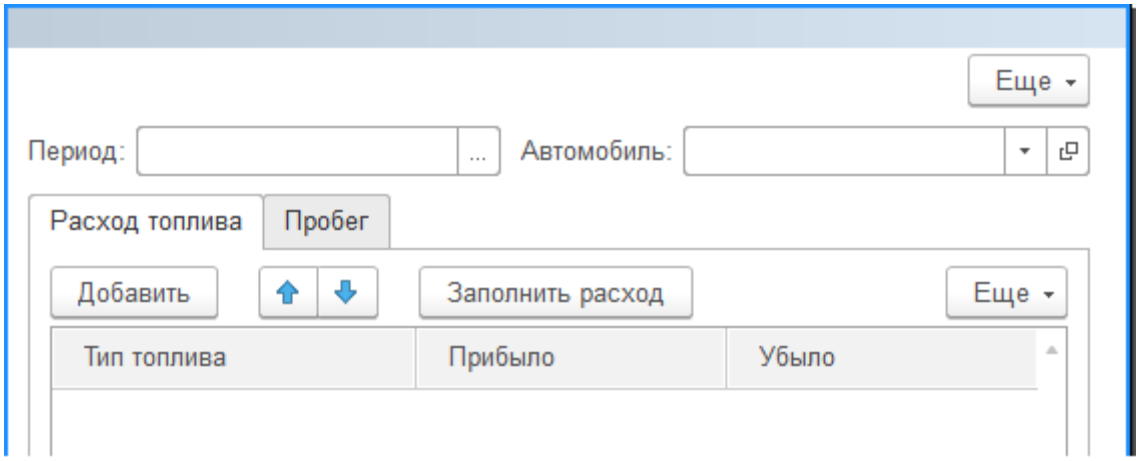


Рис. 10.4.7

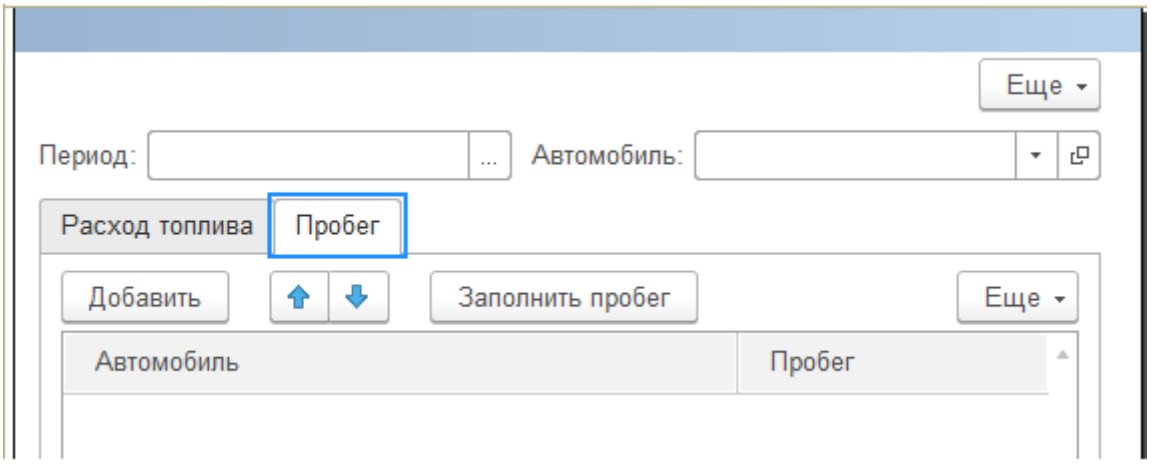


Рис. 10.4.8

Теперь в серверном обработчике команды «Заполнить расход» напишем код, получающий обороты по автомобилю из регистра накопления *ТопливоВАвтомбилях* и загружающий полученный результат в таблицу значений.

```
&НаСервере
Процедура ЗаполнитьРасходНаСервере()
    ОборотыПоТопливу.Очистить();
    Отбор = Новый Структура;
    Отбор.Вставить("Автомобиль",Автомобиль);
    МенТоплВАвто = РегистрыНакопления.ТопливоВАвтомобилях;
    ТаблицаОборотов = МенТоплВАвто.Обороты(Период.ДатаНачала,
                                           Период.ДатаОкончания,
                                           Отбор,
                                           "ТипТоплива");
    Для Каждого СтрОборот из ТаблицаОборотов цикл
        НоваяСтрока = ОборотыПоТопливу.Добавить();
        НоваяСтрока.ТипТоплива = СтрОборот.ТипТоплива;
        НоваяСтрока.Прибыло = СтрОборот.КоличествоПриход;
        НоваяСтрока.Убыло   = СтрОборот.КоличествоРасход;
    КонецЦикла;
КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьРасход(Команда)
    Если Не ЗначениеЗаполнено(Автомобиль) Тогда
        возврат;
    КонецЕсли;
    ЗаполнитьРасходНаСервере();
КонецПроцедуры
```

Листинг 10.4.3

Разберем данный код.

Первым делом мы создаем структуру, название ключа которой совпадает с названием измерения, и позже мы укажем ее в третьем параметре. Следующим шагом мы получаем с помощью метода *Обороты* менеджера регистра накопления *ТопливоВАвтомобилях* таблицу оборотов. В этот метод мы передаем следующие параметры: начало и конец периода, структуру

для отбора, созданную ранее. А в четвертом параметре идет строка с надписью *«ТипТоплива»*. Это значит, что обороты будут подсчитаны только по измерению *ТипТоплива*.

После того, как мы получили обороты, заполняем таблицу *Обороты по топливу* значениями из полученной таблицы.

Обратите внимание на название полей ресурсов в таблице оборотов.

Сохраните новую обработку и посмотрите, как она работает, касательно таблицы *Обороты по топливу*.

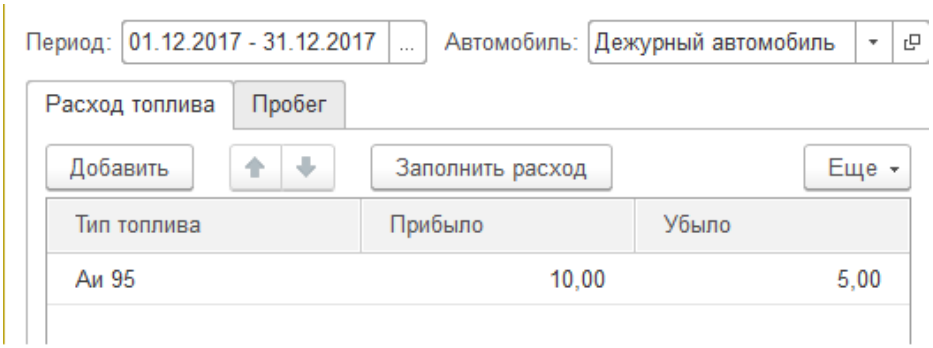


Рис. 10.4.9

Как видите, мы получили информацию, сколько конкретного топлива было заправлено в выбранный автомобиль и о каком количестве топлива отчитались. Проверьте ее самостоятельно.

Подсчитаем обороты по пробегу. Напишите в обработчике команды «Заполнить пробег» следующий код:

```
&НаСервере
Процедура ЗаполнитьПробегНаСервере()
    ОборотыПоПробегу.Очистить();
    Отбор = Новый Структура;
    Отбор.Вставить("Автомобиль",Автомобиль);
    МенПробег = РегистрыНакопления.ПробегАвтомобиля;
    ТаблицаОборотов = МенПробег.Обороты(Период.ДатаНачала,
                                        Период.ДатаОкончания,
                                        Отбор,
                                        "Автомобиль");
    Для Каждого СтрОборот из ТаблицаОборотов цикл
        НоваяСтрока = ОборотыПоПробегу.Добавить();
        НоваяСтрока.Автомобиль = СтрОборот.Автомобиль;
        НоваяСтрока.Пробег     = СтрОборот.Пробег;
    КонецЦикла;
КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьПробег(Команда)
    Если Не ЗначениеЗаполнено(Автомобиль) Тогда
        возврат;
    КонецЕсли;
    ЗаполнитьПробегНаСервере();
КонецПроцедуры
```

Листинг 10.4.4

Прокомментируем данный код. Первым делом мы создаем структуру для отбора, название ключа которой совпадает с названием измерения, ниже мы укажем её в третьем параметре. После этого, используя метод *Обороты* менеджера регистра накопления *ПробегАвтомобиля*, получаем обороты этого регистра по измерению *Автомобиль* для указанного выше отбора. И в конце заполняем таблицу значений *ОборотыПоПробегу*.

Теперь посмотрите, как работает данный код, и проверьте, правильные ли данные выведены в табличное поле.

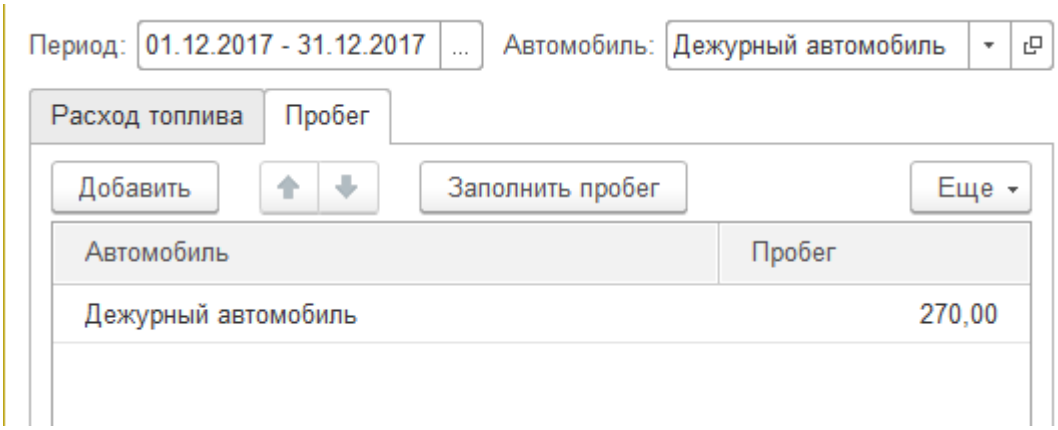


Рис. 10.4.10

Последний метод менеджера регистров накоплений, который мы рассмотрим в данной части, это метод *Остатки*.

Остатки

Данный метод применим только к регистрам *остатков*, поэтому его работу мы будем рассматривать на примере регистра накопления *ТопливоВАвтомобилях*.

Вкратце объясню, что такое *Остатки*. Если у нас в автомобиль было залито десять литров топлива, а было истрачено два (по ним есть отчет), то оставшиеся восемь литров топлива - это и будет остаток топлива по данному автомобилю. Остаток может быть положительным, а может и отрицательным. Если, например, залито три литра топлива, а истрачено пять, то остаток будет минус два литра топлива по автомобилю. Фактически это невозможно, но программа позволяет такое делать, и на практике часто это встречается, особенно когда не ведется контроль отрицательных остатков и есть привычка в оперативном учете заводить документы задним числом. Что, конечно же, неправильно и должно пресекаться!

Для регистров накопления можно получать остатки по всем измерениями, которые есть, также можно получать остатки по комбинации измерений. В нашем случае можно получить остаток по измерению *ТипТоплива*, в этом случае в остатки будут аккумулироваться остатки по всем автомобилям, в которые залито в данный момент какое-то топливо. Мы узнаем точное количество остатка топлива каждого типа во всех автомобилях. Также можно получить остаток по

комбинации *Автомобиль - ТипТоплива*, тогда мы узнаем, у какого автомобиля какое конкретное топливо залито в данный момент.

Рассмотрим синтаксис метода *Остатки* регистра накопления.

Остатки(<МоментВремени>, <Отбор>, <Измерения>, <Ресурсы>)

Момент времени – это дата, на которую мы будем подсчитывать остатки. К примеру, количество топлива в автомобиле на начало месяца и на текущий день может быть разное, тогда эту дату необходимо указывать. Если она не указана, то берутся остатки на текущую дату.

Отбор – это структура, которая содержит значение измерения, по которому должен осуществиться отбор. Если она есть, то остаток будет подсчитан только для указанного в отборе измерения. Например, это могут быть остаток по конкретному типу топлива или конкретному автомобилю.

Измерения - это строка, где через запятую перечислены измерения, по которым будут подсчитаны остатки. Если указано несколько измерений, например, *Автомобиль* и *ТипТоплива*, то будут подсчитаны остатки по всем типам топлива и по всем автомобилям.

Ресурсы - это строка, в которой через запятую перечислены ресурсы, для которых нужно получить остатки.

Данный метод возвращает таблицу значений со следующими колонками: колонки, указанные в третьем параметре (если не указан, то все измерения), и колонки, которые указаны в четвертом параметре (если не указан, то все ресурсы).

Для демонстрации работы данного метода, создайте обработку с формой, на которой разместите таблицу значений, где будут выводиться остатки по всем типам топлива. Также предусмотрим возможность отбора по типу топлива (создадим отдельный реквизит). И создадим команду «Заполнить остатки», при выполнении которой будем заполнять остатки по типам топлива.

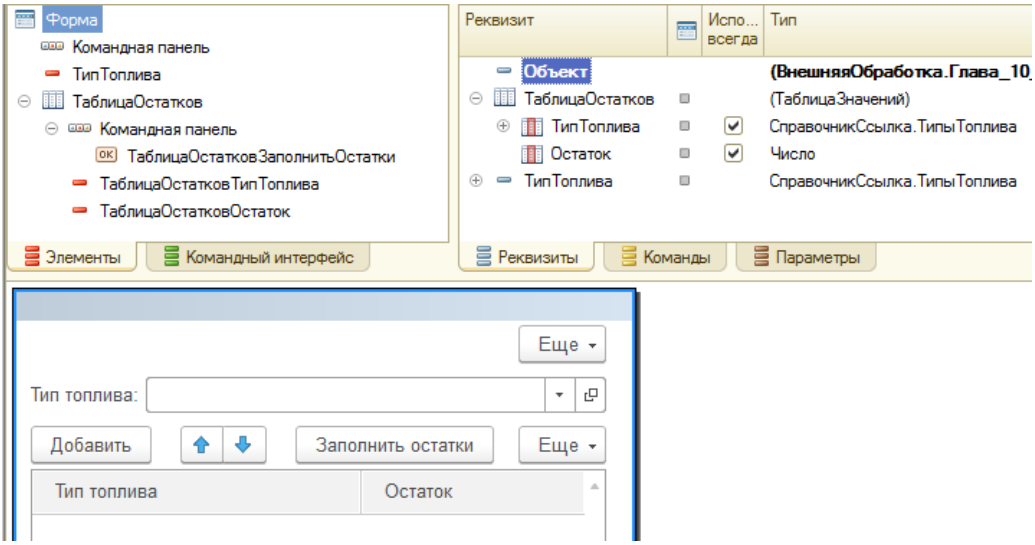


Рис. 10.4.11

В обработчике команды напишем следующий код:

```
&НаСервере
Процедура ЗаполнитьОстаткиНаСервере()
    ТаблицаОстатков.Очистить();
    МенТопливоВАвто = РегистрыНакопления.ТопливоВАвтомобилях;
    Если Не ЗначениеЗаполнено(ТипТоплива) тогда
        Остатки = МенТопливоВАвто.Остатки(,,"ТипТоплива",);
    иначе
        Отбор = новый Структура;
        Отбор.Вставить("ТипТоплива",ТипТоплива);
        Остатки = МенТопливоВАвто.Остатки(,Отбор,"ТипТоплива",);
    КонецЕсли;
    Для Каждого СтрокаТабл из Остатки цикл
        НоваяСтрока = ТаблицаОстатков.Добавить();
        НоваяСтрока.ТипТоплива = СтрокаТабл.ТипТоплива;
        НоваяСтрока.Остаток = СтрокаТабл.Количество;
    КонецЦикла;
КонецПроцедуры

&НаКлиенте
Процедура ЗаполнитьОстатки(Команда)
    ЗаполнитьОстаткиНаСервере();
КонецПроцедуры
```

Листинг 10.4.5

В данном случае мы не стали устанавливать момент времени, предполагая, что нам нужны текущие остатки.

В том случае, если поле *ТипТоплива* пустое, то используем метод *Остатки* только с одним параметром, это третий параметр, в котором перечислены измерения. Мы указываем измерение *ТипТоплива,* поскольку, по условиям задачи, нам необходимо получить остатки только в разрезе типа топлива. Когда же поле *ТипТоплива* не пустое, то создаем структуру, в которой указываем выбранный тип топлива. Эту структуру указываем в качестве второго параметра. После этого мы заполняем таблицу остатков, предварительно очистив ее.

Проверьте работу данного метода.

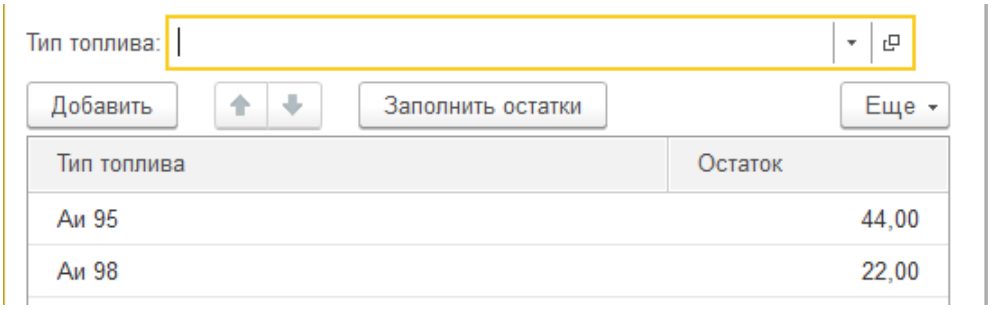


Рис. 10.4.12

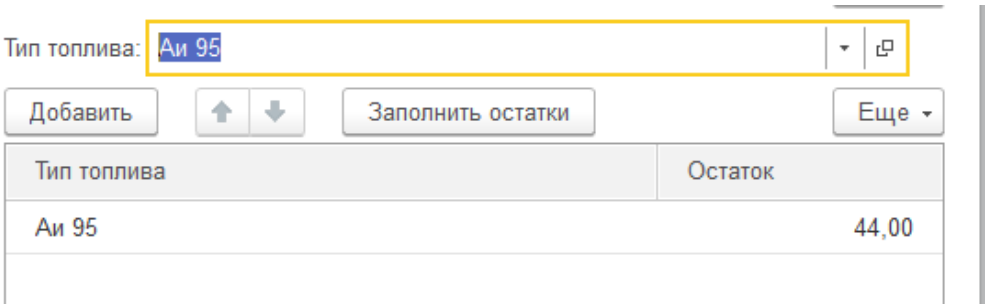


Рис. 10.4.13

Допишите данную обработку: пусть выходят остатки в разрезе автомобилей и типов топлива (добавьте новую колонку *Автомобиль* в таблицу значений с соответствующим типом).

Мы изучили методы менеджера регистров накопления, теперь перейдем к языку запросов. С помощью языка запросов платформы 1С можно получить выборку записей регистра накопления, мы на этом не будем останавливаться, т.к. здесь все по аналогии с регистрами сведений. Мы изучим основы использования виртуальных таблиц, с помощью них можно получить остатки, обороты и остатки с оборотами.

Рассмотрим первую виртуальную таблицу *Остатки*.

Виртуальная таблица Остатки

Виртуальная таблица языка запросов 1С *Остатки* может быть создана у регистров накопления, имеющих вид *Остатки*. Она также, как и метод *Остатки* менеджера регистра накопления, возвращает остаток ресурса в разрезе определенных измерений

Решим следующую задачу: на форме разместите поля ввода: *Топливо*, *Автомобиль*, *Дата* и *Остаток топлива*. На указанную дату необходимо получить остаток регистра накопления *ТопливоВАвтомобилях* по указанному типу топлива и указанному автомобилю.

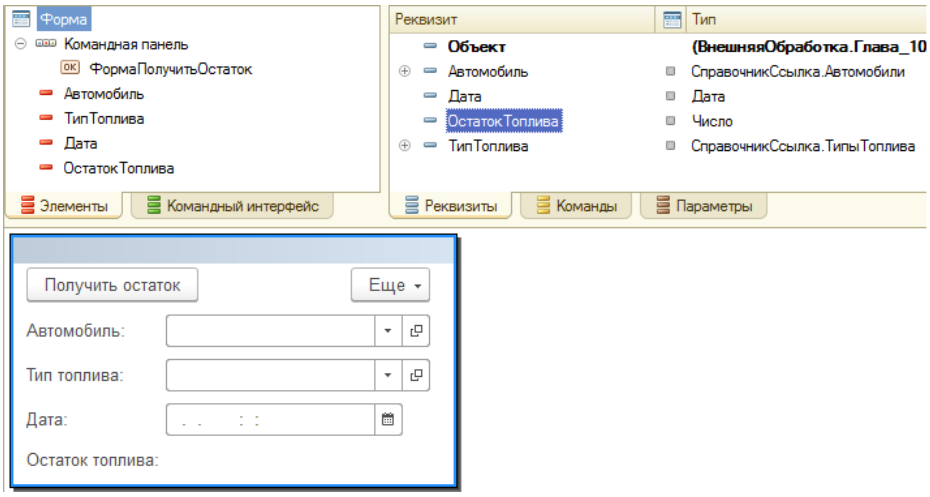


Рис. 10.4.14

Создадим команду формы «Получить остаток» и в серверном обработчике команды создадим пустой запрос и откроем конструктор запроса. Разверните дерево регистров накопления и обратите внимание на таблицу *Топливо в автомобилях Остатки*.

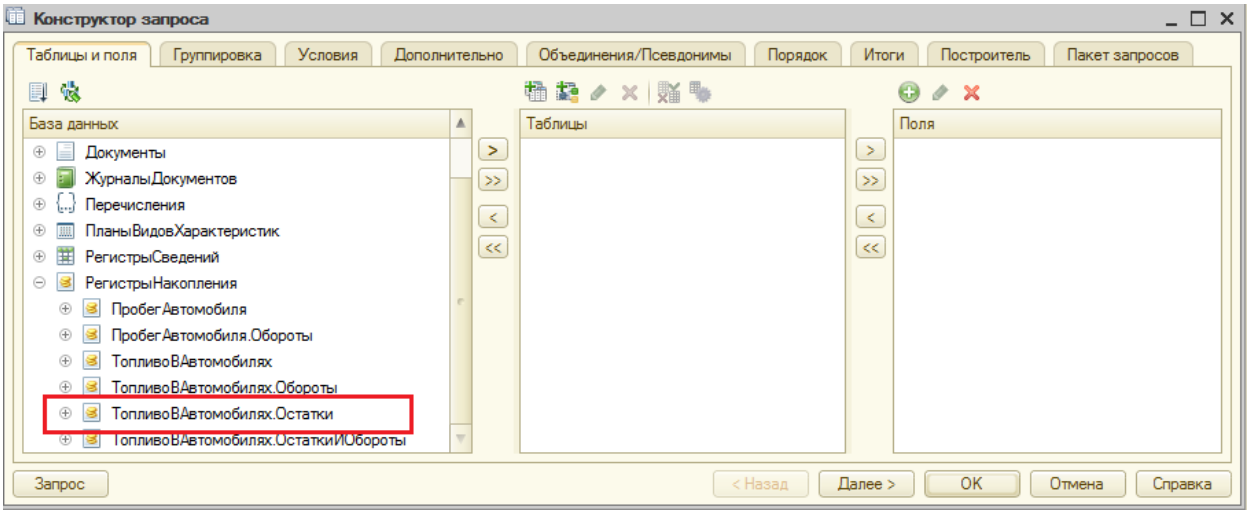


Рис. 10.4.15

Выберете ее. Теперь Вам необходимо задать отбор по автомобилю и типу топлива. Для этого нажмите на кнопку *«Параметры виртуальной таблицы»* (таблица должна быть выделена).

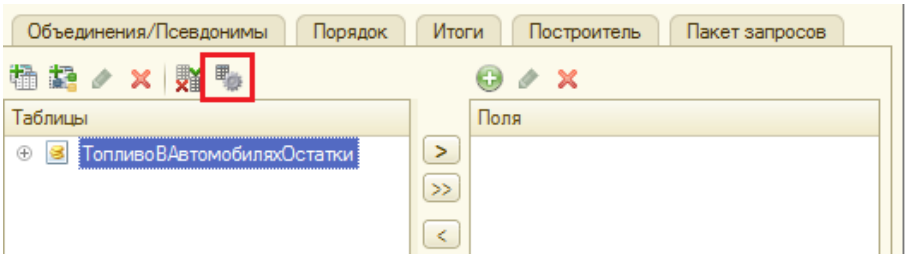


Рис. 10.4.16

Откроется окно *«Параметры виртуальной таблицы»*.

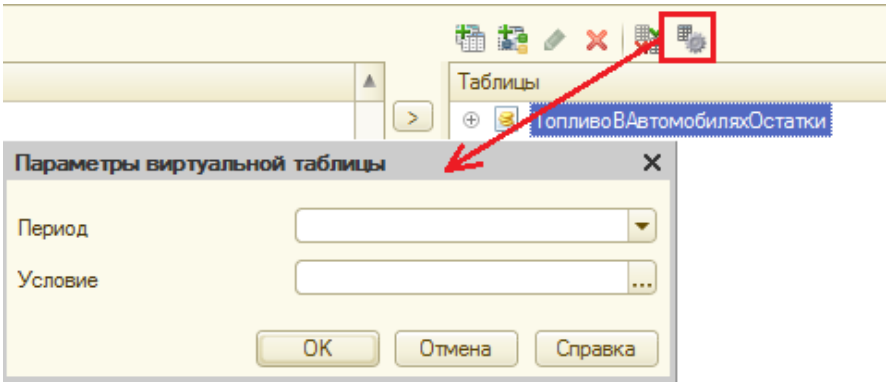


Рис. 10.4.17

Заносите дату, на которую мы будем получать остатки, в поле *«Период»*. В виде параметра *&Дата* в этот параметр мы и передадим дату остатков перед выполнением запроса.

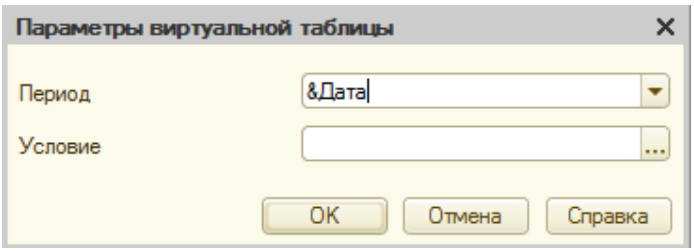


Рис. 10.4.18

Раскроем поле *Условие* и поставим отбор по автомобилю и типу топлива.

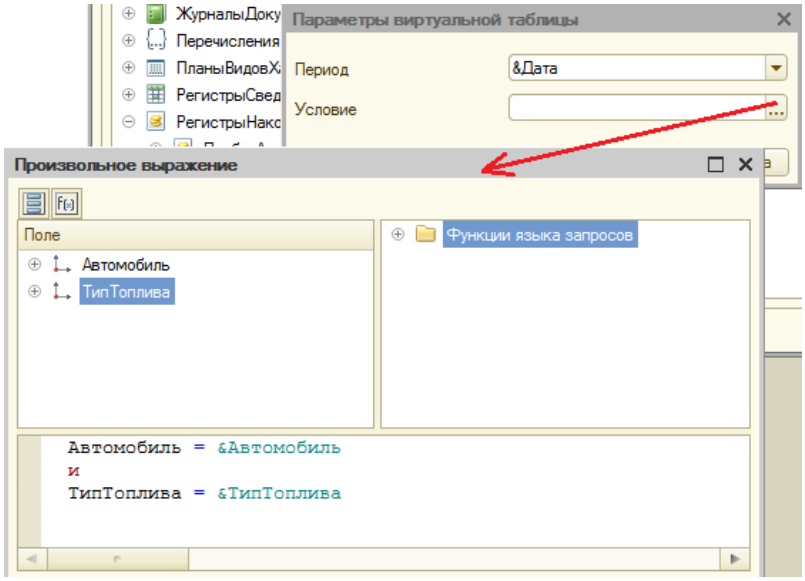


Рис. 10.4.19

В поле *Условие* параметров виртуальной таблицы *Остатки* регистров накоплений мы можем накладывать любые условия на измерения регистра. Также можно передать в это поле параметры запроса, чтобы наложить определенные ограничения на выборку данных.

При работе с виртуальными таблицами всегда накладывайте ограничения используя именно параметры виртуальной таблицы. Неправильно накладывать ограничения посредством директивы «ГДЕ» языка запросов! Это существенно скажется на производительности!

Сохраните установленные параметры. Теперь Вам необходимо выбрать только одно поле – *КоличествоОстаток*. Задайте псевдоним для этого поля – «Остаток».

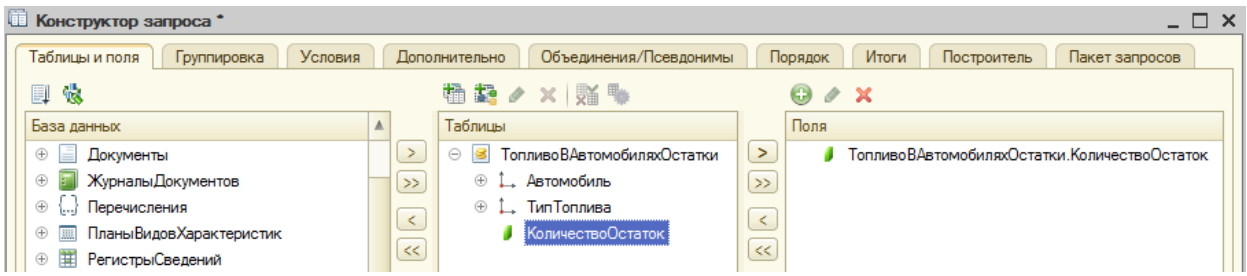


Рис. 10.4.20

Переименуем саму таблицу, чтобы запрос был более читаемый. Для этого нужно выделить саму таблицу, кликнуть правой кнопкой мышки и в открывшемся контекстном меню выбрать пункт «Переименовать таблицу…».

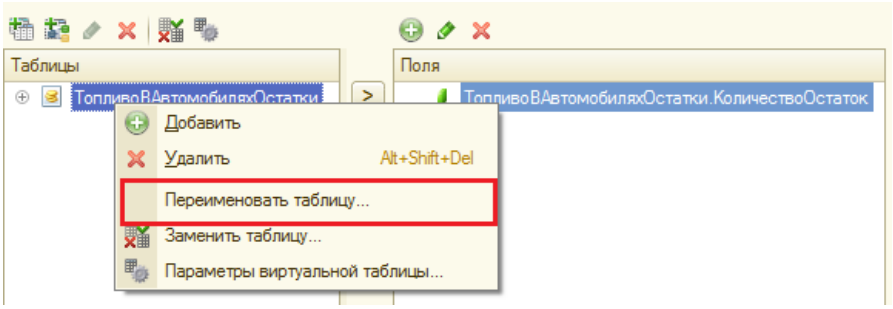


Рис. 10.4.21

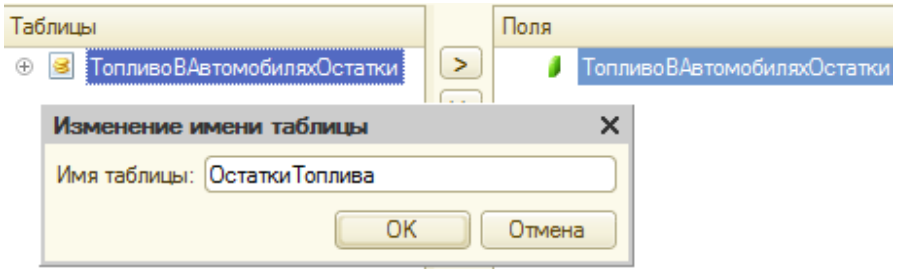


Рис. 10.4.22

После переименования нажмите кнопку *ОК* конструктора запроса, и Ваш запрос должен быть в следующем виде:

```
Запрос = Новый Запрос;
Запрос.Текст = "ВЫБРАТЬ
               |    ОстаткиТоплива.КоличествоОстаток КАК Остаток
               |ИЗ
               |    РегистрНакопления.ТопливоВАвтомобилях.Остатки(
               |            &Дата,
               |            Автомобиль = &Автомобиль
               |                И ТипТоплива = &ТипТоплива) КАК
ОстаткиТоплива";
```

Листинг 10.4.6

Установим параметры запроса.

```
Запрос.УстановитьПараметр("Дата",Дата);
Запрос.УстановитьПараметр("Автомобиль",Автомобиль);
Запрос.УстановитьПараметр("ТипТоплива",ТипТоплива);
```

Листинг 10.4.7

А также получим и выведем результат запроса.

```
Результат = Запрос.Выполнить();
Если Результат.Пустой() Тогда
    ОстатокТоплива = 0;
КонецЕсли;
```

```
Выборка = Результат.Выбрать();
Выборка.Следующий();
ОстатокТоплива = Выборка.Остаток;
```

Листинг 10.4.8

Поскольку у данного регистра всего два измерения, и по обоим мы установили отбор, то на выходе запроса должна быть только одна запись, т.к. не может быть двух разных остатков для одного типа топлива в одном автомобиле на один момент времени. Поэтому мы просто получили следующий элемент выборки, он же первый, и вывели информацию об остатке.

Проверьте, как работает данная обработка.

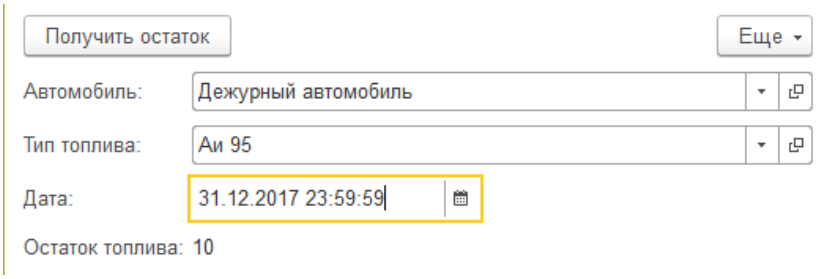


Рис. 10.4.23

Виртуальная таблица Обороты

Следующая виртуальная таблица, которую мы рассмотрим, называется *Обороты*. Это единственная таблица, которую можно применять и к оборотным регистрам, и к регистрам остатков.

Для демонстрации работы виртуальной таблицы сделаем обработку, на форму которой будем выводить пробег по дням выбранного автомобиля за указанный период.

Разместите на форме следующие элементы: реквизит *Автомобиль* (тип *СправочникСсылка.Автомобили*), реквизит *Период* (тип *СтандартныйПериод*) и таблицу значений *Пробег по дням* (с колонками *Период* и *Пробег*). А также команду «Получить пробег», которую разместите в командной панели таблицы формы.

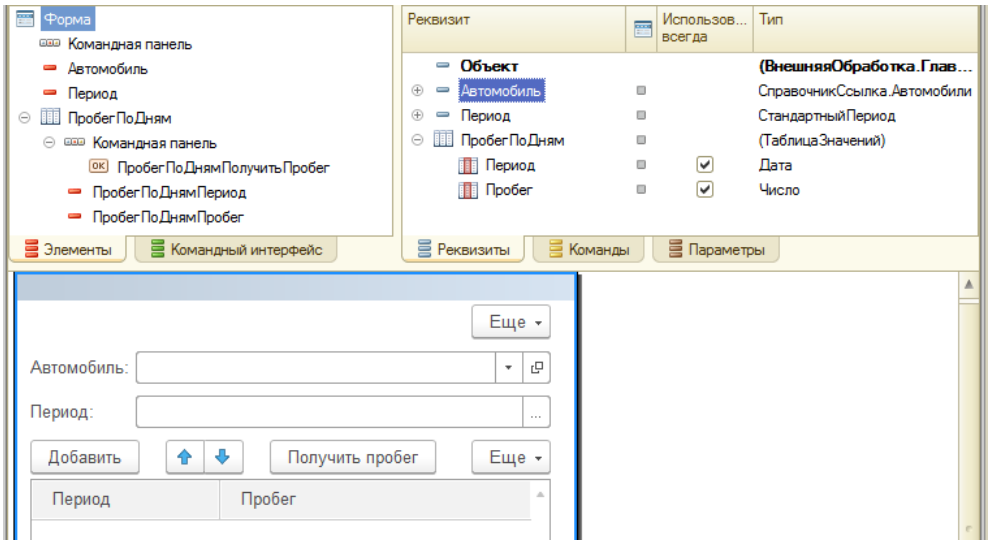


Рис. 10.4.24

В серверном обработчике команды создадим пустой запрос, откроем конструктор запроса, в котором раскроем дерево регистров накопления. Обратите внимание на виртуальную таблицу *Пробег автомобиля Обороты*.

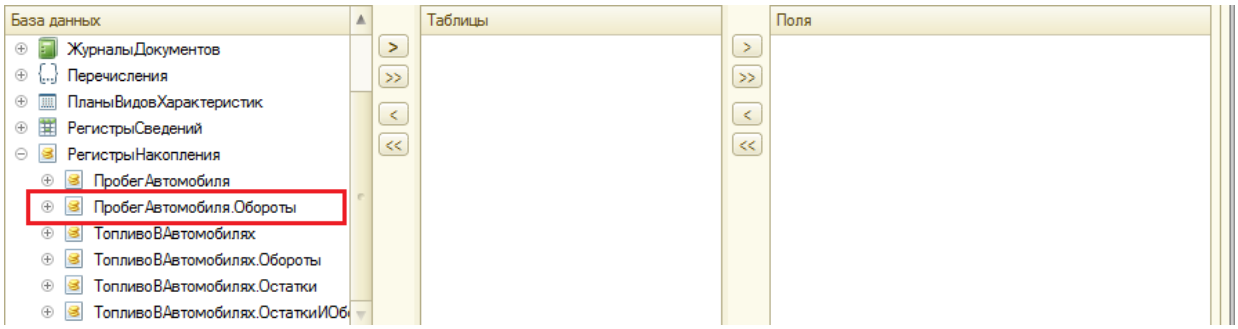


Рис. 10.4.25

Выберете ее. Теперь Вам необходимо установить параметры. Для этого нажмите на кнопку *Параметры виртуальной таблицы*. Обращаю Ваше внимание, что сама таблица должна быть выделена курсором и иметь синий фон.

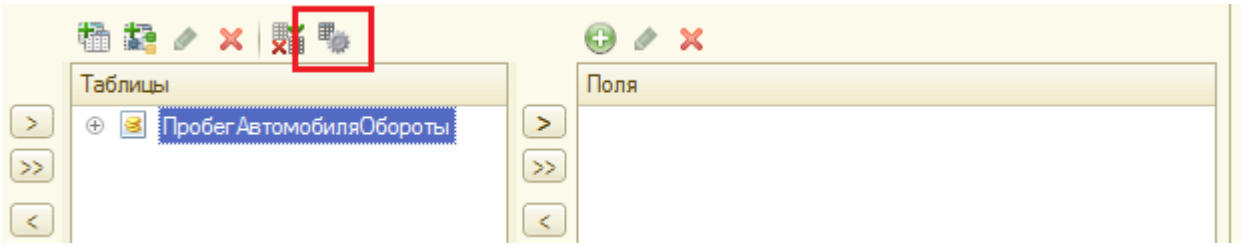


Рис. 10.4.26

Зайдите в параметры и установите начало периода и конец периода.

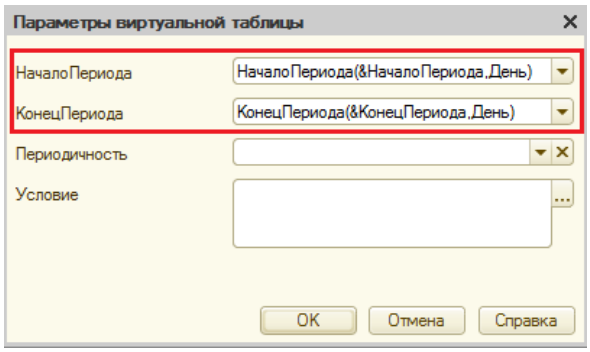


Рис. 10.4.27

Для этого мы используем процедуры *КонецПериода* и *НачалоПериода*, являющиеся процедурами языка запроса платформы 1С 8 и возвращающими начало или конец периода в зависимости от второго параметра. Если указан день, это будет начало или конец дня, если указан месяц - начало или конец месяца и т.д.

Теперь необходимо установить периодичность.

Периодичность - это то, с какой детальностью будут выводиться записи. Если мы, к примеру, установим периодичность *Месяц*, то выйдут обороты по автомобилям за месяц. Если установим периодичность *Год* – то за год. Также можно установить периодичность *Запись*, тогда будут выведены обороты по всем записям регистра накопления. Или периодичность *Регистратор*, тогда тоже будет периодичность по записям, но только с группировкой по регистратору. Нас интересуют обороты за день. Поэтому установите периодичность *День*.

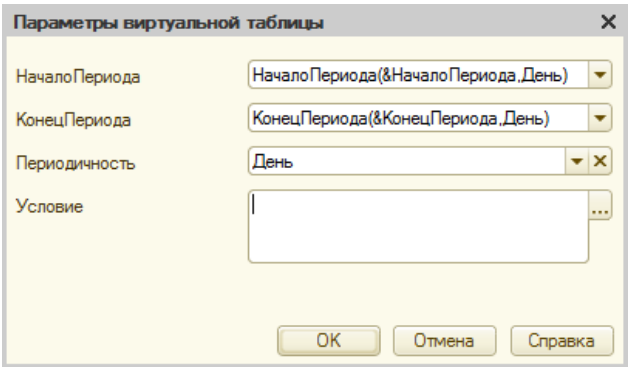


Рис. 10.4.28

Осталось задать условия.

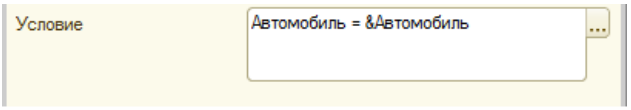


Рис. 10.4.29

Выберем поля: *Период* и *ПробегОборот* (для этого поля присвоим псевдоним *Пробег*). А саму таблицу переименуем в *ПробегАвто*.

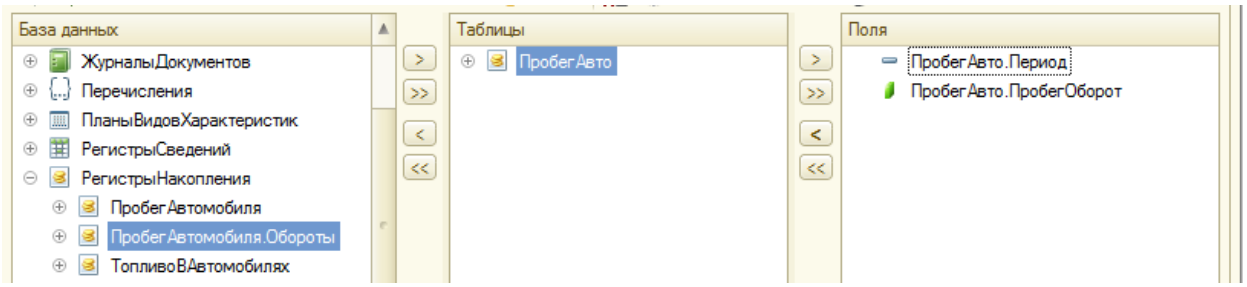


Рис. 10.4.30

Отсортируем результат, который выдаст виртуальная таблица по периоду.

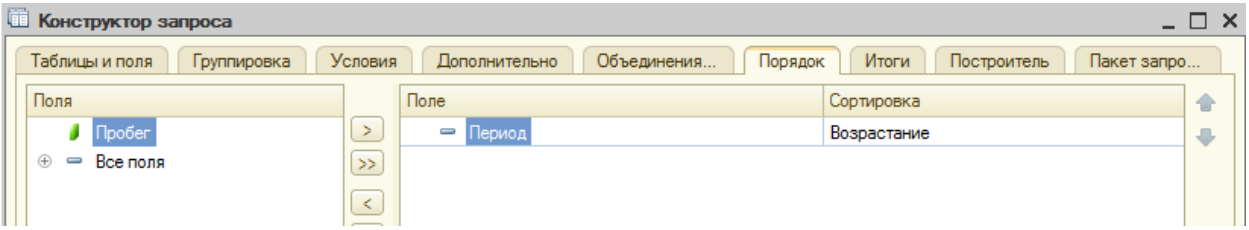


Рис. 10.4.31

Сохраните запрос. Он должен получиться такого вида:

```
ВЫБРАТЬ
    ПробегАвто.Период КАК Период,
    ПробегАвто.ПробегОборот КАК Пробег
ИЗ
    РегистрНакопления.ПробегАвтомобиля.Обороты(НАЧАЛОПЕРИОДА(&НачалоПериода, ДЕНЬ),
                                                КОНЕЦПЕРИОДА(&КонецПериода, ДЕНЬ),
                                                День,
                                                Автомобиль = &Автомобиль) КАК
ПробегАвто
УПОРЯДОЧИТЬ ПО
    Период
```

Листинг 10.4.9

Передадим в запрос параметры.

```
Запрос.УстановитьПараметр("НачалоПериода",Период.ДатаНачала);
Запрос.УстановитьПараметр("КонецПериода",Период.ДатаОкончания);
Запрос.УстановитьПараметр("Автомобиль",Автомобиль);
```

Листинг 10.4.10

Заполним таблицу значений при помощи выборки запроса.

```
Выборка = Запрос.Выполнить().Выбрать();
Пока Выборка.Следующий() Цикл
    НовСтр = ПробегПоДням.Добавить();
    ЗаполнитьЗначенияСвойств(НовСтр,Выборка);
КонецЦикла;
```

Листинг 10.4.11

Проверьте, как работает Ваша обработка.

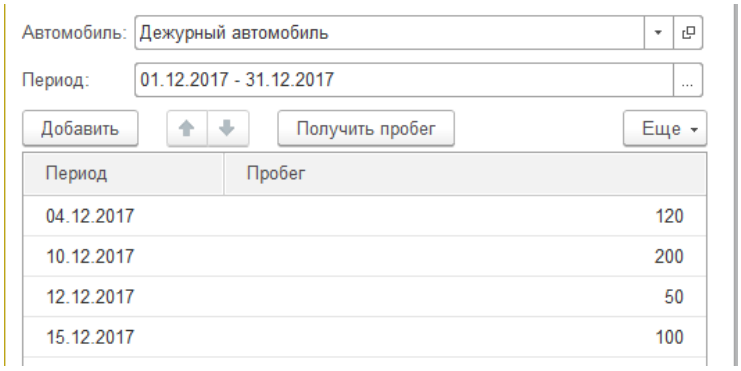


Рис. 10.4.32

Вы видите, что вывелся пробег автомобиля за каждый день, когда была запись в регистре. Зайдите обратно в запрос и измените в виртуальной таблице периодичность на *Неделю*.

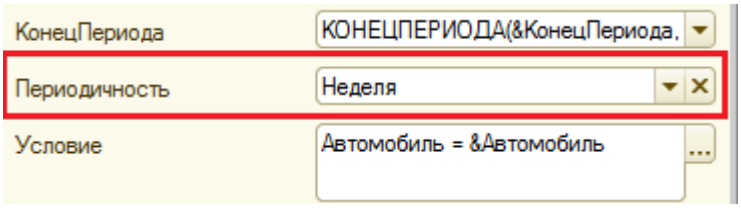


Рис. 10.4.33

Посмотрите, какие данные будут на выходе:

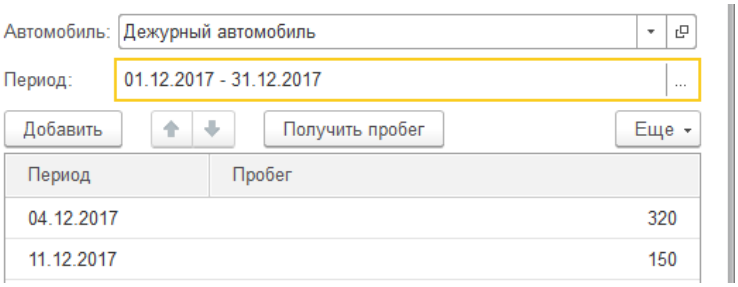


Рис. 10.4.34

Запрос сгруппировал все данные по неделям.

Таким образом, делаем вывод, что параметр *Периодичность*, по сути, определяет уровень группировки данных по строкам в разрезе дат. Посмотрим, что будет, если его вообще обнулить: зайдите в запрос и уберите период в параметрах.

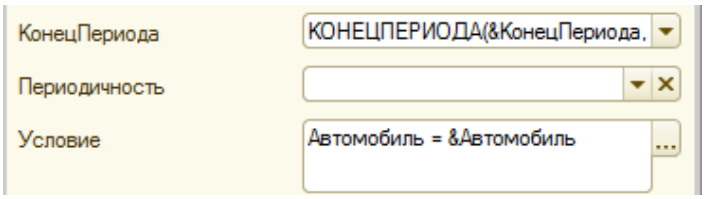


Рис. 10.4.35

Из списка полей сразу же убралось поле *Период*. Сохраните запрос, обработку и посмотрите на результат.

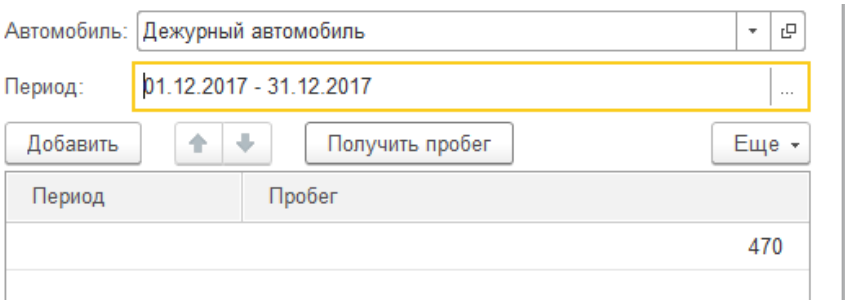


Рис. 10.4.36

Видим, что вышел общий итог за весь выбранный период.

И еще момент. Мы рассмотрели работу виртуальной таблицы *Обороты* на примере оборотного регистра. Регистры остатков тоже могут работать с данной таблицей, только помимо ресурсного поля с приставкой *оборот* появляются поля с приставками *Приход* и *Расход*.

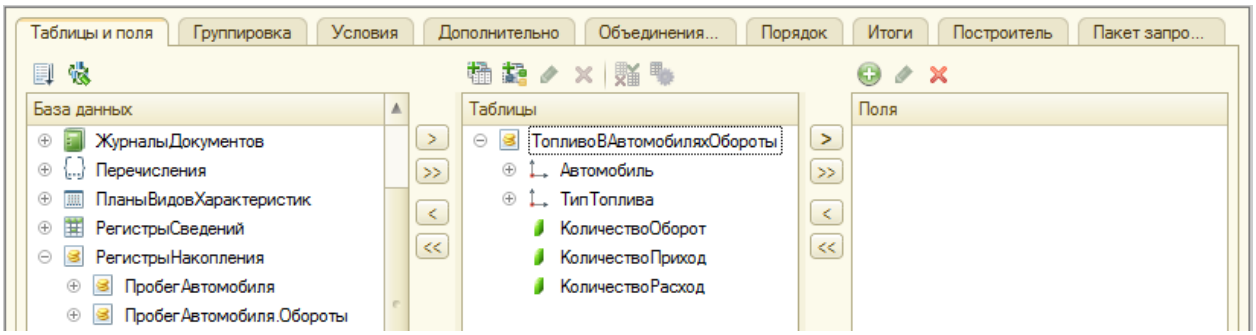


Рис. 10.4.37

Теперь рассмотрим следующую виртуальную таблицу - это *Остатки и обороты*.

Виртуальная таблица Остатки и обороты

Сделаем обработку, в которой будем выводить в таблицу поля с данными по заправке топлива в автомобиль и расходу этого топлива, а также остатки на начало и конец периода.

Разместите на форме обработки следующие реквизиты: период, *Автомобиль* и таблицу значений *ТопливоВАвтомобиле* (с колонками: *ТипТоплива*, *Остаток на начало*, *Заправка*, *Расход*, *Остаток на конец*). Создайте команду «Заполнить таблицу», которую разместите в командной панели таблицы формы.

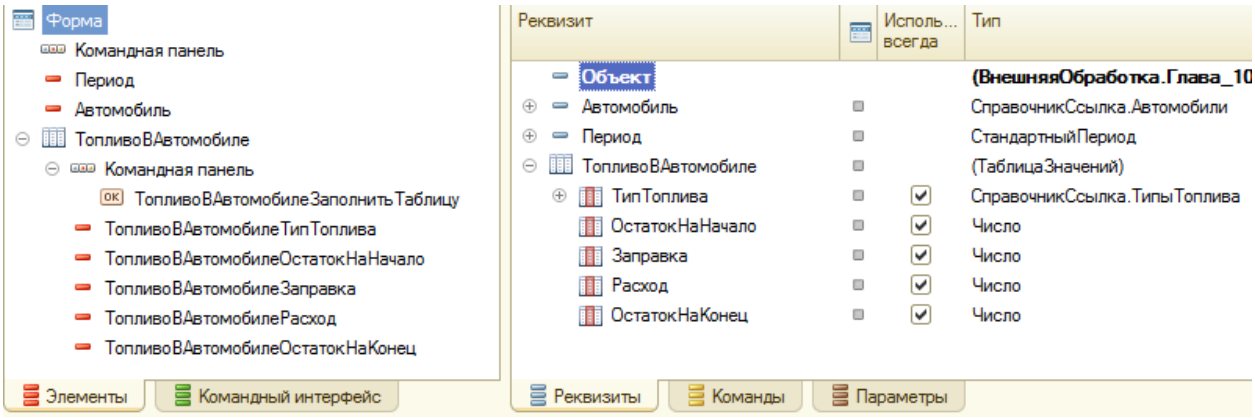


Рис. 10.4.38

В серверном обработчике команды создадим объект *Запрос* и зайдем в конструктор запроса. В конструкторе раскроем дерево регистров накопления. Обратите внимание на таблицу *Остатки и обороты Топливо в автомобиле*.

Рис. 10.4.39

Выберем эту таблицу и установим параметры для нее: начало периода, конец периода и условие отбора по автомобилю. А саму таблицу назовем «ДвижениеТопливаАвто».

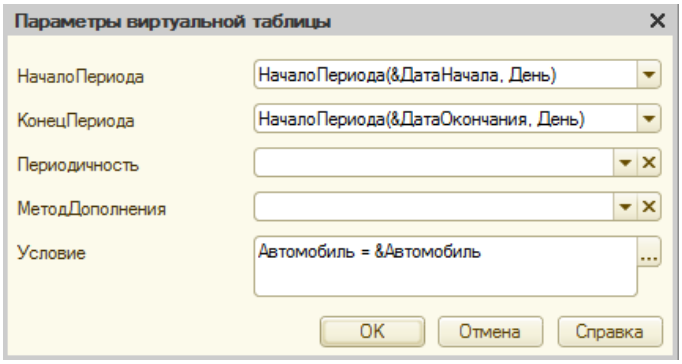


Рис. 10.4.40

Выбираем поля: *ТипТоплива*, *Количество начальный остаток, Количество приход, Количество расход* и *Количество конечный остаток*. Дадим им псевдонимы по аналогии с названием колонок в таблице значений на форме.

Должен получиться следующий запрос:

```
ВЫБРАТЬ
    ДвижениеТопливаАвто.ТипТоплива КАК ТипТоплива,
    ДвижениеТопливаАвто.КоличествоНачальныйОстаток КАК ОстатокНаНачало,
    ДвижениеТопливаАвто.КоличествоПриход КАК Заправка,
    ДвижениеТопливаАвто.КоличествоРасход КАК Расход,
    ДвижениеТопливаАвто.КоличествоКонечныйОстаток КАК ОстатокНаКонец
ИЗ
    РегистрНакопления.ТопливоВАвтомобилях.ОстаткиИОбороты(НАЧАЛОПЕРИОДА(&ДатаНачала, ДЕНЬ),
                                                          НАЧАЛОПЕРИОДА(&ДатаОкончания, ДЕНЬ), ,
,
                                                          Автомобиль = &Автомобиль) КАК
ДвижениеТопливаАвто
```

Листинг 10.4.12

Осталось заполнить параметры и вывести результат в таблицу значений.

```
Запрос.УстановитьПараметр("ДатаНачала",Период.ДатаНачала);
Запрос.УстановитьПараметр("ДатаОкончания",Период.ДатаОкончания);
Запрос.УстановитьПараметр("Автомобиль",Автомобиль);
Выборка = Запрос.Выполнить().Выбрать();
Пока Выборка.Следующий() Цикл
    НовСтр = ТопливоВАвтомобиле.Добавить();
    ЗаполнитьЗначенияСвойств(НовСтр,Выборка);
КонецЦикла;
```

Листинг 10.4.13

Сохраните обработку, запустите и посмотрите, что получилось.

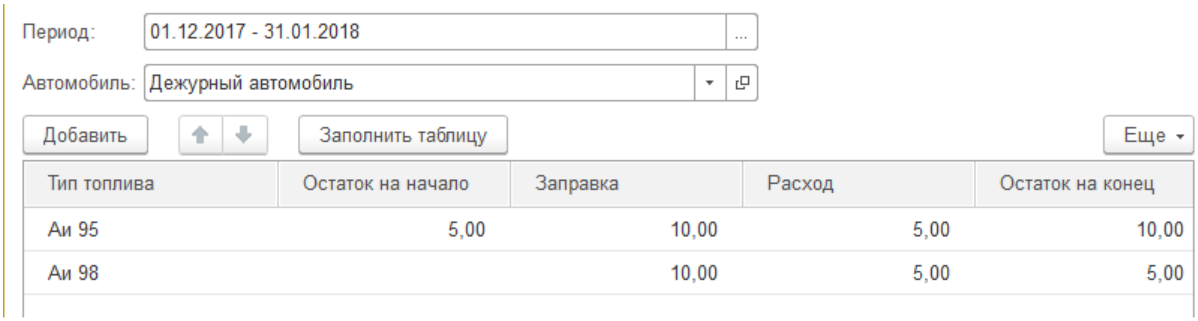


Рис. 10.4.41

Как видите, вышло количество топлива в автомобиле на начало периода, на конец периода и сколько было топлива в течение периода заправлено, а сколько израсходовано.

А теперь попробуем все посмотреть в разрезе документов. Для этого добавьте колонку *Документ* в таблицу значений (составной тип документы *Заправка топлива* и *Отчет о расходе топлива*).

Зайдем обратно в конструктор запроса и в параметр *Периодичность* установим значение *Регистратор*.

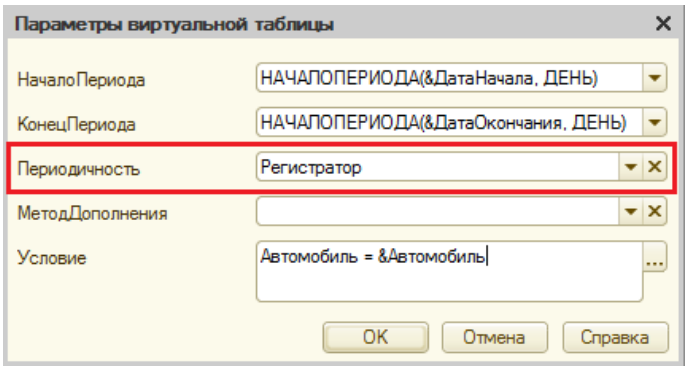


Рис. 10.4.42

Теперь в полях таблицы появилось поле *Регистратор*.

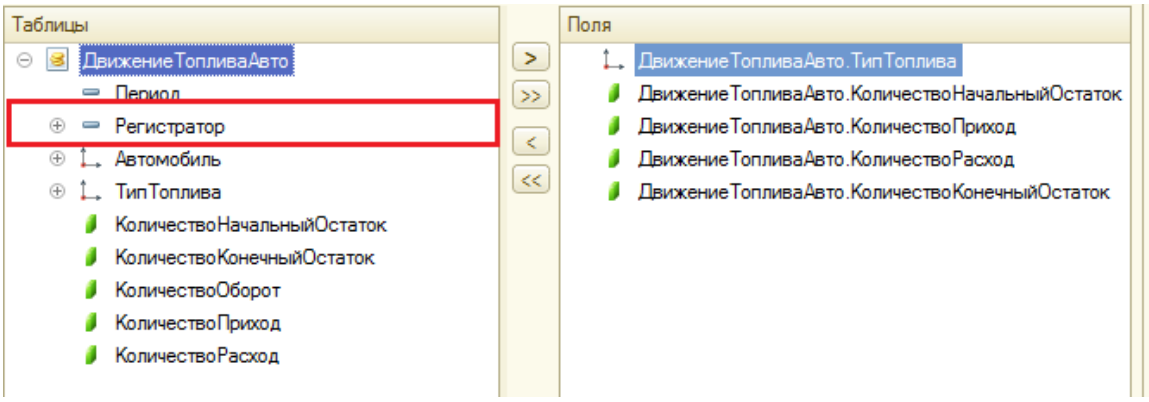


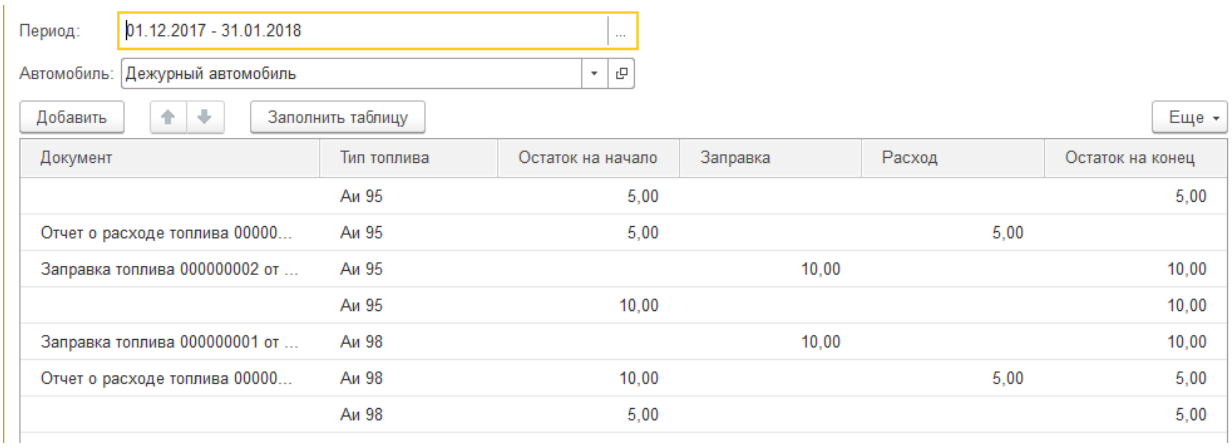
Рис. 10.4.43

Выберем это поле и присвоим ему псевдоним *Документ*. Должен получиться вот такой запрос:

```
ВЫБРАТЬ
    ДвижениеТопливаАвто.ТипТоплива КАК ТипТоплива,
    ДвижениеТопливаАвто.КоличествоНачальныйОстаток КАК ОстатокНаНачало,
    ДвижениеТопливаАвто.КоличествоПриход КАК Заправка,
    ДвижениеТопливаАвто.КоличествоРасход КАК Расход,
    ДвижениеТопливаАвто.КоличествоКонечныйОстаток КАК ОстатокНаКонец,
    ДвижениеТопливаАвто.Регистратор КАК Документ
ИЗ
    РегистрНакопления.ТопливоВАвтомобилях.ОстаткиИОбороты(НАЧАЛОПЕРИОДА(&ДатаНачала, ДЕНЬ),
                                                          НАЧАЛОПЕРИОДА(&ДатаОкончания, ДЕНЬ),
                                                          Регистратор, ,
                                                          Автомобиль = &Автомобиль) КАК
ДвижениеТопливаАвто
```

Листинг 10.4.14

Сохраните запрос, обработку и посмотрите, как она работает.



Документ	Тип топлива	Остаток на начало	Заправка	Расход	Остаток на конец
	Аи 95	5,00			5,00
Отчет о расходе топлива 00000...	Аи 95	5,00		5,00	
Заправка топлива 000000002 от ...	Аи 95		10,00		10,00
	Аи 95	10,00			10,00
Заправка топлива 000000001 от ...	Аи 98		10,00		10,00
Отчет о расходе топлива 00000...	Аи 98	10,00		5,00	5,00
	Аи 98	5,00			5,00

Рис. 10.4.44

Как Вы видите, запрос вывел нужную нам информацию, но, к сожалению, появились ненужные строки конца и начала периода, без регистратора. Они не очень информативны и мешают воспринимать информацию. Для этого нам необходимо вернуться обратно в *Параметры виртуальной таблицы*. И установить *МетодДополнения* в *Движения*.

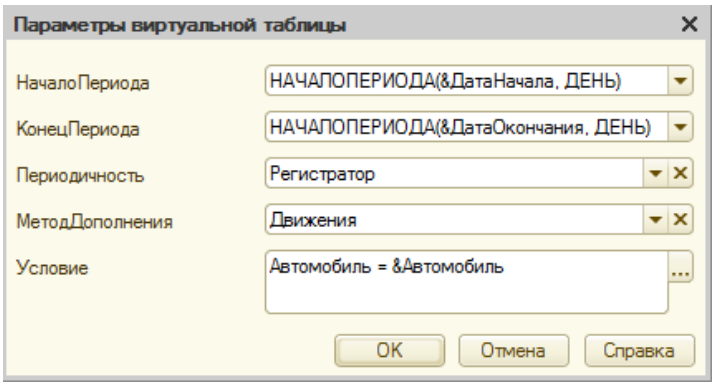
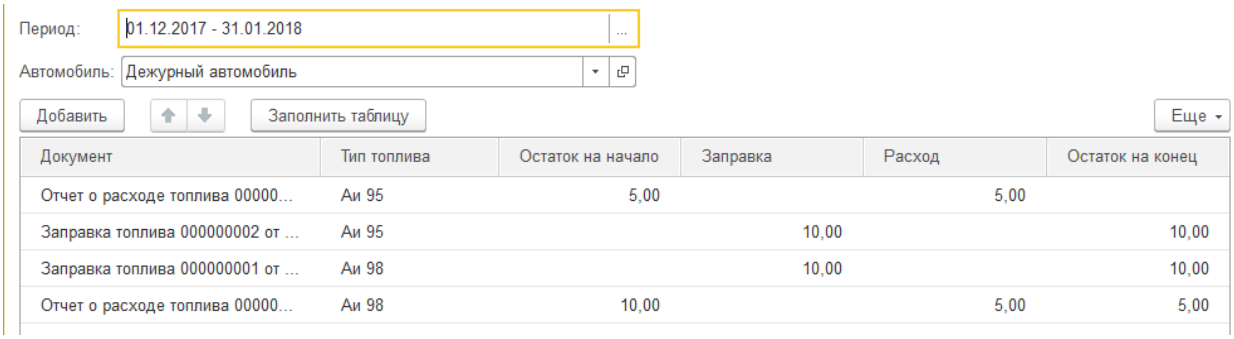


Рис. 10.4.45

Сохраните запрос, обработку и посмотрите, что получилось.



Документ	Тип топлива	Остаток на начало	Заправка	Расход	Остаток на конец
Отчет о расходе топлива 00000...	Аи 95	5,00		5,00	
Заправка топлива 000000002 от ...	Аи 95		10,00		10,00
Заправка топлива 000000001 от ...	Аи 98		10,00		10,00
Отчет о расходе топлива 00000...	Аи 98	10,00		5,00	5,00

Рис. 10.4.46

Мы видим, что границы периода убрались и информация предоставляется в удобном для понимания виде.

Резюме

На этом мы закончим изучать способы получения информации из регистров накопления, как Вы поняли, их немало. Для регистров накопления действует то же правило: если необходима простая информация, то удобнее использовать методы менеджера регистров, а если необходимо получить более сложную периодическую информацию, то лучше использовать запросы. У языка запроса 1С гораздо больше возможностей, чем у методов менеджеров регистра. В этой главе мы разобрали только общие моменты использования виртуальных таблиц регистров сведений и накопления. Больше практической информации по работе с виртуальными таблицами регистров дается в моем видео-курсе «Запросы в 1С для начинающих», у Вас, как у покупателя этой книги, есть промо-код на скидку в 25 % - hrW0rl9Nnx.

Глава 11. Вывод информации

Одиннадцатую, последнюю главу этой книги мы посвятим различным способам вывода информации пользователю. Вы научитесь создавать и заполнять печатные формы документов, выводить информацию о деятельности предприятия с помощью простых отчетов и СКД. А также узнаем, что такое динамический список и как с ним работать.

Начнем мы с печатных форм.

Часть 1. Печатные формы

Практически всегда в прикладных задачах есть необходимость привязать к определенному документу *Печатную форму*. Форма может представлять из себя некоторый бланк, который пользователь будет распечатывать после записи или проведения документа. В данной части мы научимся создавать примитивные печатные формы.

Макет

Вывод любой печатной формы выполняется с помощью макетов. Всего существует девять типов макетов, но мы с Вами в этой книге рассмотрим два – табличный документ и макет системы компоновки данных (СКД).

Сейчас мы создадим какой-нибудь макет к документу *Прибытие в гараж* нашей конфигурации тип *Табличный документ*. Макеты СКД мы будем проходить во второй части этой главы.

Раскройте в дереве конфигурации документ *Прибытие в гараж* и обратите внимание на элемент *Макеты*.

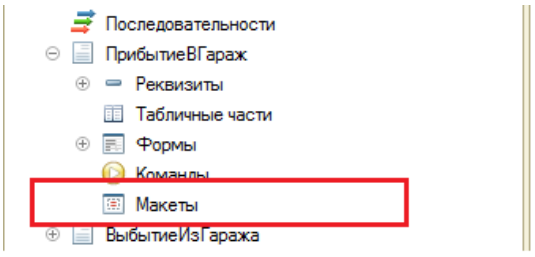


Рис. 11.1.1

Кликните правой кнопкой мышки и в открывшемся контекстном меню выберите пункт *Добавить.*

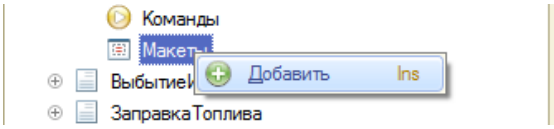


Рис. 11.1.2

Открывается конструктор макетов. В данном конструкторе мы ничего не меняем, но обращаем внимание, что тип макета выбран *Табличный документ*.

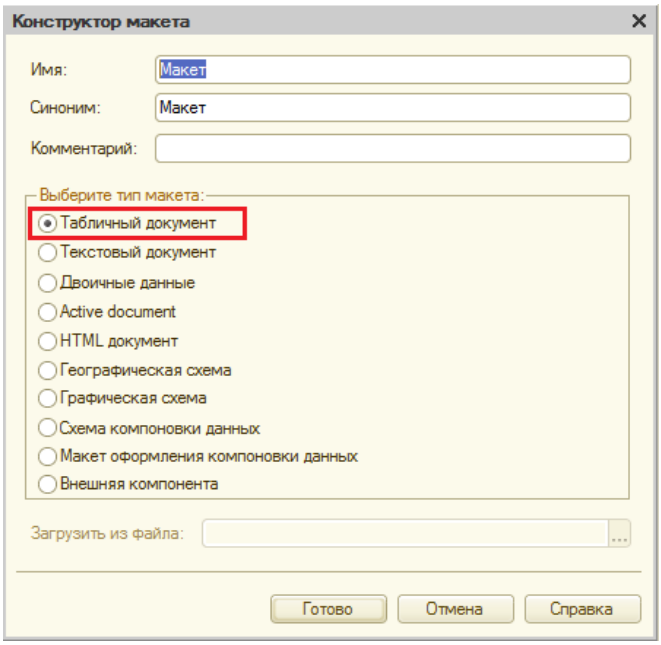


Рис. 11.1.3

Нажмите кнопку *Готово*, и мы создали макет под названием *Макет*.

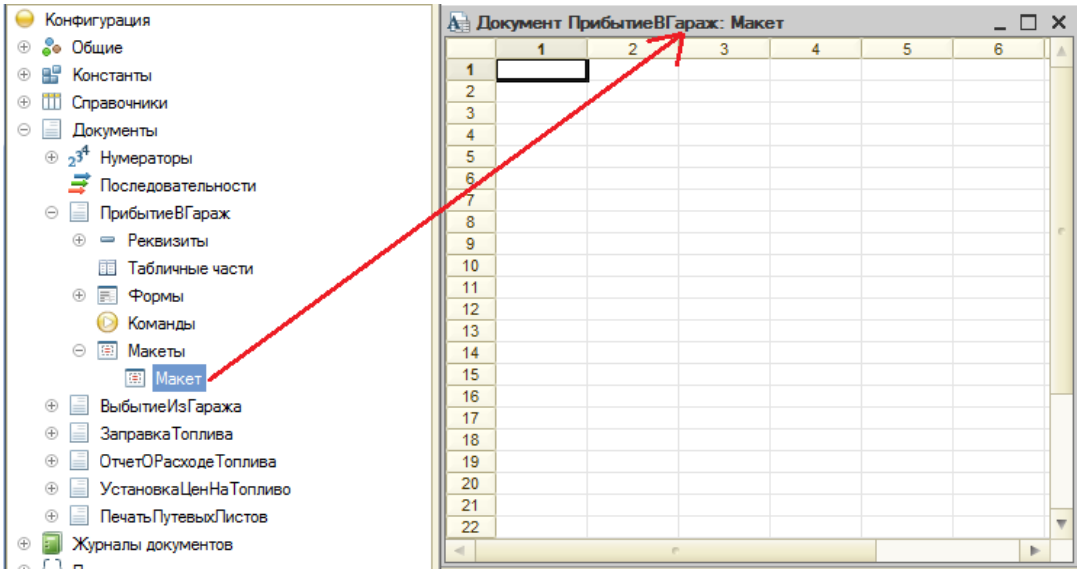


Рис. 11.1.4

Как видите, макет табличного документа представляет собой форму наподобие таблиц Excel. Она также состоит из строк, столбцов и ячеек. Каждая ячейка имеет свои уникальные свойства (кликните правой кнопкой мышки и в выпадающем меню выберите пункт *Свойства*).

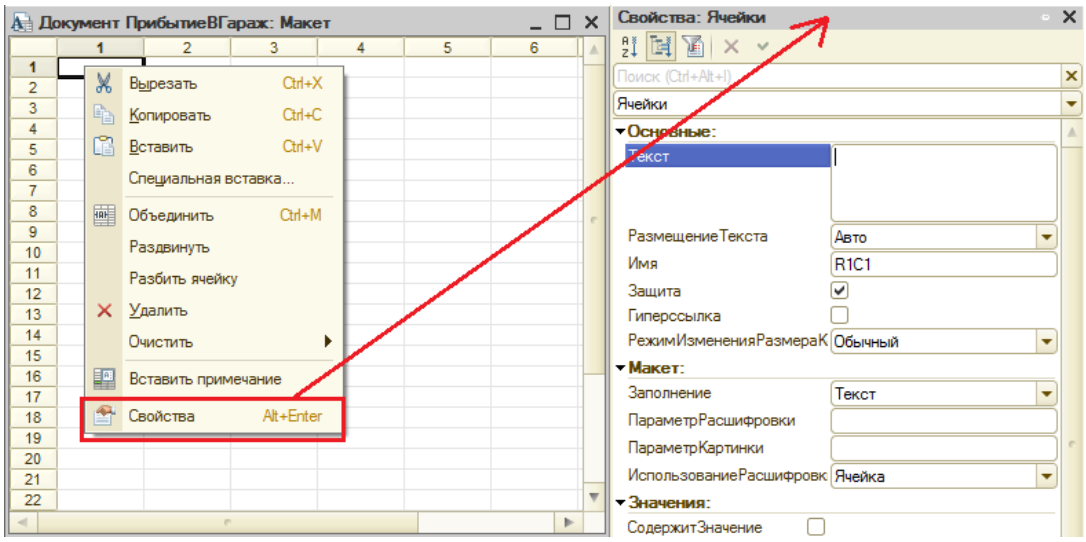


Рис. 11.1.5

Внутрь любой ячейки можно поместить текст, поместим в ячейку второй колонки второго столбца текст: «Прибытие автомобиля №».

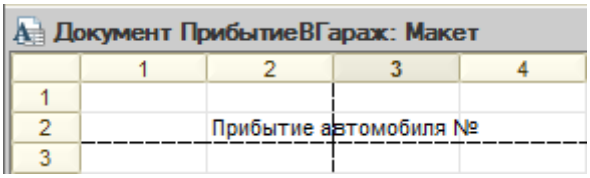


Рис. 11.1.6

Вы видите, что наш текст не уместился на одной ячейке, а сразу перешел на вторую, и появились две линии по краям ячейки – горизонтальная и вертикальная. Эти линии ограничивают область, которая будет выводиться на экран.

Понятно что, строка *«Прибытие автомобиля…»* будет выведена не полностью. Чтобы такого не произошло, объединим ячейки, и увеличим шрифт. Для этого необходимо их выделить (нажать левую кнопку мышки и вести по ячейкам курсор с нажатой кнопкой).

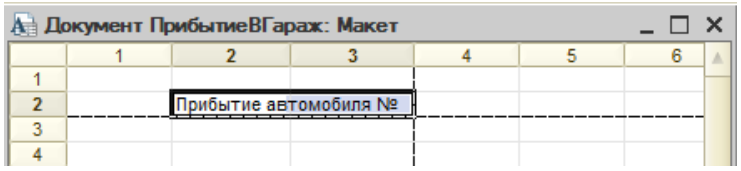


Рис. 11.1.7

После перейдите в «*Главное меню*» - *«Таблица»* - *«Ячейки»* и нажмите на кнопку *Объединить.*

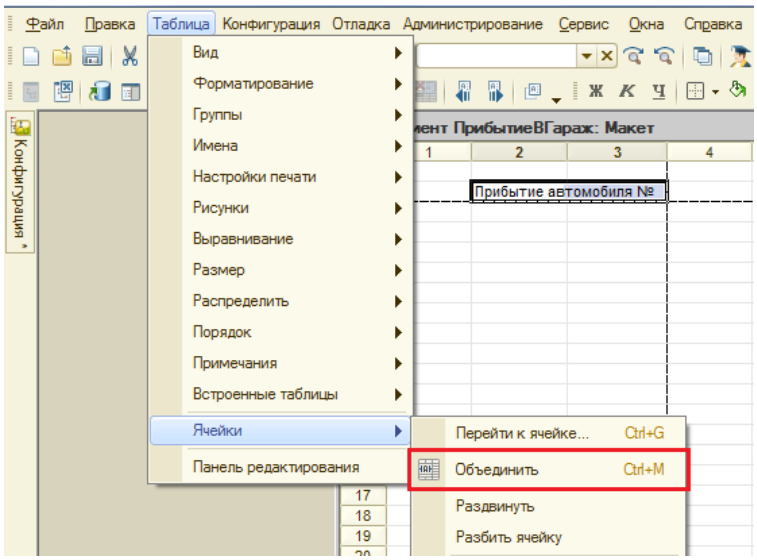


Рис. 11.1.8

Видите: ячейка стала одна.

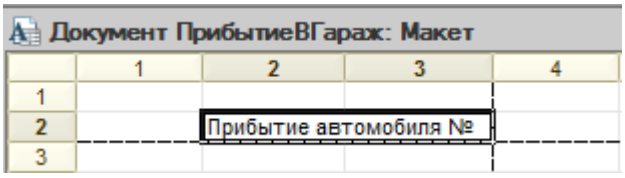


Рис. 11.1.9

Измените величину шрифта на *14* (ячейка должна быть выделена).

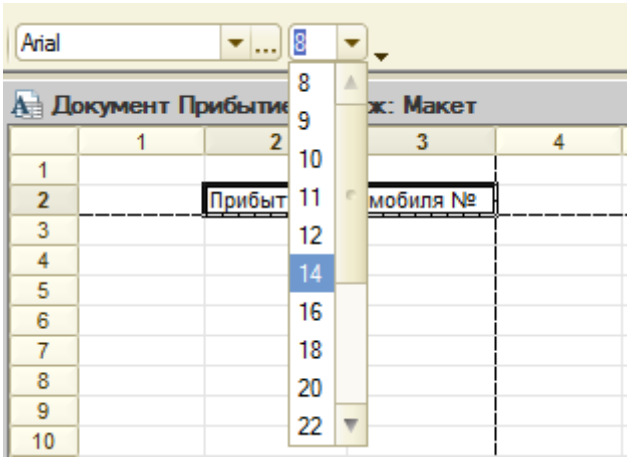


Рис. 11.1.10

Ячейка увеличилась, и текст опять не помещается. Снова объедините ячейки с двумя соседними. Для этого нужно снять имеющееся объединение, также выделив ячейки и нажав на кнопку *Объединить* в меню *«Главное меню» - «Таблица» - «Ячейки»*. И заново объединить в этот раз четыре ячейки.

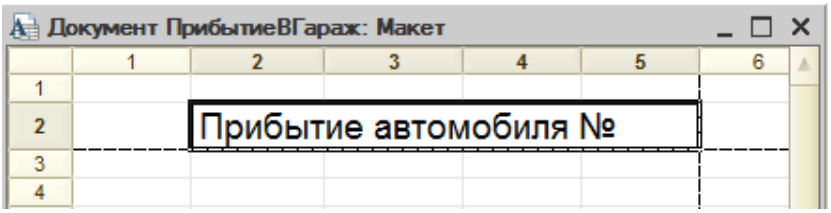


Рис. 11.1.11

Помните: все, что Вы напишете в данной таблице, будет, при определенных условиях, выведено в печатную форму.

Теперь нам необходимо передать номер и дату документа прибытия в макет.

Как это сделать? Делается это достаточно просто: необходимо изменить свойство *Заполнение ячейки*.

Внесите текст в следующую ячейку после объединенной (сузьте крайнюю объединенную колонку) и измените шифр на 14.

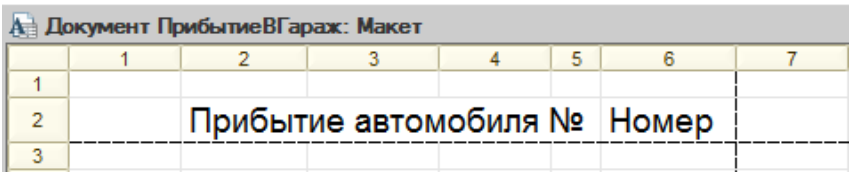


Рис. 11.1.12

Далее Вам нужно сделать так, чтобы в ячейку можно было передавать номер документа (как это делается, мы изучим немного позже). А пока настроим ячейку на такую возможность. Для этого откройте свойства ячейки: выделите ее и кликните правой кнопкой мышки, выйдет контекстное меню, где выберите пункт *Свойства*.

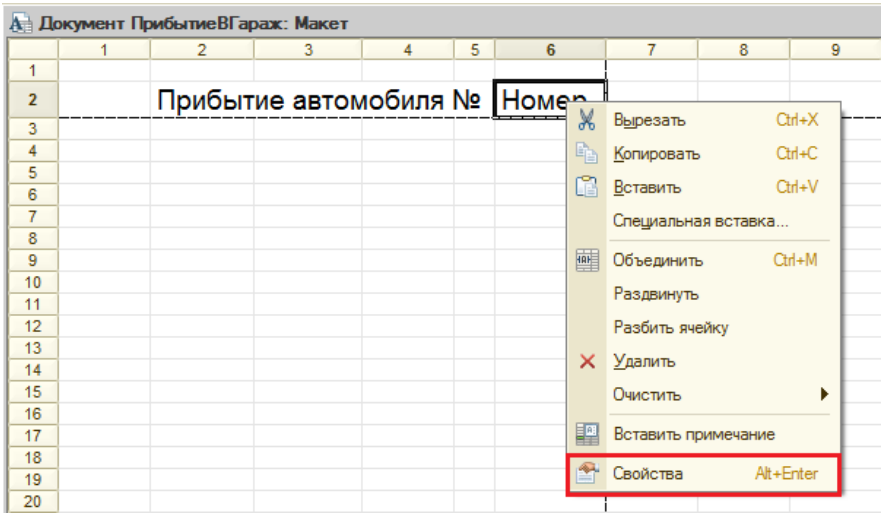


Рис. 11.1.13

Откроется палитра свойств этой ячейки, и в них обратите внимание на свойство *Заполнение.*

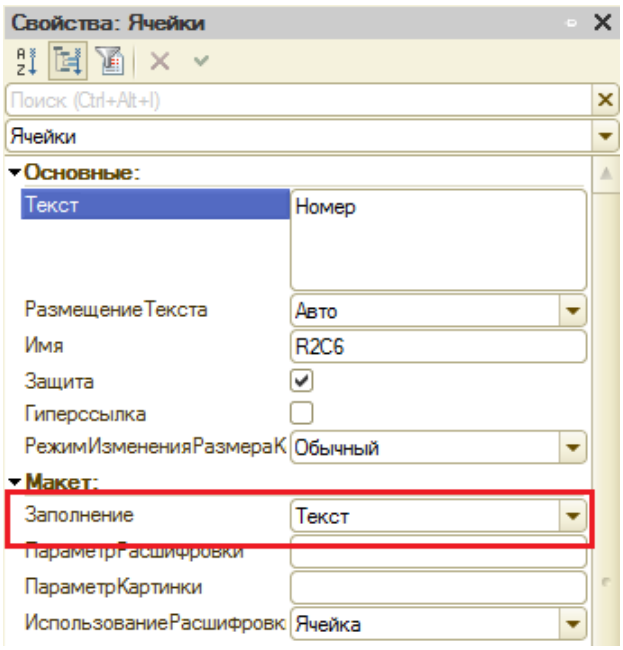


Рис. 11.1.14

Измените его значение в *Параметр.*

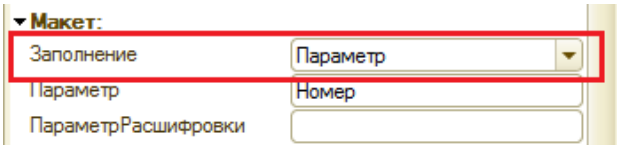


Рис. 11.1.15

После этого, обратите внимание, слово *Номер* закрылось треугольными скобками.

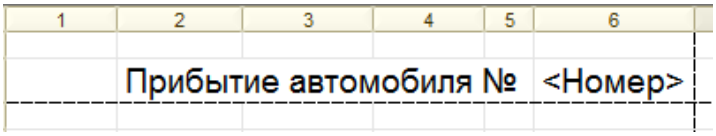


Рис. 11.1.16

Добавим еще и дату. Ячейке со словом *Дата* также поставьте свойство

Заполнение в значение *Параметр*.

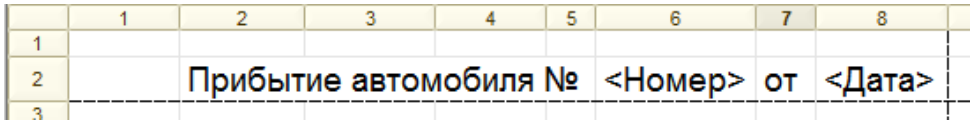


Рис. 11.1.17

Сразу же установите формат даты в удобный для восприятия, чтобы потом не возвращаться к этому в коде.

Для этого зайдите также в палитру свойств ячейки *Дата* и перейдите в свойство *Формат*.

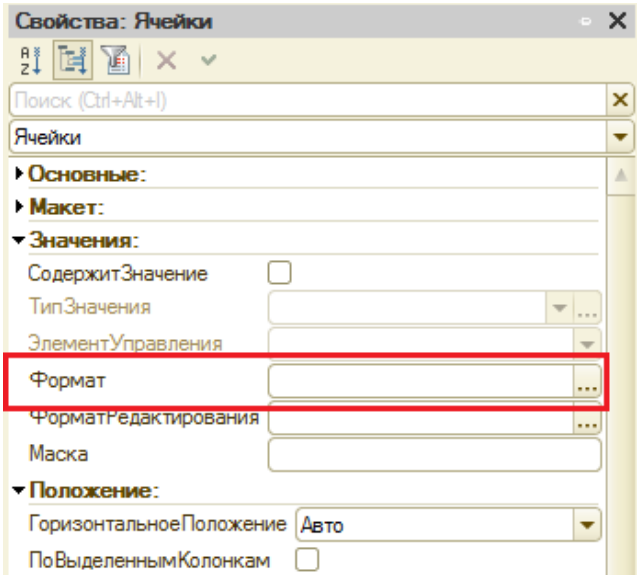


Рис. 11.1.18

Установим нужный формат даты.

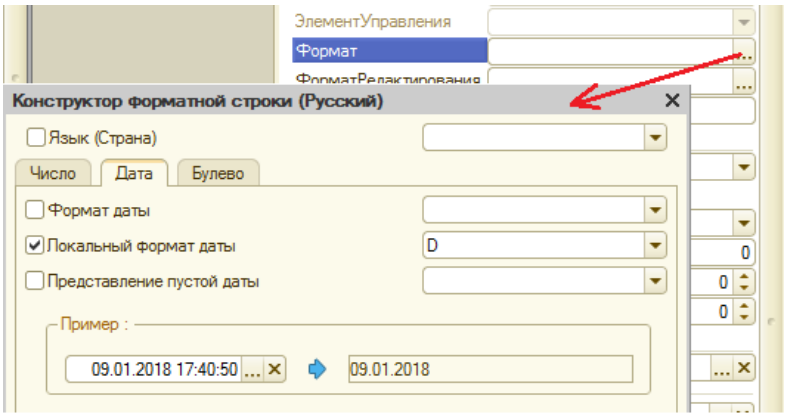


Рис. 11.1.19

Теперь у многих из вас может возникнуть вопрос: а можно ли передавать параметр внутрь текста? Да, можно, для этого необходимо устанавливать свойство *Заполнение* в значение *Шаблон*.

Напишите через строку следующий текст: «Автомобиль [Автомобиль] прибыл в [Гараж] [ДатаПрибытия]». Поставьте *12* шрифт и объедините ячейки.

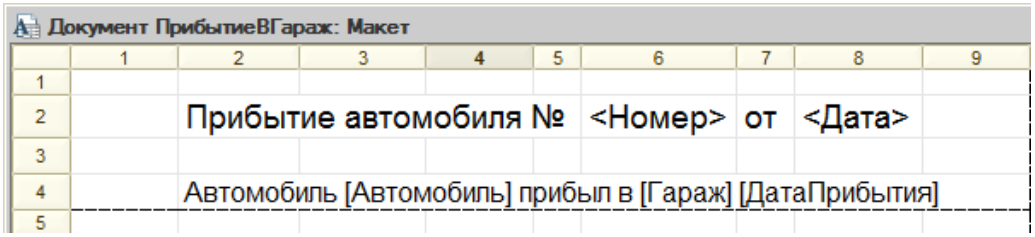


Рис. 11.1.20

Перейдем в палитру свойств объединенной ячейки и установим в свойство *Заполнение* значение *Шаблон*.

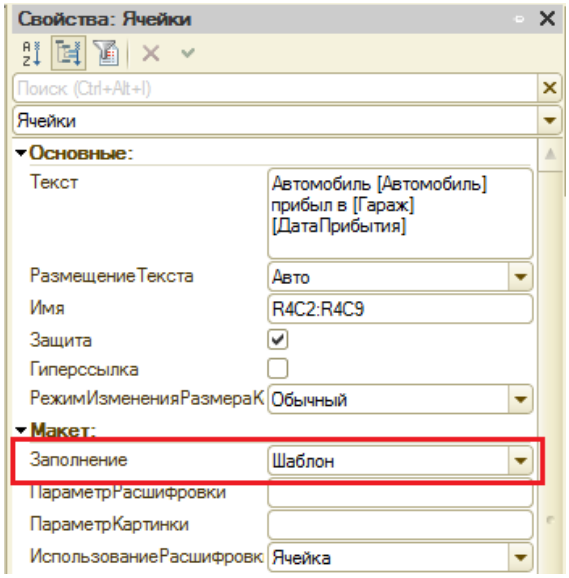


Рис. 11.1.21

Все слова, которые находятся в квадратных скобках, будут передаваться в виде параметров.

Для ячеек можно устанавливать границы в виде различных линий. Делается это с помощью свойств ячейки *Граница слева, Граница сверху, Граница справа, Граница снизу*.

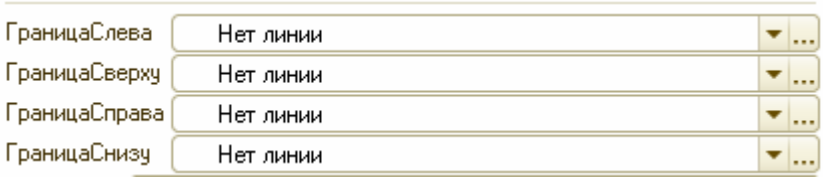


Рис. 11.1.22

Поставим на несколько нижних ячеек границы снизу, а также напишем текст под ними:

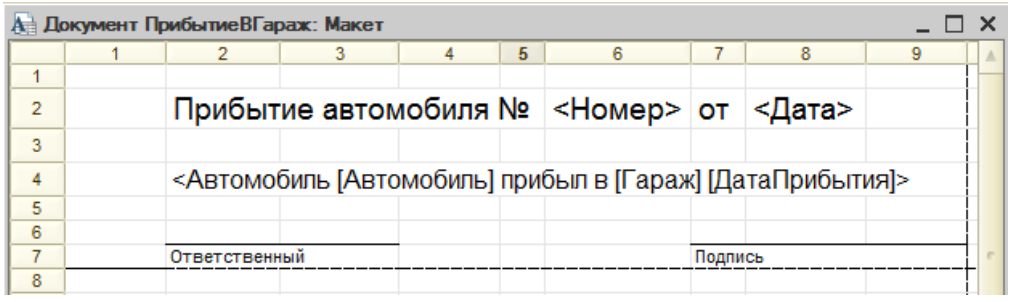


Рис. 11.1.23

Текст «Ответственный» и «Подпись» сдвинут влево. Сделаем этот текст по центру ячейки. Для этого выделим объединенную ячейку и в свойство *ГоризонтальноеПоложение* установим значение *Центр*.

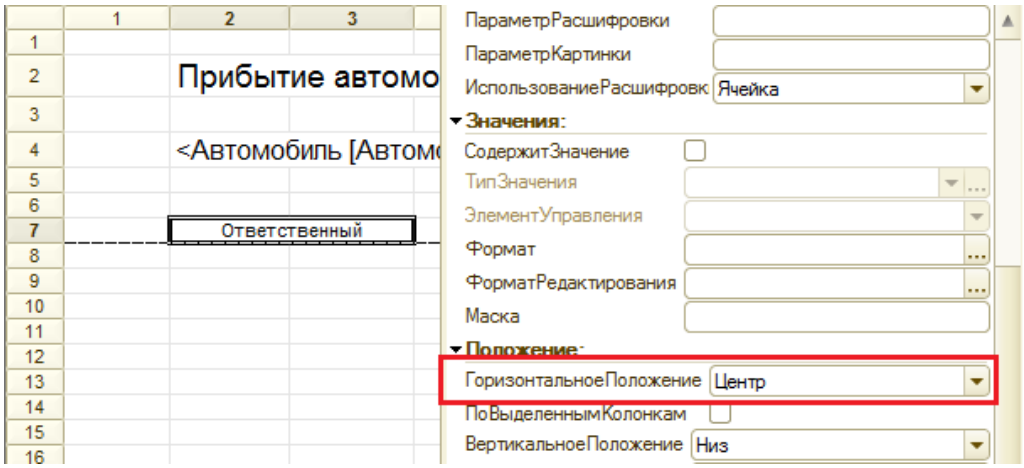


Рис. 11.1.24

В целом макет готов, но это еще не все. Для того, чтобы было удобнее выводить макет, принято использовать области. Зададим одну область и назовем ее *Основная*.

Для этого выделите те ячейки, которые мы хотим поместить в область (мы выделим все строки в границе печати).

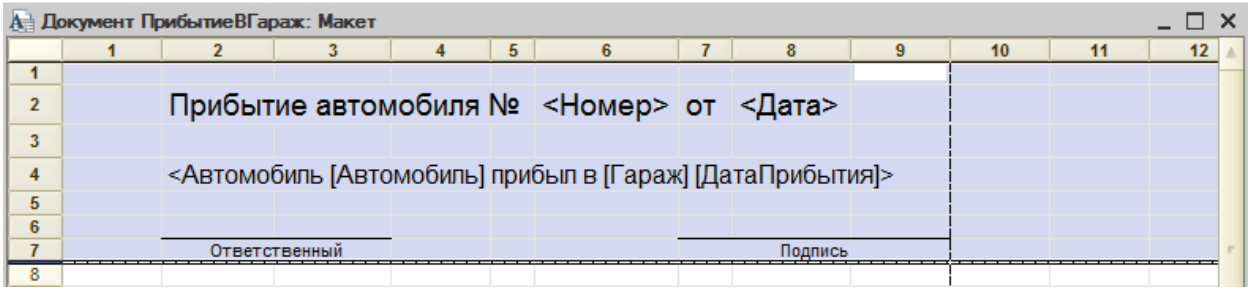


Рис. 11.1.25

После этого перейдите в меню (область должна оставаться выделенной) *«Главное меню» - «Таблица» – «Имена» – «Назначить имя»*.

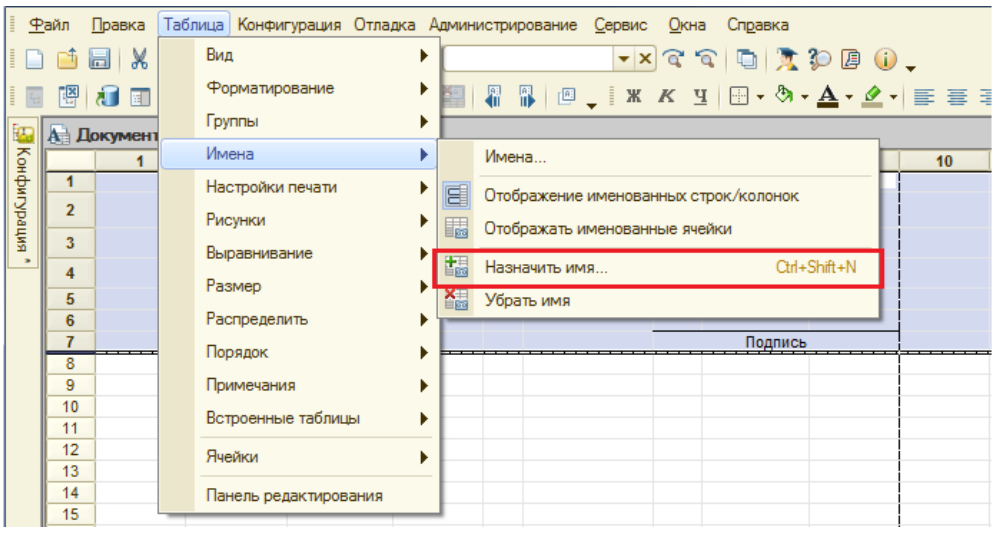


Рис. 11.1.26

Создайте область и назовите ее *Основная*.

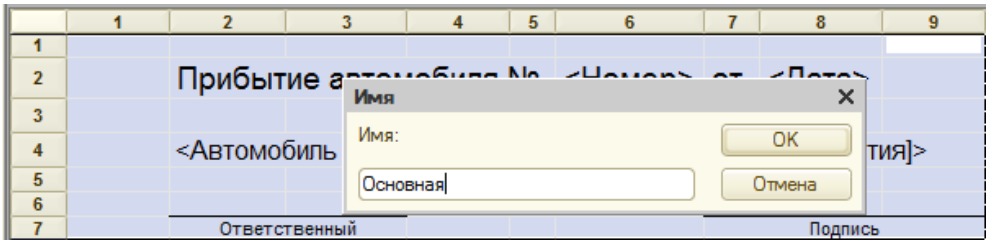


Рис. 11.1.27

После назначения области макет должен приобрести следующий вид:

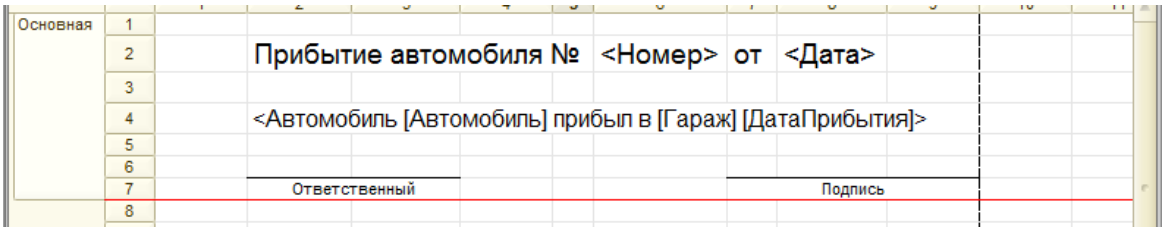


Рис. 11.1.28

Все, мы создали самый примитивнейший макет. Теперь необходимо вывести эту информацию на экран при выполнении команды *Печать*, которую мы создали в 5 главе (см. стр. 322). Только процедуру «Печать» в *модуле объекта* документа *Прибытие в гараж* переделаем в функцию и напишем в ней следующий код вместо того, что был ранее:

```
Функция Печать() Экспорт
    Запрос = Новый Запрос;
    Запрос.Текст = "ВЫБРАТЬ
                   |    ПрибытиеВГараж.Номер КАК Номер,
                   |    ПрибытиеВГараж.Дата КАК Дата,
                   |    ПрибытиеВГараж.Автомобиль КАК Автомобиль,
                   |    ПрибытиеВГараж.Гараж КАК Гараж,
                   |    ПрибытиеВГараж.ДатаПрибытия КАК ДатаПрибытия
                   |ИЗ
                   |    Документ.ПрибытиеВГараж КАК ПрибытиеВГараж
                   |ГДЕ
                   |    ПрибытиеВГараж.Ссылка = &Ссылка";
    Запрос.УстановитьПараметр("Ссылка",Ссылка);
    Выборка = Запрос.Выполнить().Выбрать();
    Выборка.Следующий();

    Макет = ПолучитьМакет("Макет");
    ОбластьОсновная = Макет.ПолучитьОбласть("Основная");
    ОбластьОсновная.Параметры.Номер         = Выборка.Номер;
    ОбластьОсновная.Параметры.Дата          = Выборка.Дата;
    ОбластьОсновная.Параметры.Автомобиль    = Выборка.Автомобиль;
    ОбластьОсновная.Параметры.Гараж         = Выборка.Гараж;
    ОбластьОсновная.Параметры.ДатаПрибытия = Выборка.ДатаПрибытия;

    ТабДокумент = Новый ТабличныйДокумент;
    ТабДокумент.Вывести(ОбластьОсновная);
    Возврат ТабДокумент;
КонецФункции
```

Листинг 11.1.1

Разберем данный код.

Первым делом я при помощи запроса получаю информацию по реквизитам объекта. Конечно, Вы спросите, почему я сделал так, а не обратился напрямую к этим реквизитам. Сделано это в целях повышения производительности. Потому что при выполнении запроса мы только один раз обращаемся к базе данных, а после получаем данные уже из запроса, которые хранятся в объекте выбора. Иначе мы каждый раз будем обращаться к базе данных, чтобы получить значение того или иного реквизита.

После запроса и выборки мы получаем макет с помощью метода объекта документа *ПолучитьМакет*. В качестве параметра мы передали название макета, которое указано в конфигураторе.

В следующей строке мы получаем область, которую задали в макете, указав ее название. Обращаю Ваше внимание, что можно и не задавать имя области, а просто указать диапазон ячеек. Но на начальном этапе, да и в дальнейшем, лучше все делать через имена областей, т.к. это гораздо более удобно для доработок и чтения кода.

Теперь нам нужно заполнить параметры области, которые мы задали в макете. Делается это с помощью свойства области *Параметры*.

Данное свойство представляет объект *Параметры макета табличного документа*, который является коллекцией параметров макета.

К параметрам макета можно обращаться через квадратные скобки, а проще напрямую, как в данном примере.

Мы заполнили макет и параметры его области. Но сам по себе макет бесполезен. Его необходимо вывести. Делается это с помощью объекта *Табличный документ*. Данный объект предназначен для вывода табличных форм на экран. Он создается с помощью оператора *Новый*.

С помощью метода *Вывести* табличного документа добавляем основную область макета в табличный документ. Прежде чем вывести сам *табличный документ*, его необходимо скомпоновать из имеющихся в макете областей. Дело в том, что областей может быть несколько, некоторые из них должны будут повторяться, а некоторые, наоборот, отсутствовать при определенных условиях. Поэтому с помощью данного метода *табличного документа* и формируется результирующий документ.

Мы не можем сейчас вывести табличный документ на экран, поскольку код в модуле объекта выполняется в серверном контексте. А табличный документ должен показываться на компьютере пользователя, т.е. в клиентском контексте. Поэтому мы возвращаем созданный и заполненный табличный документ, чтобы его использовать в дальнейшем.

Переходим в модуль формы и переделываем уже имеющийся обработчик команды «Печать».

```
&НаСервере
Функция  ПечатьНаСервере()
    ОбъектДок = РеквизитФормыВЗначение("Объект");
    ТабДок = ОбъектДок.Печать();
    Возврат ТабДок;
КонецФункции

&НаКлиенте
Процедура Печать(Команда)
    ТабДок = ПечатьНаСервере();
    ТабДок.Показать();
КонецПроцедуры
```

Листинг 11.1.2

В данном коде мы в серверном контексте получаем табличный документ при помощи функции модуля объекта *Печать*. Сам объект мы получили при помощи уже знакомого нам метода управляемой формы *РеквизитФормыВЗначение*. Полученный табличный документ мы передали с серверного контекста формы в клиентский, для этого нам пришлось процедуру *ПечатьНаСервере* переделать в функцию.

После этого мы с помощью метода *Показать* выводим его на экран.

Сохраните конфигурацию, и посмотрим, как выходит наша печатная форма.

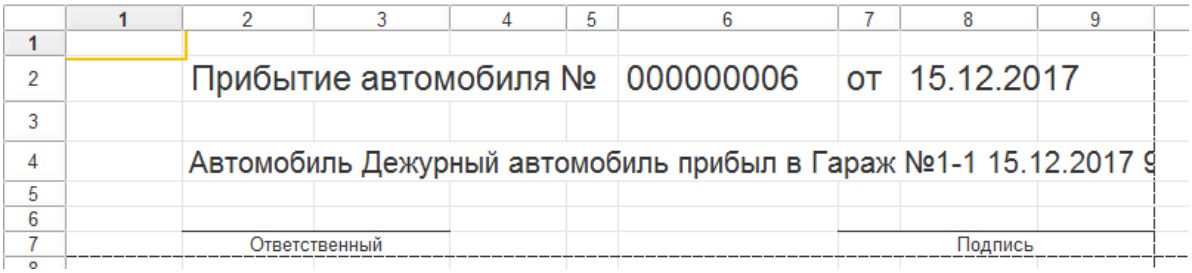


Рис. 11.1.29

Дата прибытия во второй строке печатной формы выходит не очень красиво. Сделайте вывод даты прибытия в более удобном формате.

Данный метод вывода на печать является более хрестоматийным, чем практическим. Так сделано, чтобы Вам было проще понять работу с табличным документом. Более практично печатные формы выводить, используя команды объекта. Более подробно работа с командами объекта дается в книге «Основы разработки в 1С: Такси».

Макет с несколькими областями

Мы только что научились выводить макет с одной областью на печать. В реальности такие ситуации достаточно редки. Часто документы содержат табличную часть, которую тоже необходимо вывести в макет.

Сейчас мы создадим печатную форму документа *Установка цен на топливо*, в которой перечислим все виды топлива, которые указаны в табличной части.

Раскройте нужный документ в дереве конфигурации и создайте пустой макет с типом *Табличный документ*.

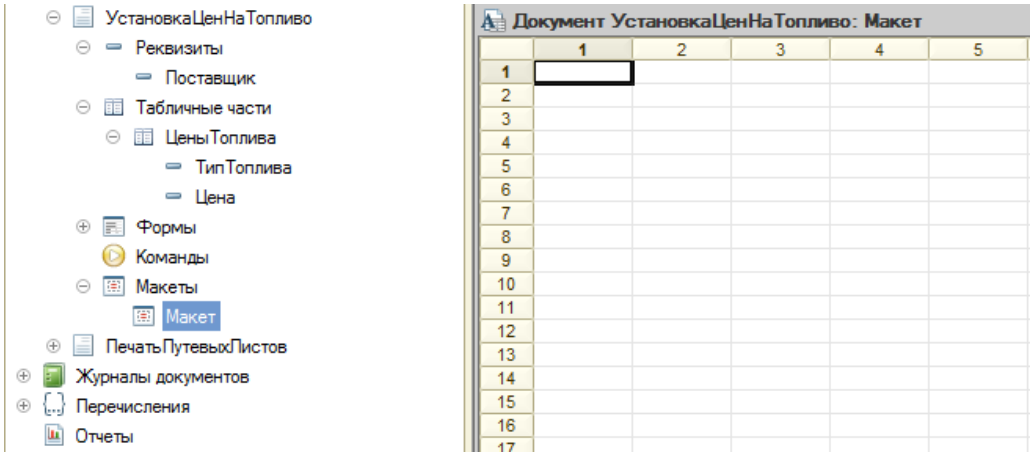


Рис. 11.1.30

Создайте новую область с 5-ю строками у нового макета и назовите ее *Шапка* (как создавать новые области, см. стр. 674).

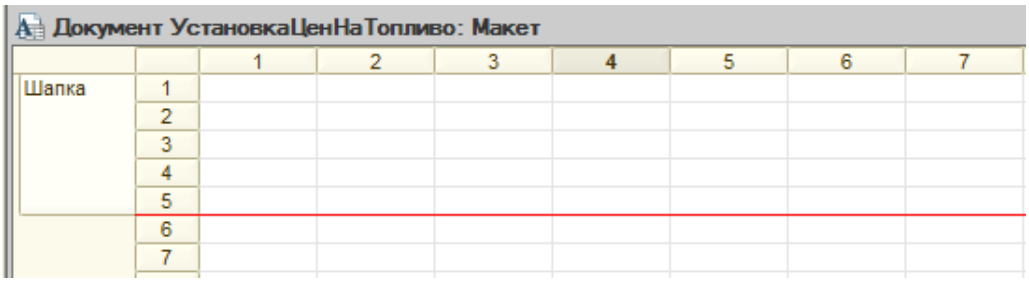


Рис. 11.1.31

Напишите во второй ячейке второй строки следующий текст:

Цены на топливо поставщика [Поставщик] от [Дата]

Увеличьте шрифт, установите значение свойства *Заполнение* в *Шаблон*.

И объедините ячейки по всей длине строки.

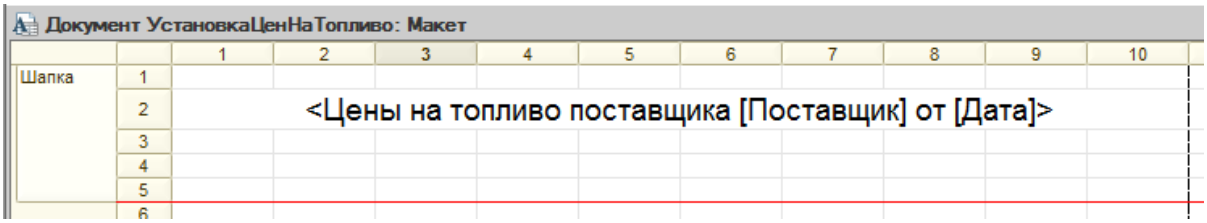


Рис. 11.1.32

Все. Шапку документа мы создали. Теперь необходимо создать шапку табличной части. Для этого создайте новую область, которую так и назовите *ШапкаТаблицы.*

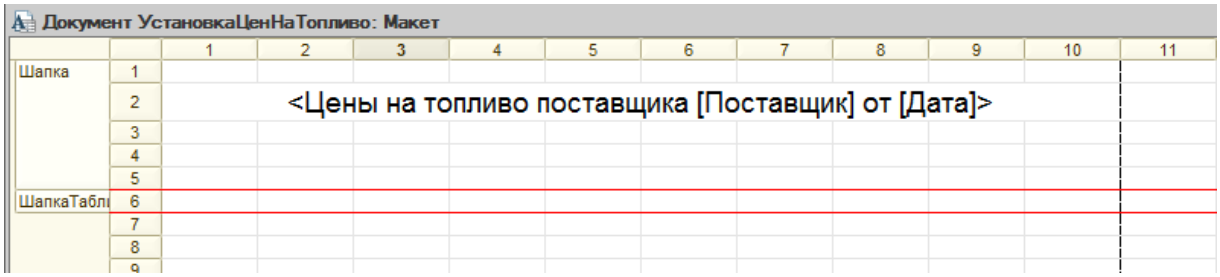


Рис. 11.1.33

В первой ячейке напишите *«№»* и обведите ее границей.

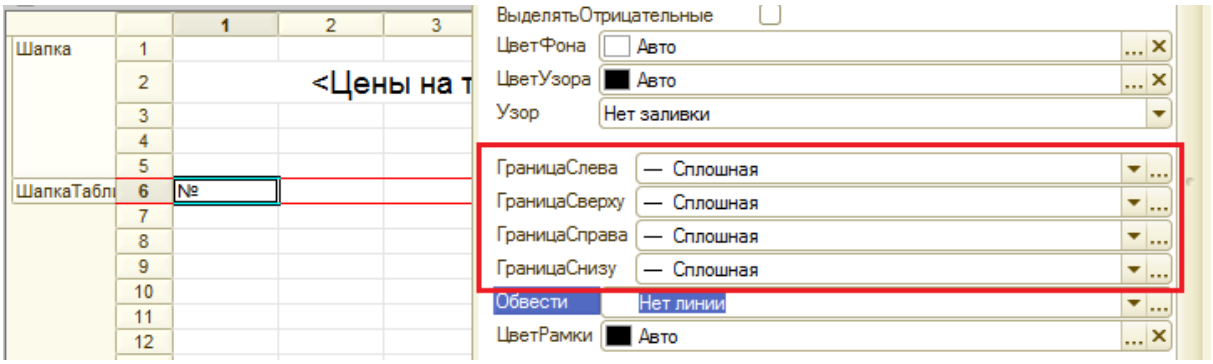


Рис. 11.1.34

Последующие пять ячеек объедините в одну, назовите их *Тип топлива* и обведите объединенную ячейку границами.

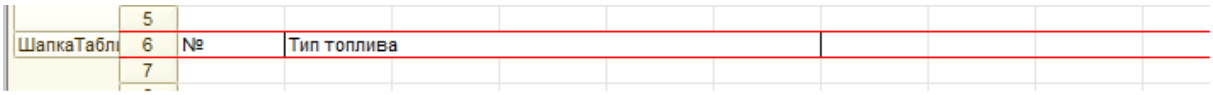


Рис. 11.1.35

Последнюю ячейку назовите «Цена» и тоже обведите границей. Для всех ячеек увеличьте шрифт до 12, и *горизонтальное положение* для всех установим *Центр*.

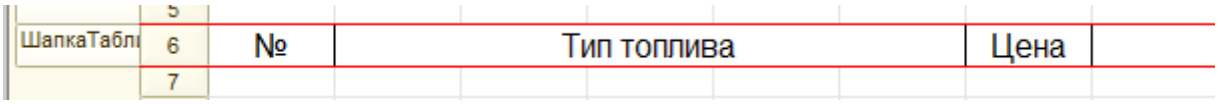


Рис. 11.1.36

Подсказка: если какое-то свойство распространяется на несколько ячеек, то можно выделить их все и установить нужное свойство. Оно установится для всех выделенных ячеек.

Теперь необходима та область, в которой будут выводиться непосредственно цены на топливо. Для этого создадим новую область и назовем ее *Таблица*.

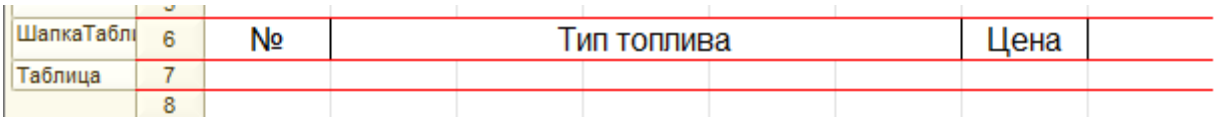


Рис. 11.1.37

В первой ячейке напишем *Номер*, сделаем эту ячейку параметром (свойство *Заполнение – Параметр*), обведем ее, *горизонтальное положение* для неё будет *Право*.

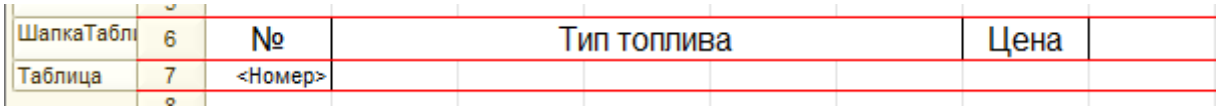


Рис. 11.1.38

Следующие пять ячеек объединяем, напишем слово *ТипТоплива*, сделаем эту объединенную ячейку параметром (свойство *Заполнение – Параметр*). *Горизонтальное положение* для неё будет *Право*.

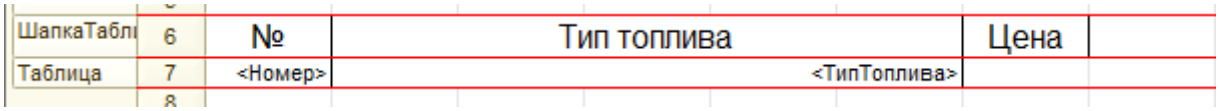


Рис. 11.1.39

В последней ячейке напишем слово *Цена*, сделаем её параметром. *Горизонтальное положение* для неё будет *Право*. Всем ячейкам установите 12 шрифт.

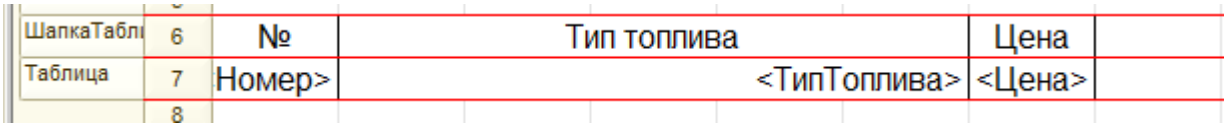


Рис. 11.1.40

Теперь создадим подвал таблицы, в котором выведем общее количество строк в документе. Для этого добавляем новую область, которую так и назовем *Подвал таблицы*.

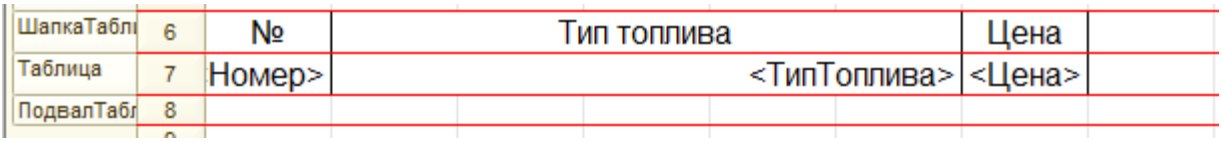


Рис. 11.1.41

Сверху ограничьте область жирной линией. Для этого выделите всю область подвала от номера до цены, зайдите в палитру свойств выделенных ячеек, установите границу сверху и поставьте толщину равную 2.

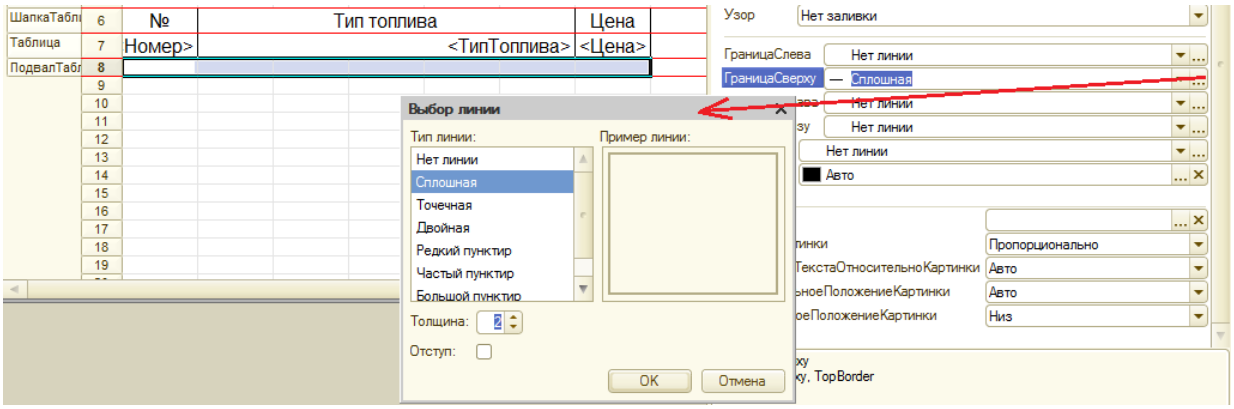


Рис. 11.1.42

Объедините ячейки под заголовком *Тип топлива* и напишите следующий текст в них:

Всего [КолТипов] типов топлива.

Сделайте эту объединенную ячейку шаблоном, выделите жирным и установите для неё 12 шрифт.

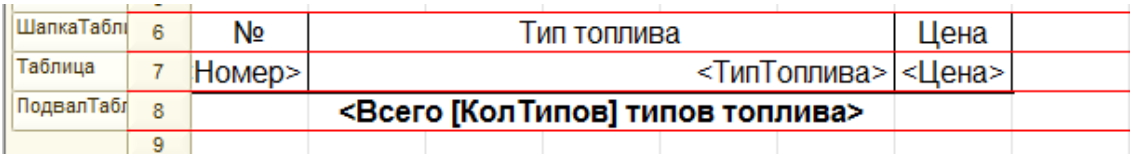


Рис. 11.1.43

Теперь Вам осталось создать подвал документа. Создадим новую область, которую назовем *Подвал*. И разместим в этой области подписи и расшифровку.

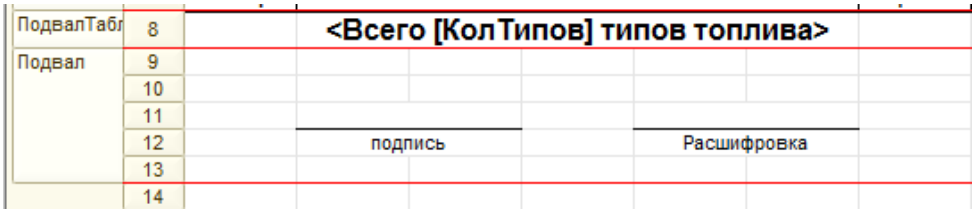


Рис. 11.1.44

Все, макет Вы создали, теперь осталось его вывести на экран. Сделаем это по аналогии с выводом простого макета: в модуле документа создадим экспортную функцию *ПечатьРеестраЦен*.

```
Функция ПечатьРеестраЦен() Экспорт

КонецФункции
```

Листинг 11.1.3

Первым делом нам необходимо получить параметры шапки документа и табличной части. В целях производительности мы будем получать эти параметры, используя запросы.

И первым запросом мы получим параметры шапки. Создайте объект «Запрос» и зайдите в конструктор запроса, где выберите таблицу *Установка цен на топлив* и поля *Дата* и *Поставщик*.

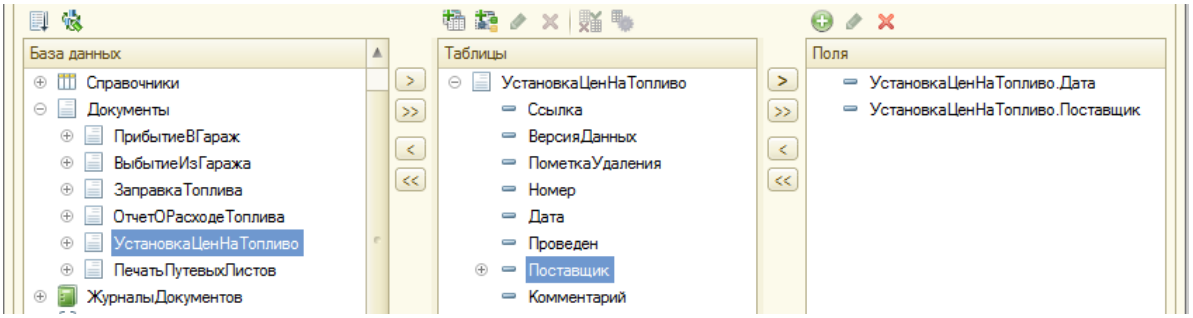


Рис. 11.1.45

Перейдите на закладку *Условие* и установите условие:

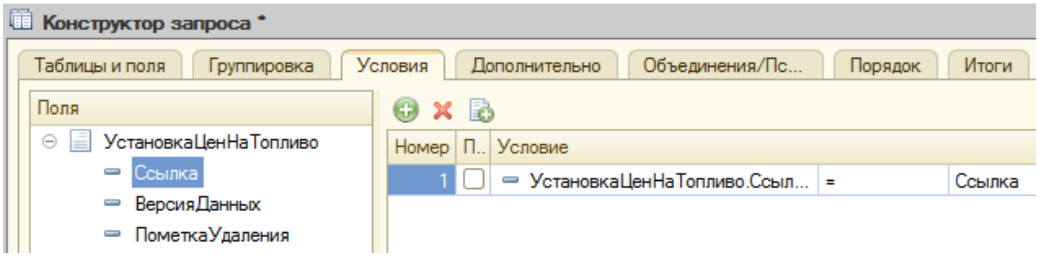


Рис. 11.1.46

Запрос создан, нажмите кнопку *ОК* и сохраните его, передайте в запрос параметры и сделайте выборку.

```
Запрос = Новый Запрос;
Запрос.Текст = "ВЫБРАТЬ
               |    УстановкаЦенНаТопливо.Дата КАК Дата,
               |    УстановкаЦенНаТопливо.Поставщик КАК Поставщик
               |ИЗ
               |    Документ.УстановкаЦенНаТопливо КАК УстановкаЦенНаТопливо
               |ГДЕ
               |    УстановкаЦенНаТопливо.Ссылка = &Ссылка";
Запрос.УстановитьПараметр("Ссылка",Ссылка);
Шапка = Запрос.Выполнить().Выбрать();
Шапка.Следующий();
```

Листинг 11.1.4

Следующим шагом создадим табличный документ и получим макет.

```
ТабДокумент = Новый ТабличныйДокумент;
Макет = ПолучитьМакет("Макет");
```

Листинг 11.1.5

Сразу же создадим все области, которые есть в макете.

```
ОблШапка      = Макет.ПолучитьОбласть("Шапка");
ОблШапкаТабл = Макет.ПолучитьОбласть("ШапкаТаблицы");
```

```
ОблТаблица    = Макет.ПолучитьОбласть("Таблица");
ОблПодвалТаб = Макет.ПолучитьОбласть("ПодвалТаблицы");
ОблПодвал     = Макет.ПолучитьОбласть("Подвал");
```

Листинг 11.1.6

Названия областей в кавычках, передаваемые в параметр метода ПолучитьОбласть, должны совпадать с названиями областей в макете.

После того как области созданы, мы будем их заполнять, если у них есть параметры, и последовательно выводить в результирующий табличный документ.

Заполните параметры области *Шапка*. Для этого воспользуемся методом *Заполнить* объекта *ПараметрыМакетаТабличногоДокумента*. Данный метод заполняет параметры из значений свойств переданного объекта. В качестве параметра в этот метод будем передавать выборку *Шапка*, которую мы получили с помощью метода *Выбрать* результата запроса. И которую мы позиционировали на первом элементе выборки при помощи метода *Следующий*.

```
ОблШапка.Параметры.Заполнить(Шапка);
ОблШапка.Параметры.Дата = Формат(Шапка.Дата,"ДЛФ=DD");
```

Листинг 11.1.7

Один параметр мы заполнили вручную. Это *Дата*.

Теперь выведем область *Шапка* в результирующий табличный документ. А также выведем область *Шапка таблицы*, так как в ней нет никаких параметров.

```
ТабДокумент.Вывести(ОблШапка);
ТабДокумент.Вывести(ОблШапкаТабл);
```

Листинг 11.1.8

Сначала всегда заполняем область, а потом выводим её в табличный документ!

Далее необходимо вывести все типы цен и цены, которые есть в табличной части. Сделаем мы это благодаря одной особенности областей: их можно выводить в табличный документ неограниченное количество раз.

Но перед этим получим данные из табличной части. Создайте новый текст запроса и зайдите в конструктор запроса, где необходимо выбирать табличную часть документа *Установка цен на топливо* (переименуем эту таблицу в *ЦеныТоплива*) и поля *НомерСтроки*, *ТипТоплива и Цена*.

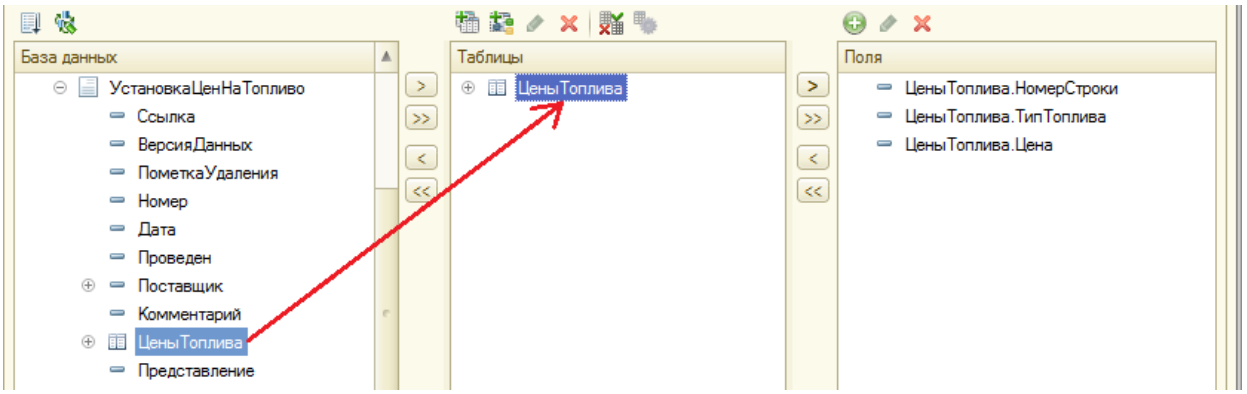


Рис. 11.1.47

Перейдите на закладку *Условие* и установите условие:

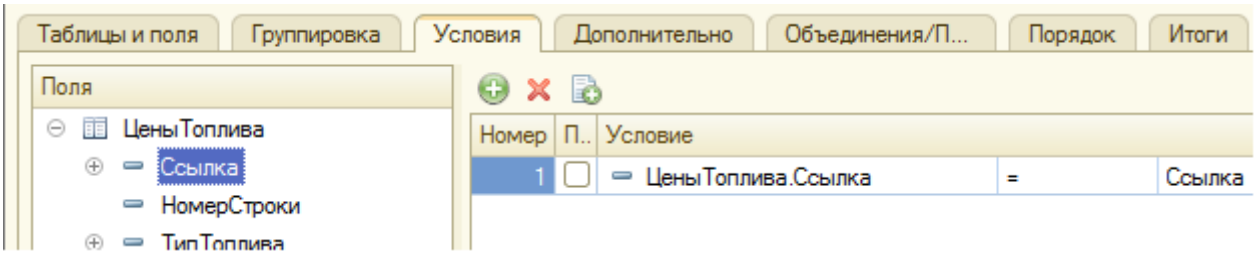


Рис. 11.1.48

Переименуем поле *НомерСтроки* в просто *Номер*.

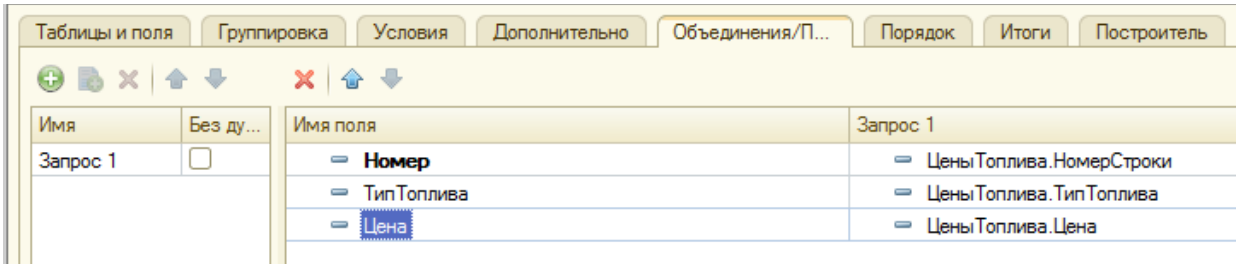


Рис. 11.1.49

Сохраним запрос, установим параметры и получим выборку.

```
Запрос = Новый Запрос;
Запрос.Текст = "ВЫБРАТЬ
               |    ЦеныТоплива.НомерСтроки КАК Номер,
               |    ЦеныТоплива.ТипТоплива КАК ТипТоплива,
               |    ЦеныТоплива.Цена КАК Цена
               |ИЗ
               |    Документ.УстановкаЦенНаТопливо.ЦеныТоплива КАК
ЦеныТоплива
               |ГДЕ
               |    ЦеныТоплива.Ссылка = &Ссылка";
Запрос.УстановитьПараметр("Ссылка",Ссылка);
Выборка = Запрос.Выполнить().Выбрать();
```

Листинг 11.1.9

Создадим цикл обхода выборки. И внутри этого цикла будем заполнять параметры области *Таблица* и выводить ее в результирующий документ.

```
Пока Выборка.Следующий() Цикл
    ОблТаблица.Параметры.Заполнить(Выборка);
    ТабДокумент.Вывести(ОблТаблица);
КонецЦикла;
```

Листинг 11.1.10

После цикла заполним параметры подвала таблицы. И выведем подвал таблицы и общий подвал.

```
ОблПодвалТаб.Параметры.КолТипов = Выборка.Количество();
ТабДокумент.Вывести(ОблПодвалТаб);
ТабДокумент.Вывести(ОблПодвал);
```

Листинг 11.1.11

И поскольку у нас *Печать* - это функция модуля документа, то мы вернем табличный документ.

```
Возврат ТабДокумент;
```

Листинг 11.1.12

Теперь по аналогии с документом *Прибытие в гараж* создадим команду формы *ПечатьРеестра*, которую разместим в командной панели формы.

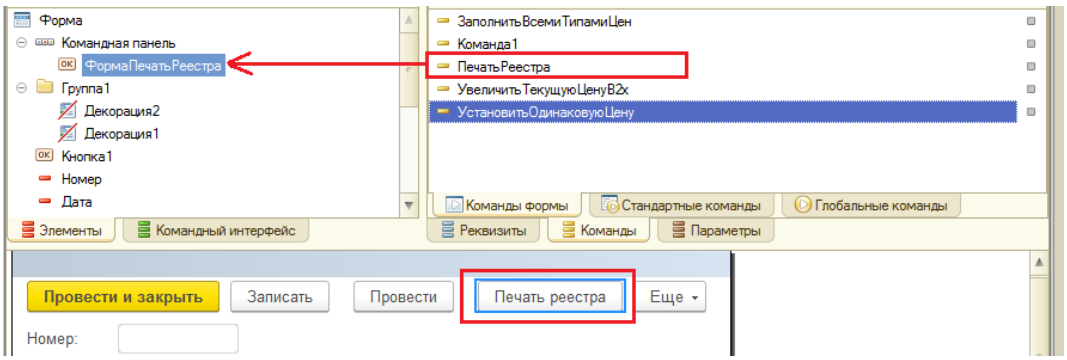


Рис. 11.1.50

Создадим обработчик этой команды на клиенте. И напишем следующий код:

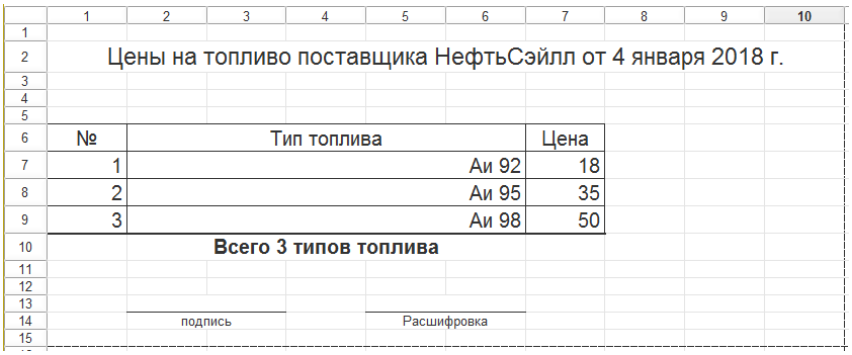
```
&НаСервере
Функция ПечатьРеестраНаСервере()
     ОбъектДок = РеквизитФормыВЗначение("Объект");
     ТабПечати = ОбъектДок.ПечатьРеестраЦен();
     Возврат ТабПечати;
КонецФункции
```

```
&НаКлиенте
Процедура ПечатьРеестра(Команда)
    ТабРеестра = ПечатьРеестраНаСервере();
    ТабРеестра.Показать();
КонецПроцедуры
```

Листинг 11.1.13

В этом коде мы в функции *ПечатьРеестраНаСервере* получаем объект документа при помощи метода формы *РеквизитФормыВЗначение*. Этот объект нам нужен, чтобы использовать экспортную функцию модуля объекта *ПечатьРеестраЦен*. Благодаря этой функции мы получаем табличный документ, который и возвращаем. А в процедуре *ПечатьРеестра* мы этот табличный документ выводим на экран.

Должна получиться такая печатная форма:



Цены на топливо поставщика НефтьСэйлл от 4 января 2018 г.

№	Тип топлива	Цена
1	Аи 92	18
2	Аи 95	35
3	Аи 98	50

Всего 3 типов топлива

подпись Расшифровка

Рис. 11.1.51

Как Вы видите, печатная форма вышла в том виде, в каком мы и хотели.

Резюме

В этой части мы научились создавать примитивные печатные формы для документов, заполнять их и выводить на экран. Вы теперь знаете, как создавать макеты, заполнять их и передавать параметры как непосредственно в ячейку, так и в шаблон. Также Вы можете выводить печатную форму любого документа, как с табличной частью, так и без нее. Эти знания очень пригодится Вам в дальнейшей работе.

Часть 2. Отчеты

В предыдущих главах мы любую информацию выводили в таблицы на форме. Для пользователя получать информацию в такой форме немного неудобно. Естественно, есть более удобный способ получать информацию о хозяйственной деятельности предприятия. Для этого необходимо использовать такие объекты конфигурации, как *Отчеты*. Можно создавать отчеты как объект конфигурации, а можно создавать и внешний отчет.

В этой части мы научимся создавать простые отчеты при помощи табличного документа и при помощи СКД. В этой книге я не ставлю цель научить Вас делать различные сложные отчеты, а просто даю базовые понятия, чтобы Вы понимали, как это работает. И в дальнейшем уже смогли углубить свои знания и умения.

Простые отчеты

Для демонстрации возможности работы с отчетами создадим простой отчет по запросу из второй части девятой главы, в котором мы получали список автомобилей.

Создайте для этого внешний отчет (аналогично созданию внешней обработке) и назовите его *«Глава 11 часть 2 отчет 1»*. Сохраните его в любом месте на жестком диске.

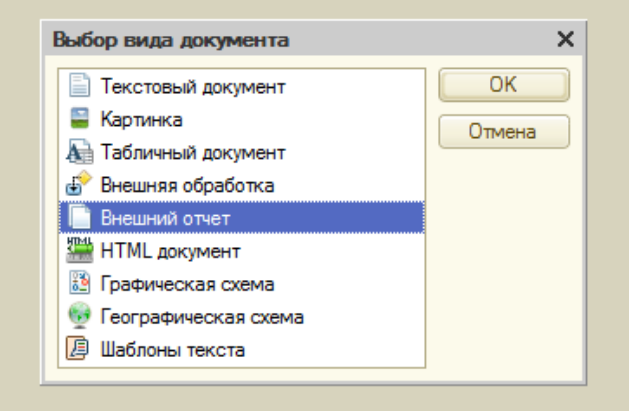


Рис. 11.2.1

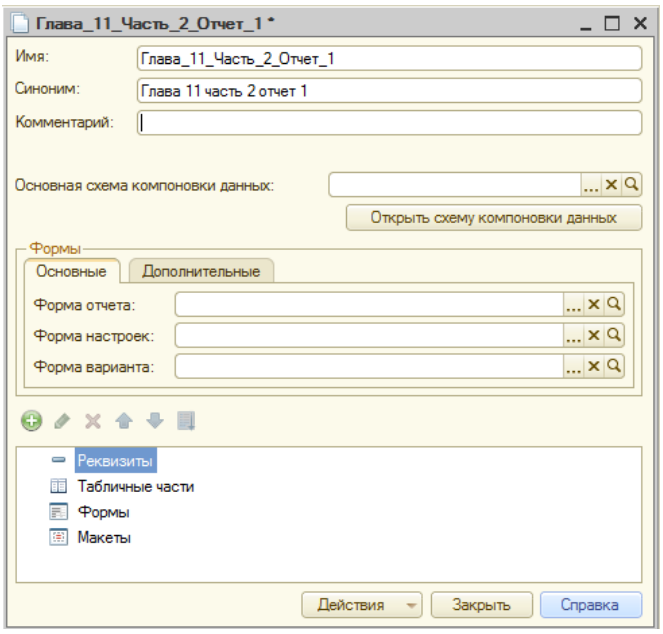


Рис. 11.2.2

Теперь Вам необходимо создать форму отчета и макет. Первым делом создадим форму отчета.

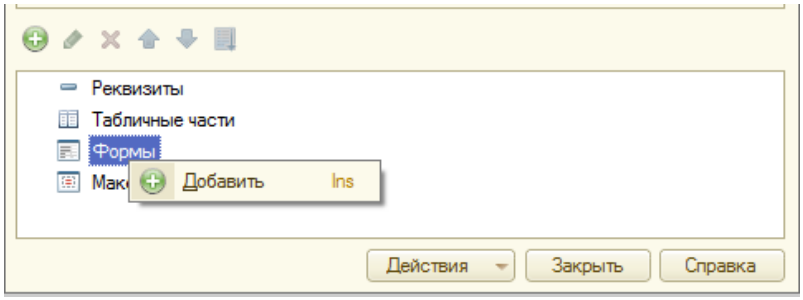


Рис. 11.2.3

На этой форме создайте реквизит, который назовите «ТаблДокумент» (заголовок *Отчет*) и который будет иметь тип *Табличный документ*.

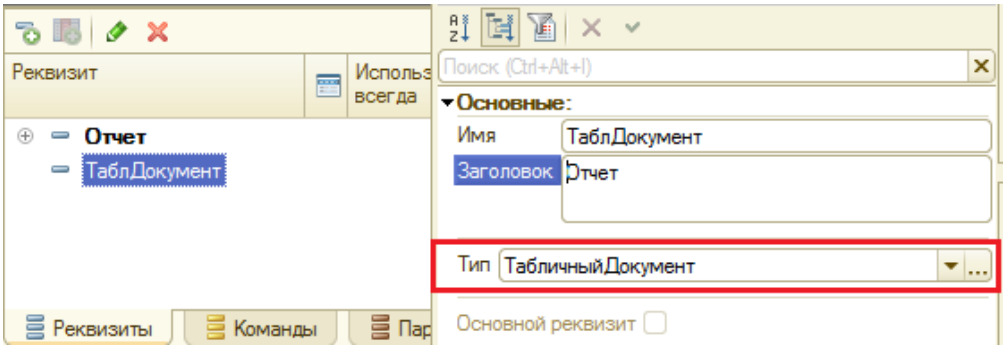


Рис. 11.2.4

«Перетащите» реквизит на форму. Будет создан элемент формы поле ввода с новым для Вас видом - *Поле табличного документа*.

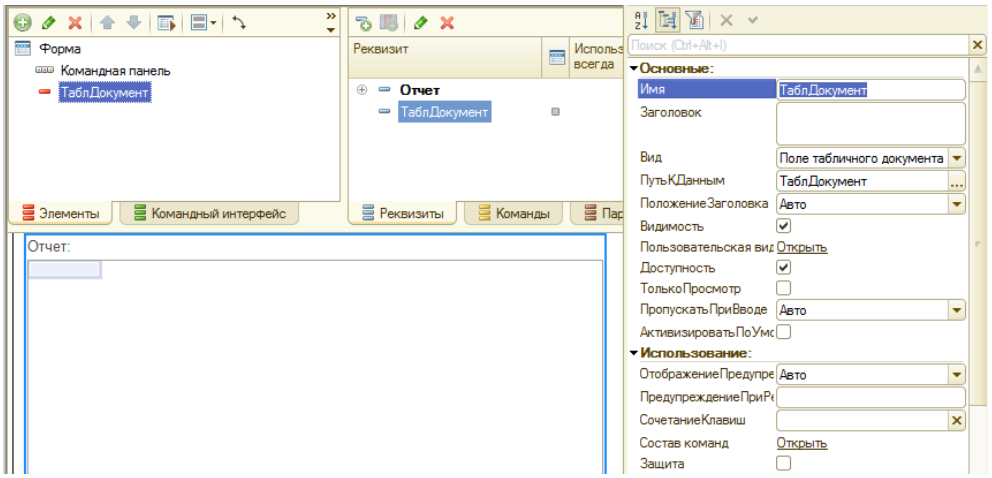


Рис. 11.2.5

Создадим команду *Печать*, которую разместим на форме.

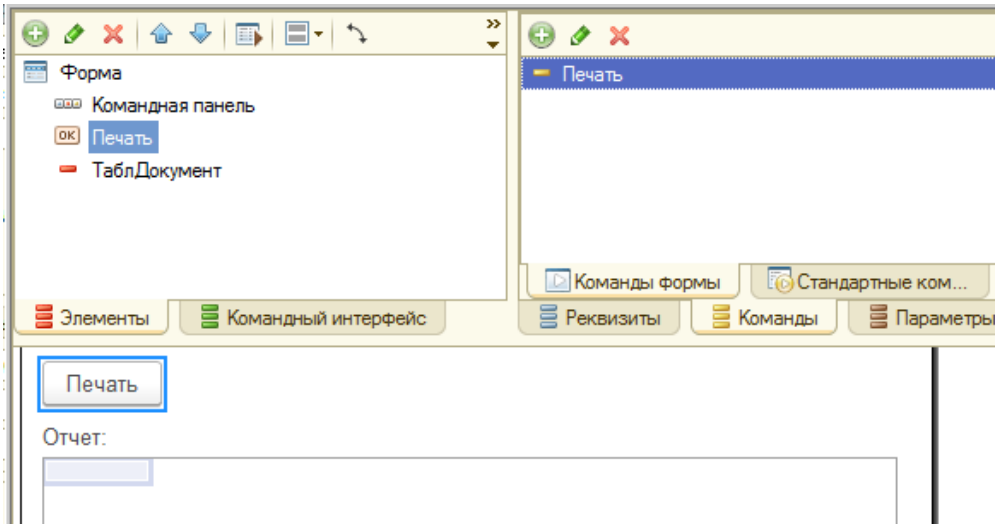


Рис. 11.2.6

Создадим макет этого внешнего отчета.

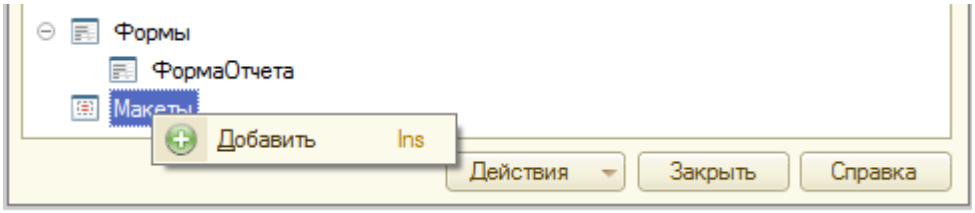


Рис. 11.2.7

Тип макета оставим по умолчанию – табличный документ.

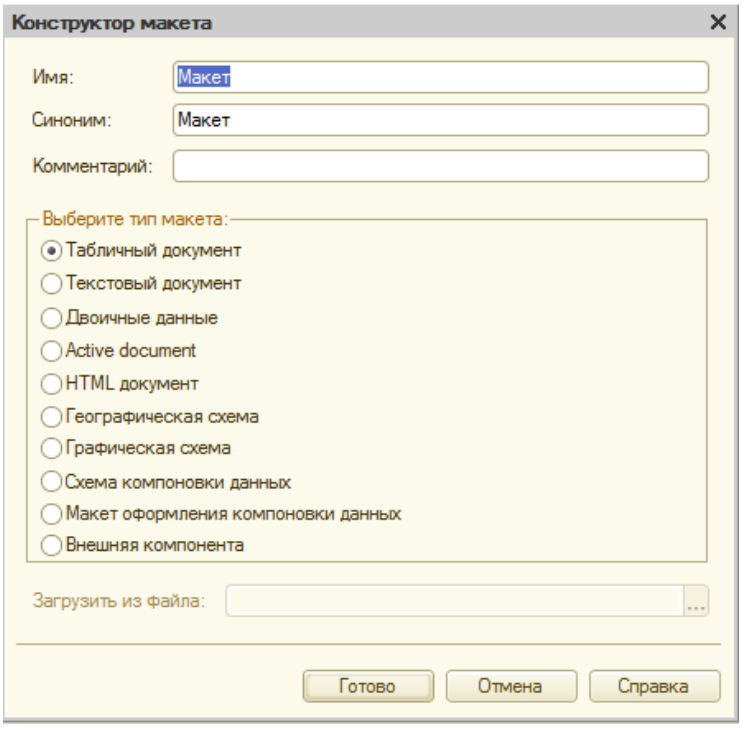


Рис. 11.2.8

В данном макете создайте четыре области. Первую назовите *Шапка*.

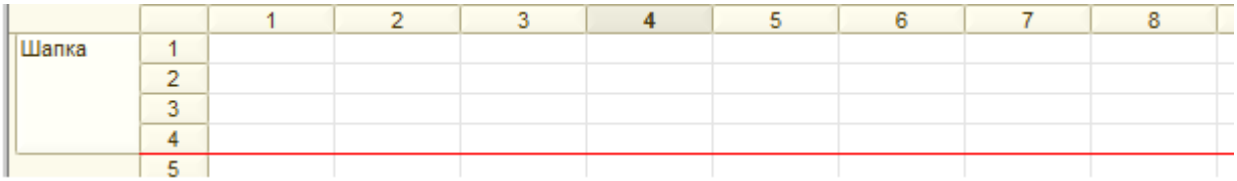


Рис. 11.2.9

Вторую – *ШапкаТаблицы.*

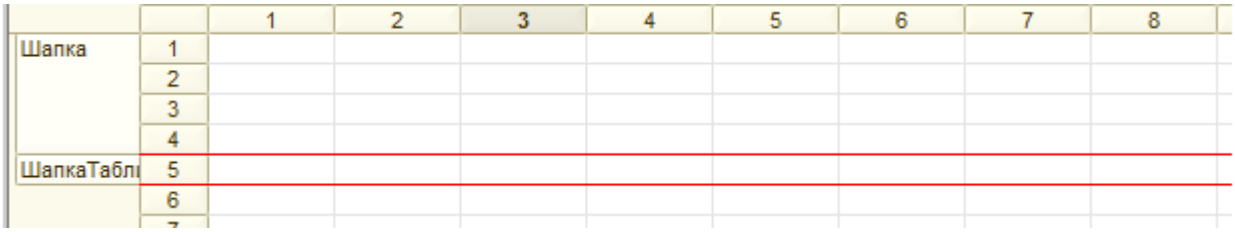


Рис. 11.2.10

Третья область – *СтрокиТаблицы*.

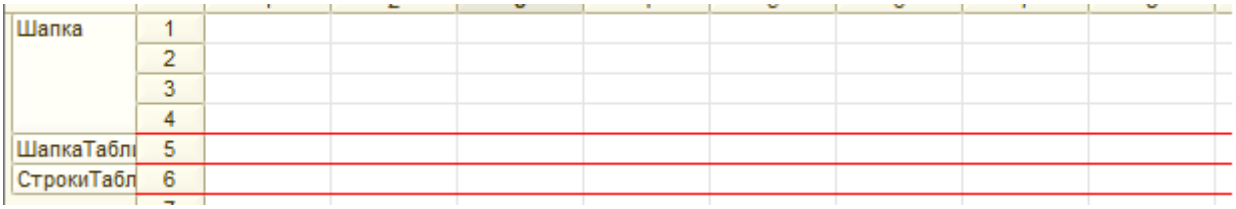


Рис. 11.2.11

Четвертая область – *Подвал*.

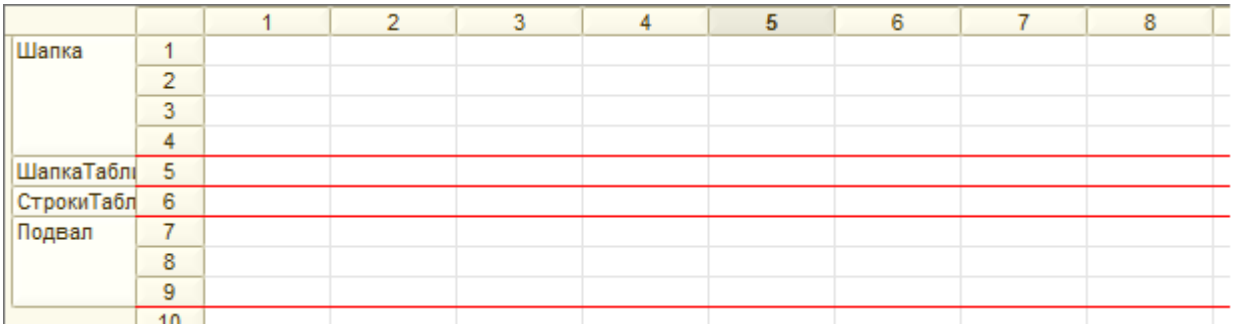


Рис. 11.2.12

В первой области напишите следующий текст: «Список автомобилей», установим тексту 14 шрифт и объединим ячейки.

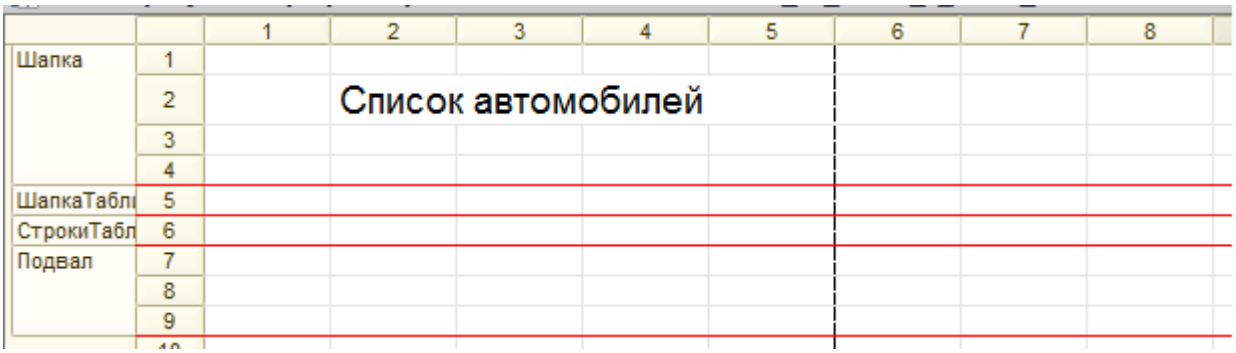


Рис. 11.2.13

В шапке таблицы в каждой ячейке напишем *«№», «Название авто», «Год выпуска», «Гос. номер», «Марка» и «Модель»*. Обведем их границами, установим 12 шрифт, и раздвинем колонки, чтобы все названия влезли.

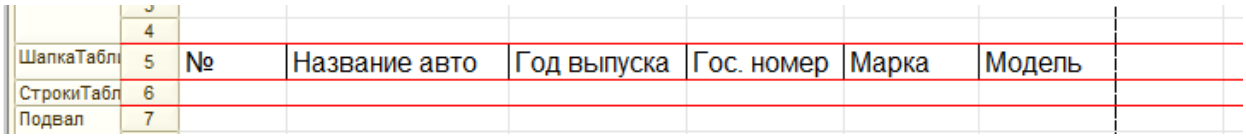


Рис. 11.2.14

В области *Строки таблицы* напишите те же названия: *«Номер», «НазваниеАвто», «ГодВыпуска», «ГосНомер», «Марка» и «Модель»*, только установите для каждой ячейки в свойство *Заполнение* значение *Параметр*, обведем ячейки и установим для них 12 шрифт.

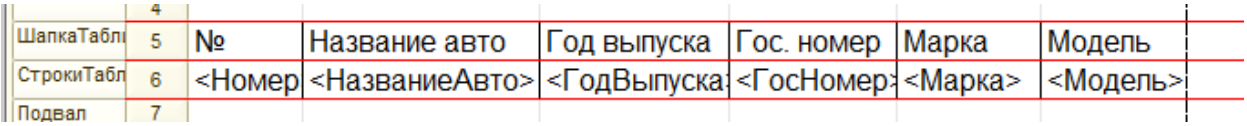


Рис. 11.2.15

А для ячейки *НазваниеАвто* области «Строка таблицы» установим в свойство *РазмещениеТекста* значение *Переносить*. Теперь если название авто будет не влезать в ширину колонки, то оно перенесется на другую строку.

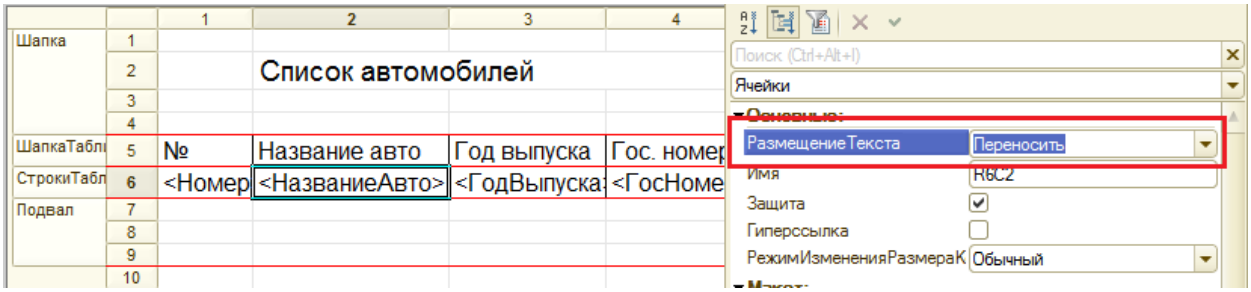


Рис. 11.2.16

В подвале проведите жирную черту, чтобы она выходила на нижней границе таблицы. Для этого выделяем верхние ячейки подвала.

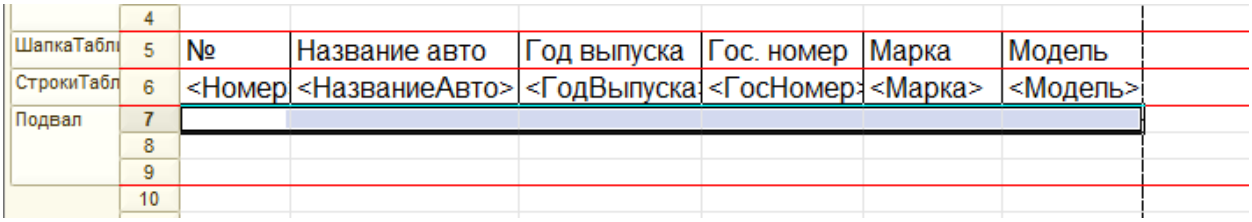


Рис. 11.2.17

Переходим в палитру свойств для всех этих ячеек (можно открыть общее свойство для нескольких ячеек) и поставим для них верхнюю границу с двойной линией.

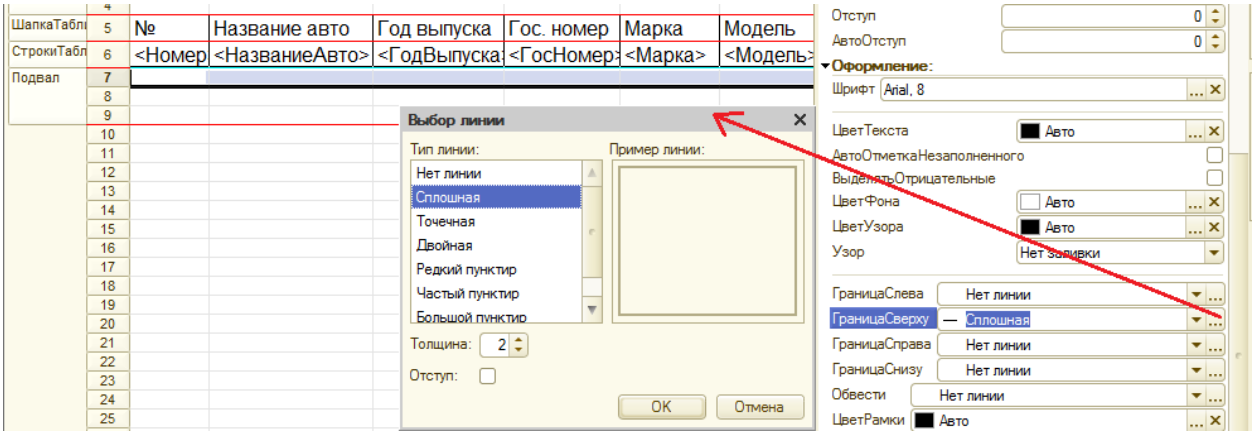


Рис. 11.2.18

Макет готов, теперь осталось написать вывод информации в данный макет. Делать мы это будем в обработчиках команды *Печать*, которую создали ранее. Создайте обработчики этой команды на сервере и на клиенте. И в серверном обработчике напишем код, который будет выводить информацию об автомобилях в табличный документ.

```
    &НаСервере
Процедура ПечатьНаСервере()
    ТаблДокумент.Очистить();
```

```
    ОтчетОбъект = РеквизитФормыВЗначение("Отчет");
    Макет = ОтчетОбъект.ПолучитьМакет("Макет");


    ОбластьШапка = Макет.ПолучитьОбласть("Шапка");
    ОбластьШапкаТаблицы = Макет.ПолучитьОбласть("ШапкаТаблицы");
    ОбластьСтрокиТаблицы = Макет.ПолучитьОбласть("СтрокиТаблицы");
    ОбластьПодвал = Макет.ПолучитьОбласть("Подвал");

    ТаблДокумент.Вывести(ОбластьШапка);
    ТаблДокумент.Вывести(ОбластьШапкаТаблицы);

    Запрос = Новый Запрос;
    Запрос.Текст = "ВЫБРАТЬ
                   |    Автомобили.Наименование КАК НазваниеАвто,
                   |    Автомобили.ГодВыпуска КАК ГодВыпуска,
                   |    Автомобили.ГосНомер КАК ГосНомер,
                   |    Автомобили.Марка КАК Марка,
                   |    Автомобили.Модель КАК Модель
                   |ИЗ
                   |    Справочник.Автомобили КАК Автомобили
                   |ГДЕ
                   |    НЕ Автомобили.ПометкаУдаления";

    Выборка = Запрос.Выполнить().Выбрать();
    Номер = 1;
    Пока Выборка.Следующий() Цикл
        ОбластьСтрокиТаблицы.Параметры.Заполнить(Выборка);
        ОбластьСтрокиТаблицы.Параметры.Номер = Номер;
        ТаблДокумент.Вывести(ОбластьСтрокиТаблицы);
        Номер = Номер + 1;
    КонецЦикла;

    ТаблДокумент.Вывести(ОбластьПодвал);

КонецПроцедуры

&НаКлиенте
Процедура Печать(Команда)
    ПечатьНаСервере();
КонецПроцедуры
```

Листинг 11.2.1

В этом коде мы обращаемся к реквизиту формы с типом *ТабличныйДокумент* напрямую и первым делом его очищаем. Потом получаем *Макет* нашего отчета, используя метод *ПолучитьМакет* объекта *ВнешнийОтчет*. Этот объект является основным реквизитом формы.

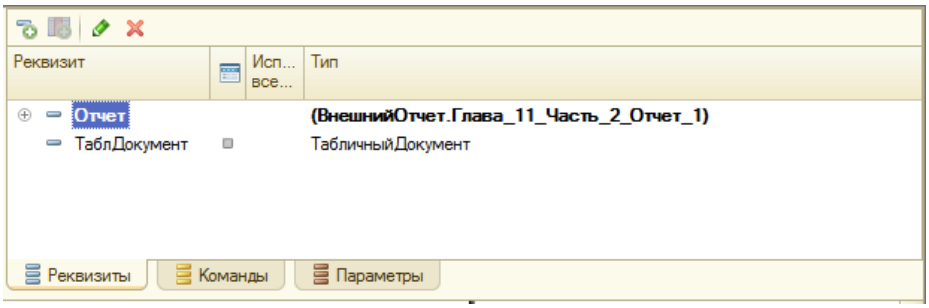


Рис. 11.2.19

Дальнейший код Вам должен быть знаком и не должен вызывать каких-либо затруднений: мы получаем области макета и выводим их в табличном документе. Заметьте, в данном случае мы не показываем табличный документ, как делали в предыдущих примерах. Вся нужная информация отобразится на поле в форме.

Список автомобилей

№	Название авто	Год выпуска	Гос. номер	Марка	Модель
1	Автомобиль главного директора	01.01.2016 0:	х001кк18RU	Ford	Focus
2	Дежурный автомобиль	10.05.2015 0:	кк123ка18	Ford	Focus
3	Авто мастера	10.10.2010 0:	ув112ц19	Ford	Focus
4	Автомобиль главного бухгалтера	01.01.2016 0:	х002бб18RU	KIA	Rio
5	Автомобиль главного инженера	01.01.2016 0:	х033ув18RU	Renault	

Рис. 11.2.20

Отчеты на СКД

В этой части мы научимся делать самые простые отчеты на СКД. Я не буду подробно вдаваться в основы работы под СКД, а покажу самые азы работы, чтобы Вам было в дальнейшем с чего начинать. Если Вы захотите углубить этот вопрос, то сможете найти соответствующую литературу, курсы и т.д. После этой книги весь узконаправленный материал Вами легче усвоится.

Что такое СКД? СКД - это инструмент платформы 1С, который позволяет гибко и динамически получать данные из базы. Мы научимся создавать простейшие отчеты на СКД.

Для тренировки сделаем внешний отчет из 10-й главы, где мы в запросе получали обороты по регистру накопления *ТопливоВАвтомобилях*.

Создайте внешний отчет и сохраните его на жестком диске. В конструкторе отчета нажмите на кнопку «Открыть схему компоновки данных» (см. рис. 11.2.21), после нажатия откроется конструктор макета, в котором уже будет выбран тип макета *Схема компоновки данных*.

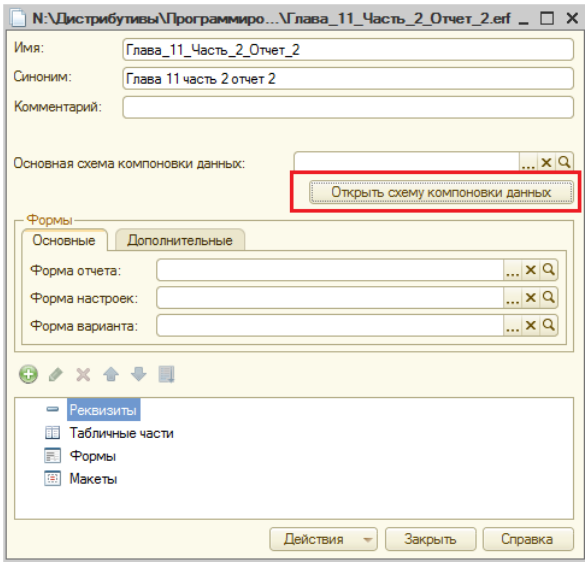


Рис. 11.2.21

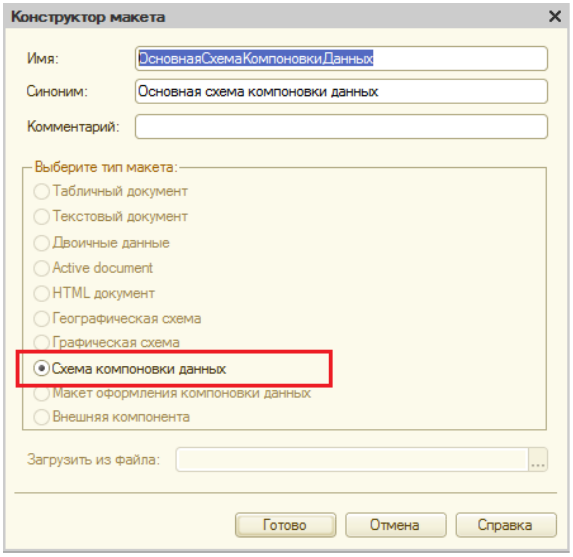


Рис. 11.2.22

Нажимаем кнопку «Готово», и откроется конструктор СКД.

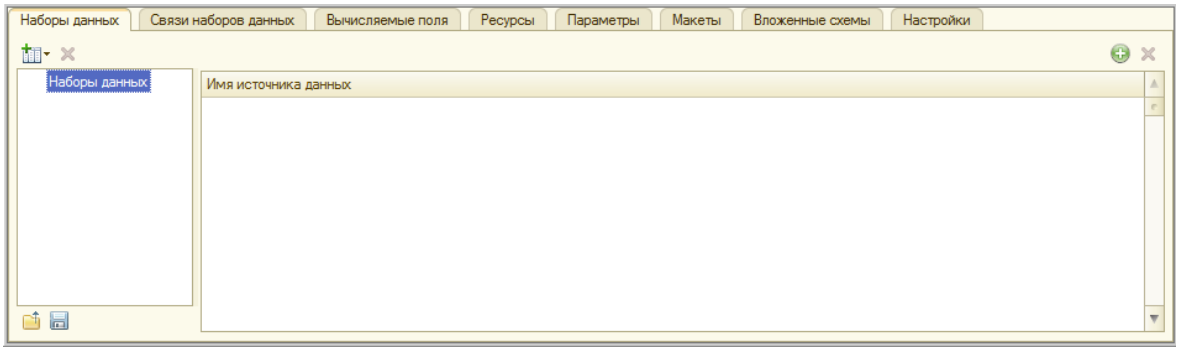


Рис. 11.2.23

С этим конструктором и будет осуществляться наша основная работа. И первым шагом нам нужно создать набор данных. Всего в СКД существует три набора данных – «Запрос», «Объект» и «Объединение». В первом варианте мы укажем какой-то запрос, по которому будет из базы данных получена информация. Во втором варианте нужно указать конкретный объект. Этот вариант применяется, когда мы работаем с СКД в программном коде, тогда мы можем передать в СКД различные объекты (например, таблицу значений). И в третьем варианте мы можем объединить несколько разных наборов (к примеру, два набора запроса). В этой книге мы разберем только один вид набора данных – запрос.

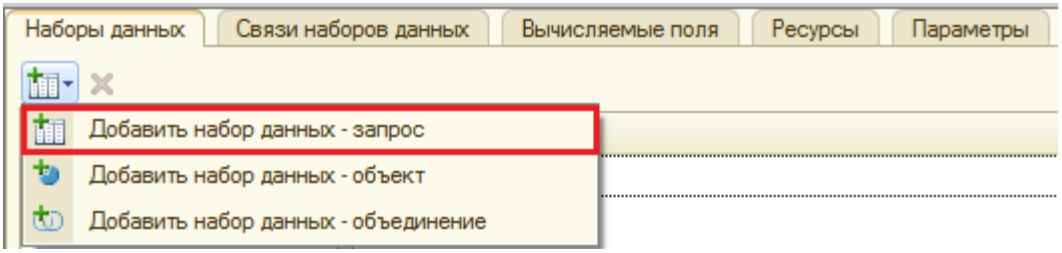


Рис. 11.2.24

После добавления набора данных – *запрос*, внизу появится поле запроса, а вверху таблица полей, которая пока пустая. Для добавления запроса нажмем кнопку «Конструктор запроса».

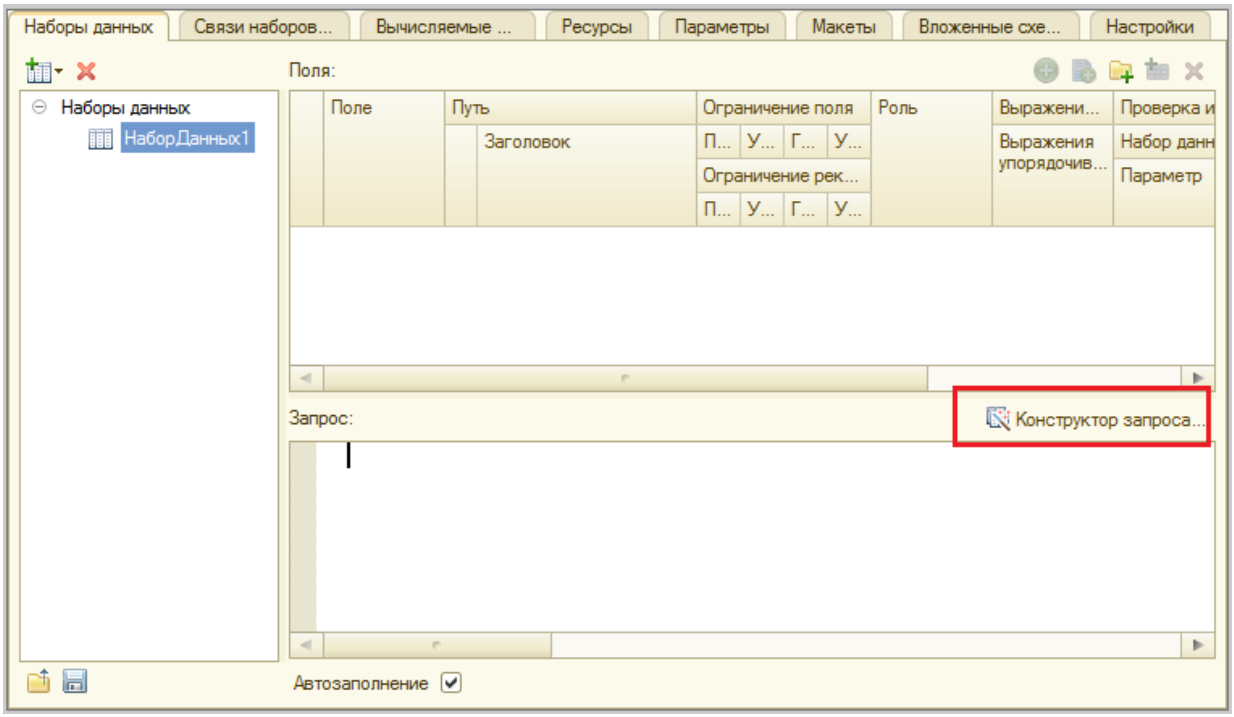


Рис. 11.2.25

После этого откроется уже знакомый Вам конструктор запроса, в этом запросе мы выберем уже знакомую Вам виртуальную таблицу «Обороты» регистра накопления *ТопливоВАвтомобилях* и выберем поля *Автомобиль*, *ТипТоплива*, *КоличествоПриход* и *КоличествоРасход*.

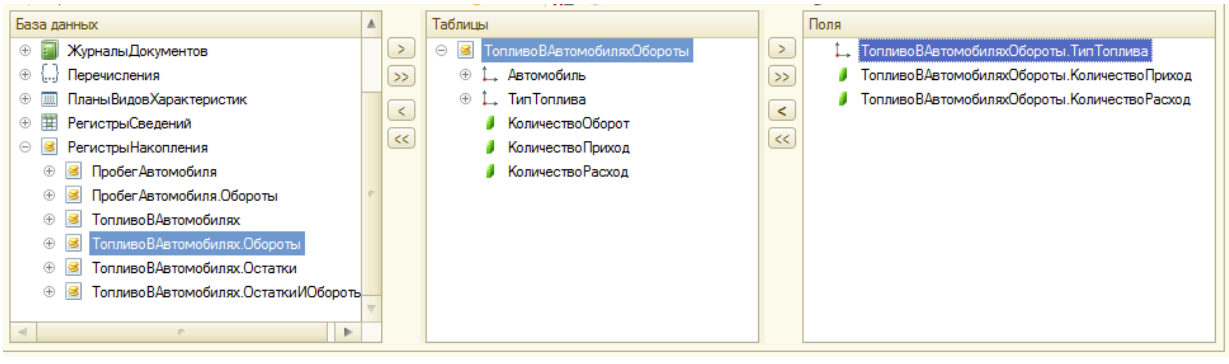


Рис. 11.2.26

Откроем окно «Параметры» виртуальной таблицы, в которой зададим только периодичность «День». Параметры *НачалоПериода* и *КонецПериода* задавать не нужно, платформа сама их определит. Различные отборы в условии нужно задавать, когда мы хотим делать конкретный жесткий отбор, в иных же случаях делать это не нужно. Все отборы можно делать в СКД.

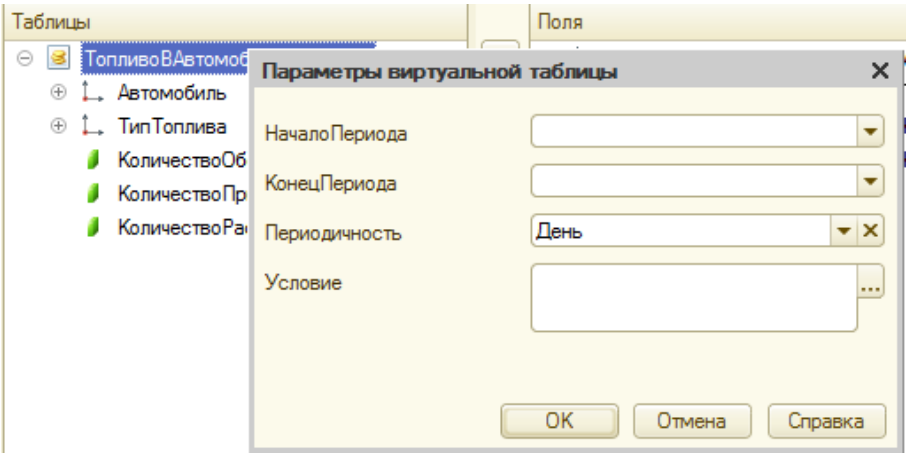


Рис. 11.2.27

Всё. Больше ничего делать не будем, нажмем кнопку *ОК*. И наша форма конструктора СКД изменится: в нижнем окне появится текст нашего запроса. А в верхнем - поля этого запроса. Обратите внимание на флаг *Автозаполнение*. Поля в верхней части появились, потому что он выставлен. Если этот флаг снять, то верхняя таблица очистится.

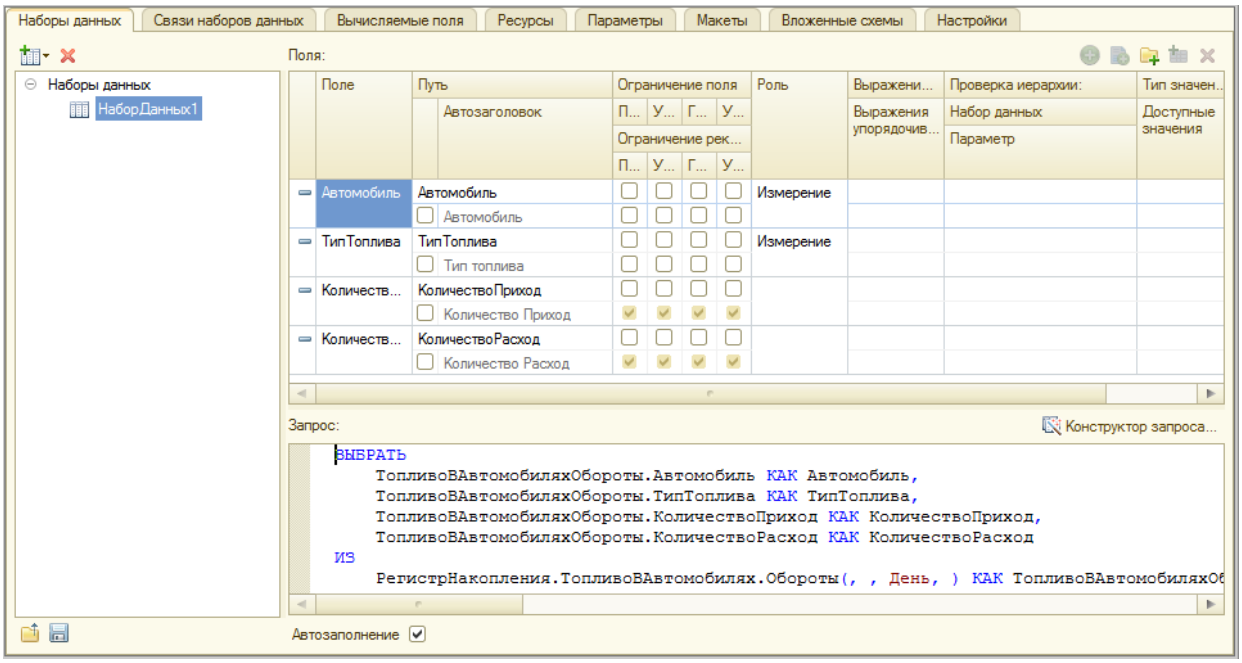


Рис. 11.2.28

Поработаем с верхней таблицей – исправим наименования полей «Количество Приход» и «Количество Расход» на «Приход» и «Расход» соответственно. Тем самым мы изменим только отображение поля в отчете. А в СКД мы будем работать с теми названиями, которые пришли из запроса. Для этого установим флаг во второй колонке и вручную исправим названия.

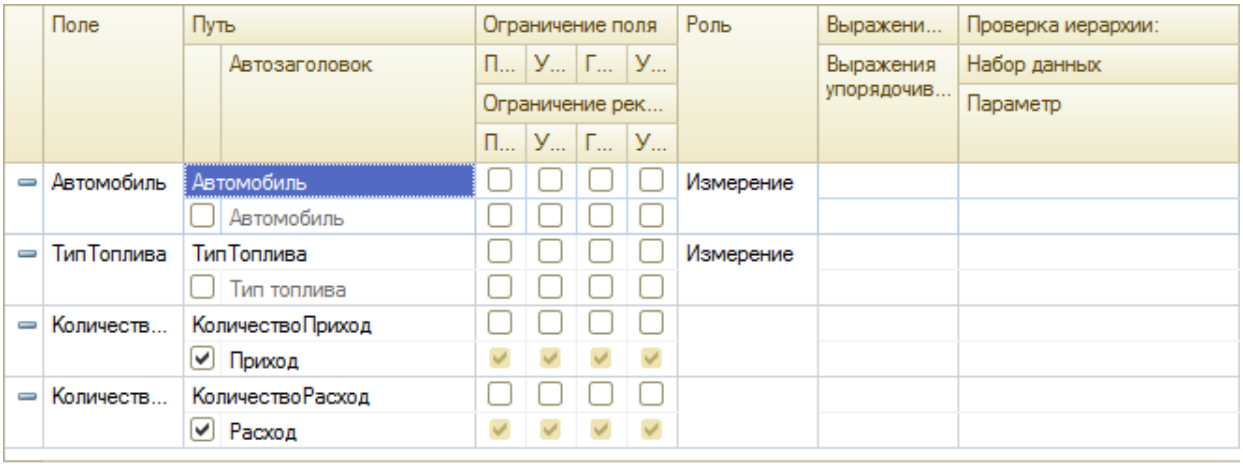


Рис. 11.2.29

Следующим шагом нам нужно определить поля, которые будут суммироваться. В СКД можно настраивать получение итогов по любому нужному Вам полю. Например, мы можем узнать, сколько любого топлива было залито в конкретный автомобиль, или наоборот, сколько конкретного топлива было израсходовано всеми автомобилями. Для этого нам нужно определить поля, которые будут суммироваться, делается это на закладке «Ресурсы».

В этой закладке мы выберем два поля - «КоличествоПриход» и «КоличествоРасход». И установим для них выражение «Сумма».

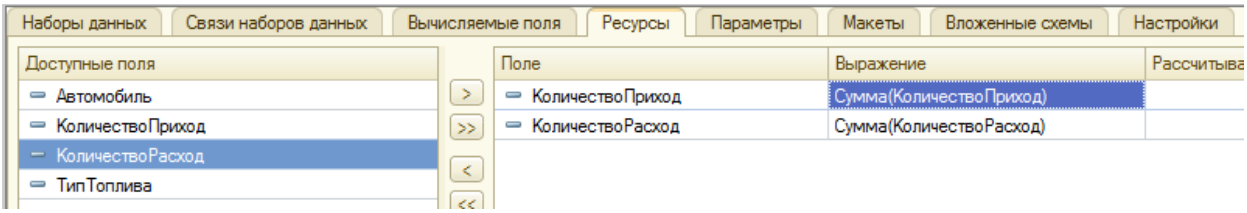


Рис. 11.2.30

Перейдем на закладку «Параметры», Вы видите, СКД автоматически подставило начало периода и конец периода.

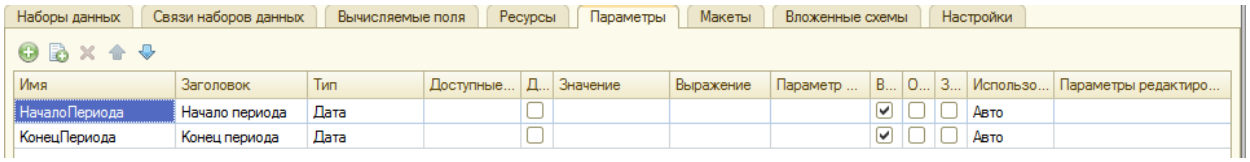


Рис. 11.2.31

Но работать через две даты на форме неудобно и непрактично. Сделаем один параметр с типом *СтандартныйПериод*. Для этого добавим в таблице параметров новый параметр, назовем его «Период» и тип ему назначим *СтандартныйПериод*.

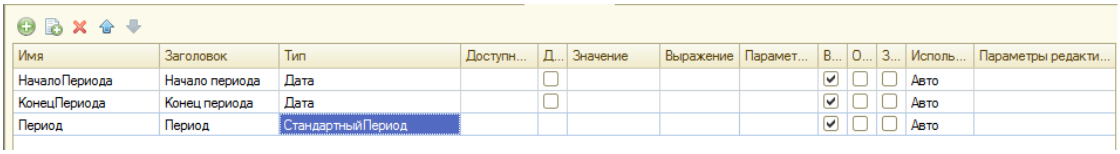


Рис. 11.2.32

Теперь нам нужно сделать так, чтобы данные нашего стандартного периода были переданы в параметры *НачалоПериода* и *КонецПериода.* Для этого будем использовать поле таблицы *Выражение*. В этом поле прописывается выражение, которое может принять тот или иной параметр. Мы передадим в качестве параметра свойств *ДатаНачала* и *ДатаОкончания* объекта *СтандартныйПериод*. И поставим флажки «Ограничения» доступности у полей *НачалоПериода* и *КонецПериода*, чтобы пользователь их не видел.

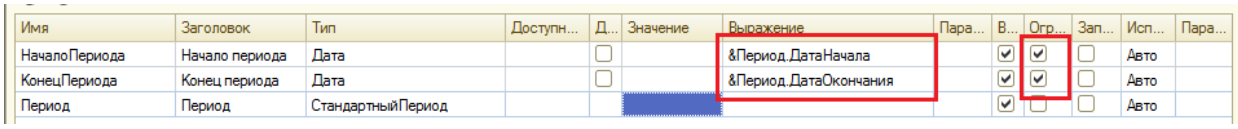


Рис. 11.2.33

Всё. Само ядро будущего отчета мы сделали. Именно эти три закладки – *Набор данных*, *Ресурсы* и *Параметры* – определяют, какие данные и как будут появляться в наших отчетах (есть еще закладки *Связи наборов данных* и *Вычисляемы поля*, которые тоже влияют на состав данных, но мы их не будем затрагивать).

Осталось настроить вид, как наш отчет будет выводиться пользователю. Для этого перейдем на закладку *Настройки*.

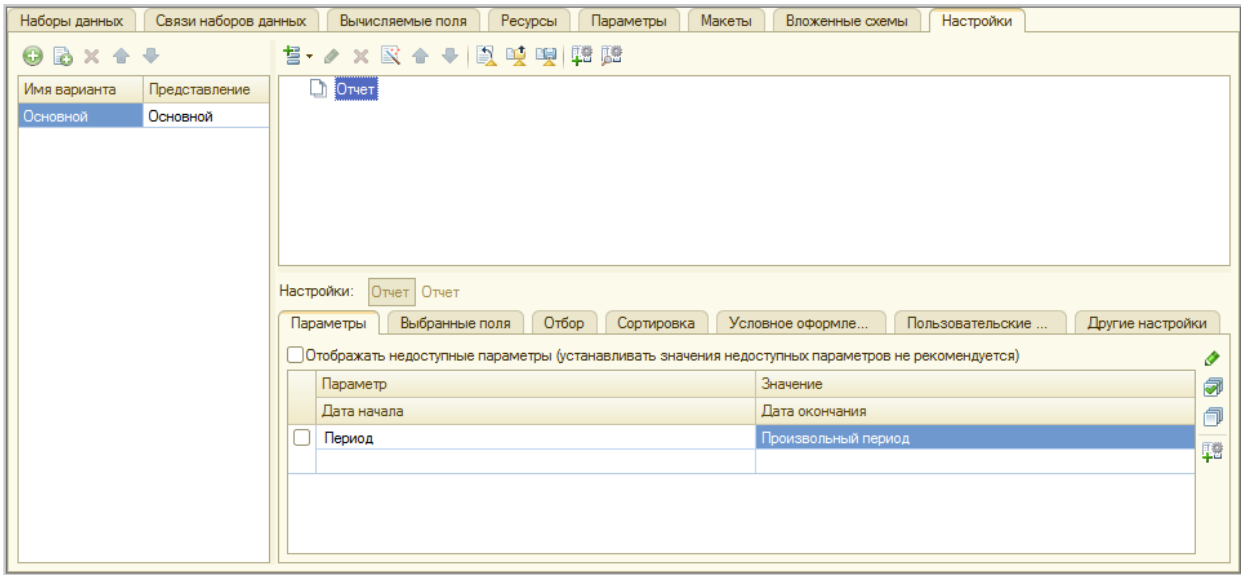


Рис. 11.2.34

Сделаем так, чтобы поле период отображалось на форме. Для этого выделите строку с параметром *Период* закладки «Параметры», расположенной в нижней части окна, и нажмите на кнопку «Свойства элементов пользовательских настроек». В открывшемся окне установите флаг «Включать в пользовательские настройки». После этого на форме отчета (она сформируется

автоматически) появится нужное поле. Для того, чтобы данные брались сразу, можно установить флаг рядом с параметром *Период*.

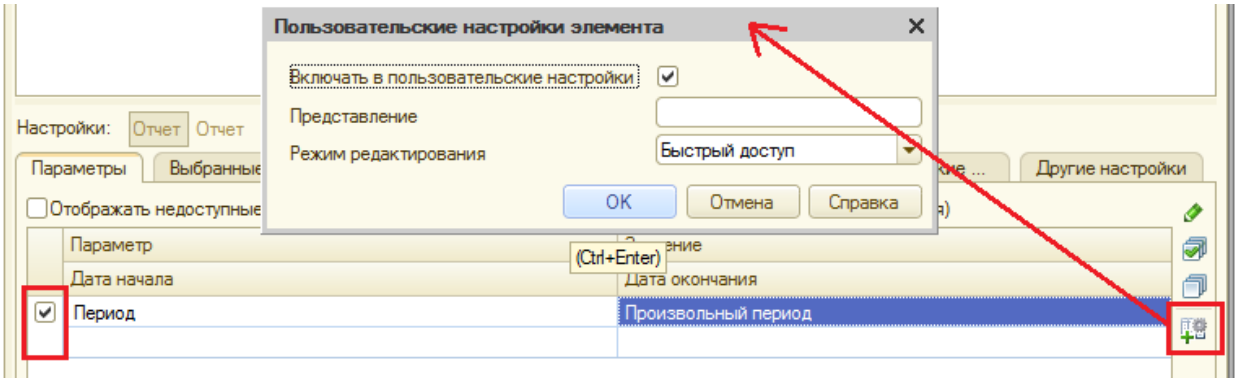


Рис. 11.2.35

Сделаем возможность установки отбора по автомобилю. Для этого перейдем на закладку «Отборы» и выберем поле «Автомобиль».

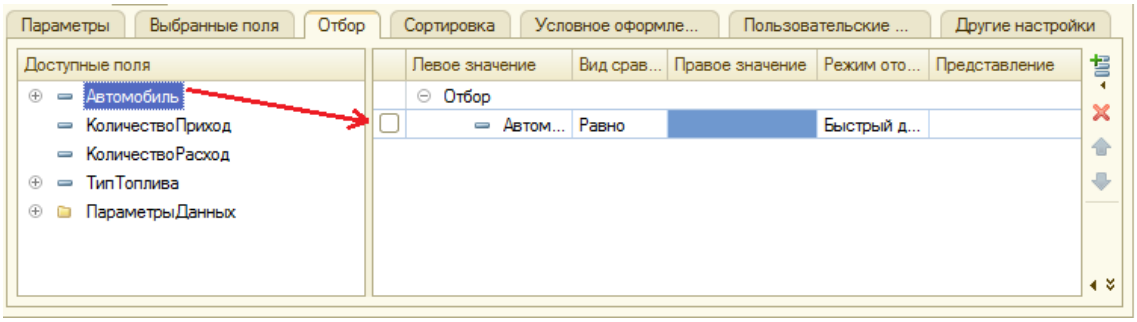


Рис. 11.2.36

Если мы не установим флаг в строке отборов, то по умолчанию отбор не будет работать (что нам, в принципе, и нужно). Но очень удобно, если возможность установки отбора будет на форме, так же, как и период. В этом нам поможет все также кнопка «Включать в пользовательские настройки». Выделите строку отбора с автомобилем, нажмите на кнопку «Включать в пользовательские настройки», а в открывшемся окне установите флаг «Включать в пользовательские настройки».

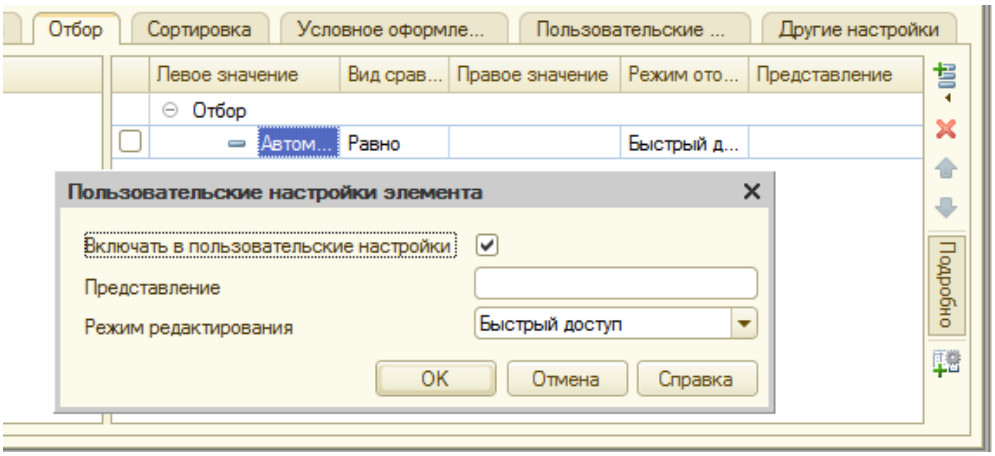


Рис. 11.2.37

Осталось настроить вид отображения пользователю. Для этого обратите внимание на верхнее окно. Зададим в нем две группировки – *Автомобиль* и *ТипТоплива*. Для этого нажмите на кнопку «Добавить» и выберите *Новая группировка*.

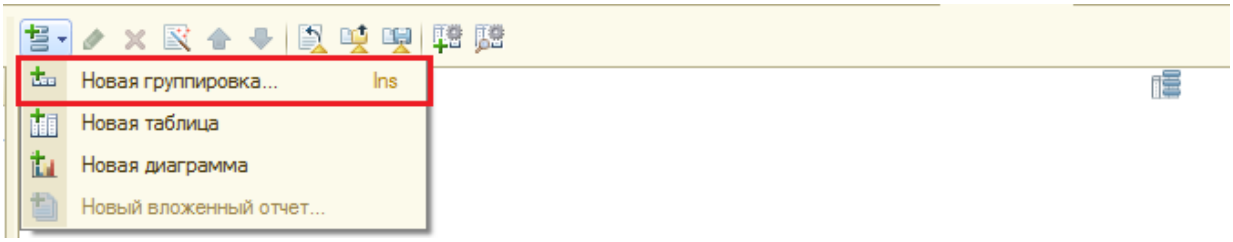


Рис. 11.2.38

В открывшемся окне «Группировка» выберем поле «Автомобиль».

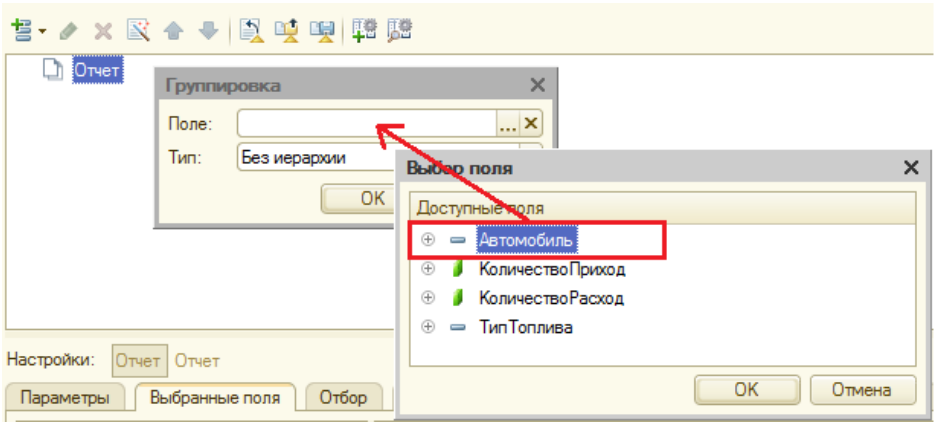


Рис. 11.2.39

Поскольку нам нужно сделать две группировки *Автомобиль* – *ТипТоплива*, то добавим поле *ТипТоплива*, которое будет подчинено уже добавленному полю *Автомобиль*.

Для этого выделите уже добавленное поле *Автомобиль* и добавьте поле *ТипТоплива*, как мы это уже делали.

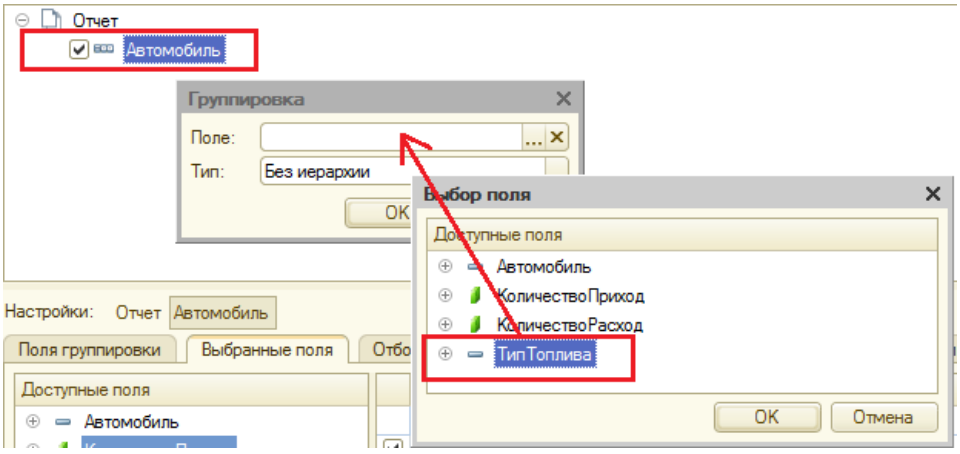


Рис. 11.2.40

Должна получиться следующая иерархия:

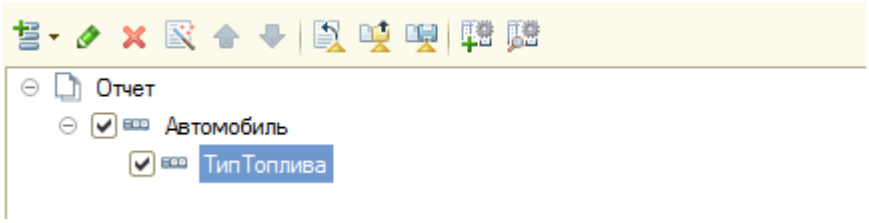


Рис. 11.2.41

Теперь сделаем так, чтобы в нашем отчете отображались поля *КоличествоПриход* и *КоличествоРасход*. Для этого выделите поле *Отчет*, в нижнем окне перейдите на закладку *ВыбранныеПоля* и перетащите поля *КоличествоПриход* и *КоличествоРасход*.

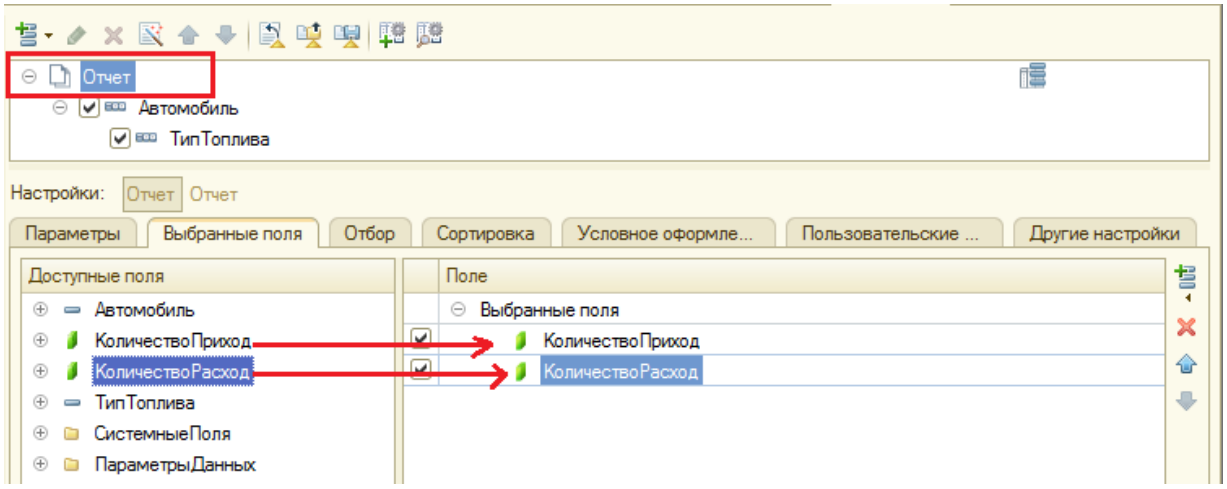


Рис. 11.2.42

Все. Сохраните отчет и запустите его в «1С:Предприятии».

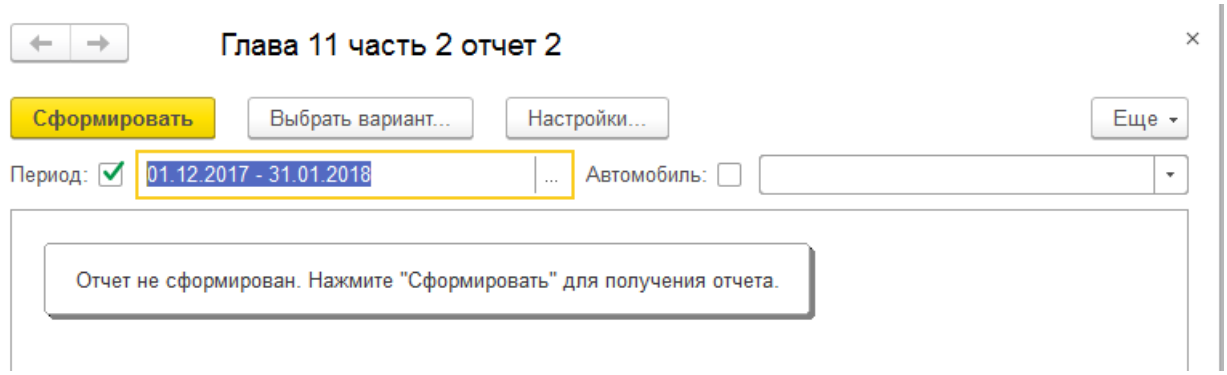


Рис. 11.2.43

Первым делом обратите внимание, что у нас на форме есть поля «Период» и «Автомобиль», так получилось потому, что мы установили флажки «Включать в пользовательские настройки». Причем у поля *Период* уже установлен флаг (значит, данные будут браться), а у поля *Автомобиль* – нет. Это потому, что мы поставили (или убрали) флаг у соответствующего поля (см. рис. 11.2.44).

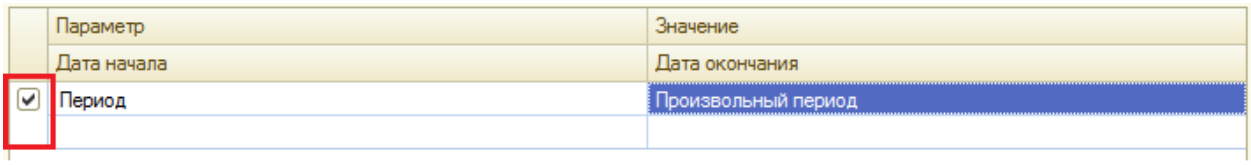


Рис. 11.2.44

Сформируем отчет.

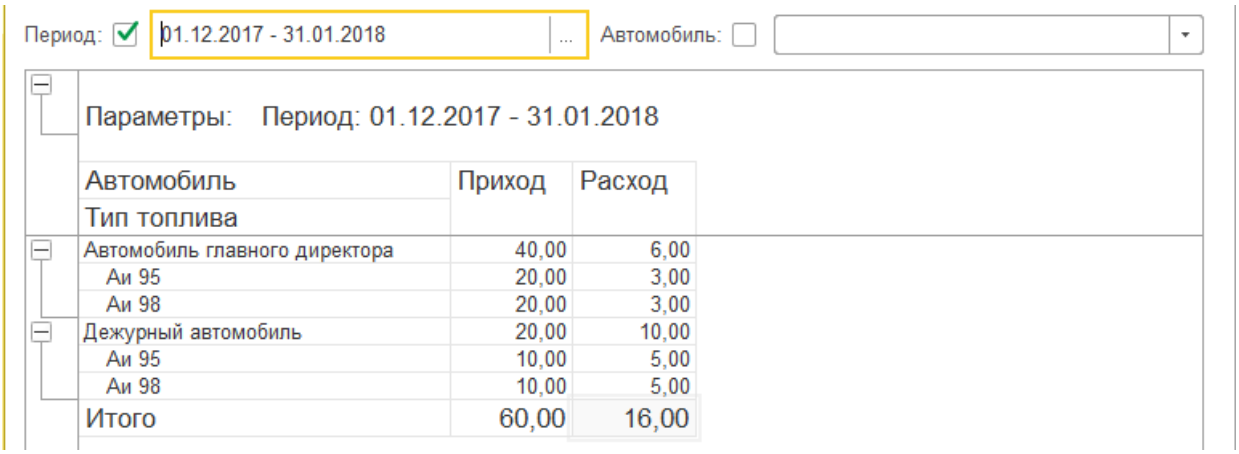


Рис. 11.2.45

Сформируем отчет с отбором по автомобилю.

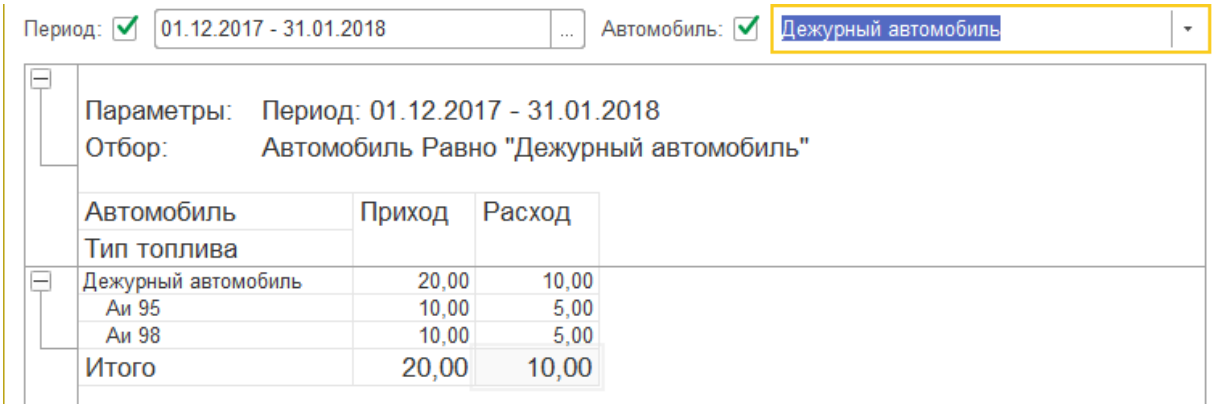


Рис. 11.2.46

На этом мы закончим изучать СКД. Понятно, что этой информации мало, чтобы полностью овладеть этим интересным механизмом. Но её достаточно, чтобы начать делать простые отчеты. В дальнейшем Вы сможете углубить свои знания при помощи различной литературы, статей и т.д. по этой теме.

Часть 3. Динамические списки

Рассмотрим последний способ вывода информации, который мы изучим в этой книге, - динамические списки.

Что такое динамический список? Динамический список - это специальный тип, при помощи которого на форме можно отображать любые произвольные данные. Он позволяет делать сортировку этих данных, отбор, группировку, условное оформление и т.д.

Разработчик может указать какую-то определенную таблицу, из которой будут браться необходимые данные, или самостоятельно написать нужный запрос.

Когда мы работали с запросами, то мы запросы загружали в таблицу значений на форме. Можно эти лишние движения не делать, а напрямую выводить данные из того же запроса в динамический список.

Что интересно, мы, сами того не зная, уже сталкивались с динамическими списками, когда создавали формы списков справочников или документов. При автоматическом создании формы списка, будь то справочника или документа, платформа всегда размещает динамический список на форме.

Откройте форму списка документа *Прибытие в гараж*, и Вы увидите, что у нас в реквизитах размещен динамический список (кстати, это основной реквизит).

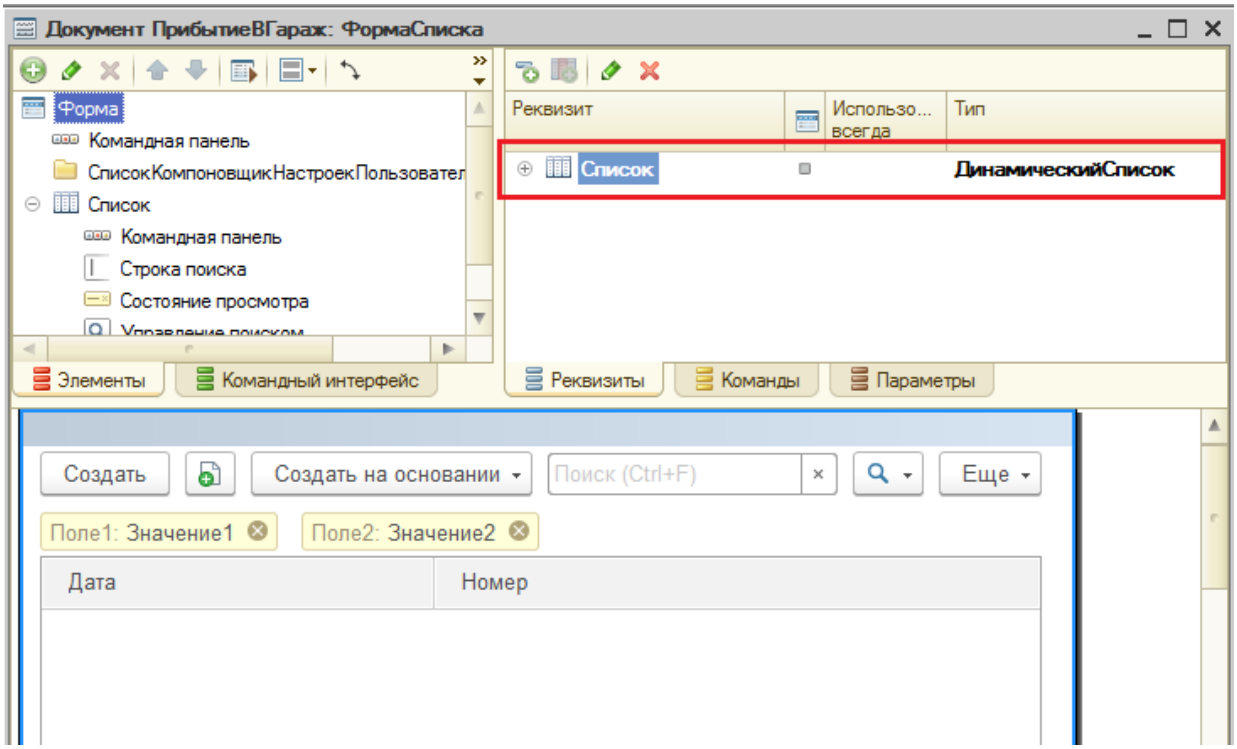


Рис. 11.3.1

Зайдем в палитру свойств этого реквизита.

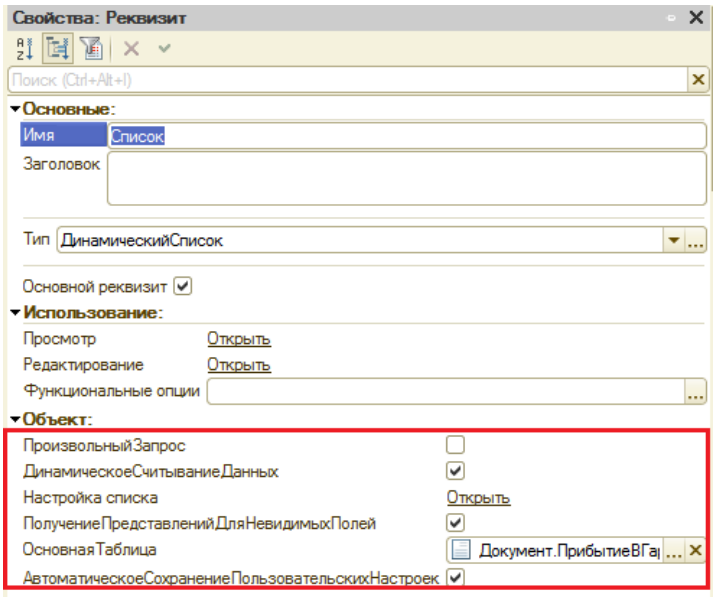


Рис. 11.3.2

Сейчас у нас установлено свойство *ОсновнаяТаблица* и снят признак *ПроизвольныйЗапрос*. Это значит, что при выводе списка платформа будет автоматически заполнять его данными из указанной таблицы. Если же мы поставим флаг *ПроизвольныйЗапрос*, то необходимо будет написать запрос к базе данных на языке запросов 1С, по которому будут выводиться данные в таблицу.

В качестве основной таблицы можно выбрать любую таблицу любого объекта, который есть в базе. Для этого достаточно нажать кнопку выбора (кнопка «…»), и откроется форма выбора таблицы (см. рис. 11.3.3).

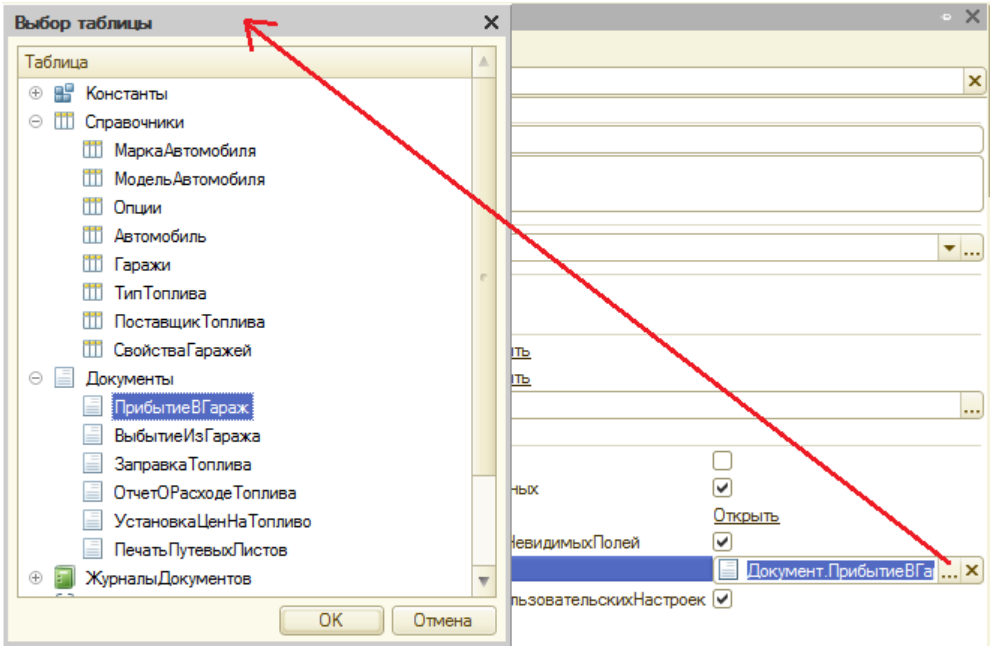


Рис. 11.3.3

Таким образом, в качестве основной таблицы Вы можете выбрать любую таблицу, которая есть в базе.

Если же установить флаг *Произвольный запрос*, то свойство *Основная таблица* уберется из списка свойств (см. рис. 11.3.4).

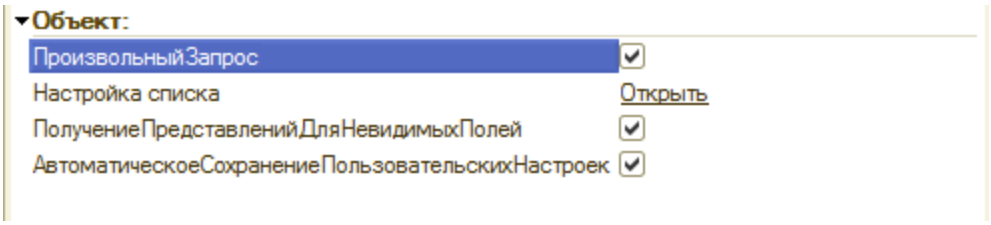


Рис. 11.3.4

Но если мы нажмем гиперссылку свойства *Настройка списка*, то откроется форма, в которой мы увидим окно запроса и опять свойство *Основная таблица* (см. рис. 11.3.5).

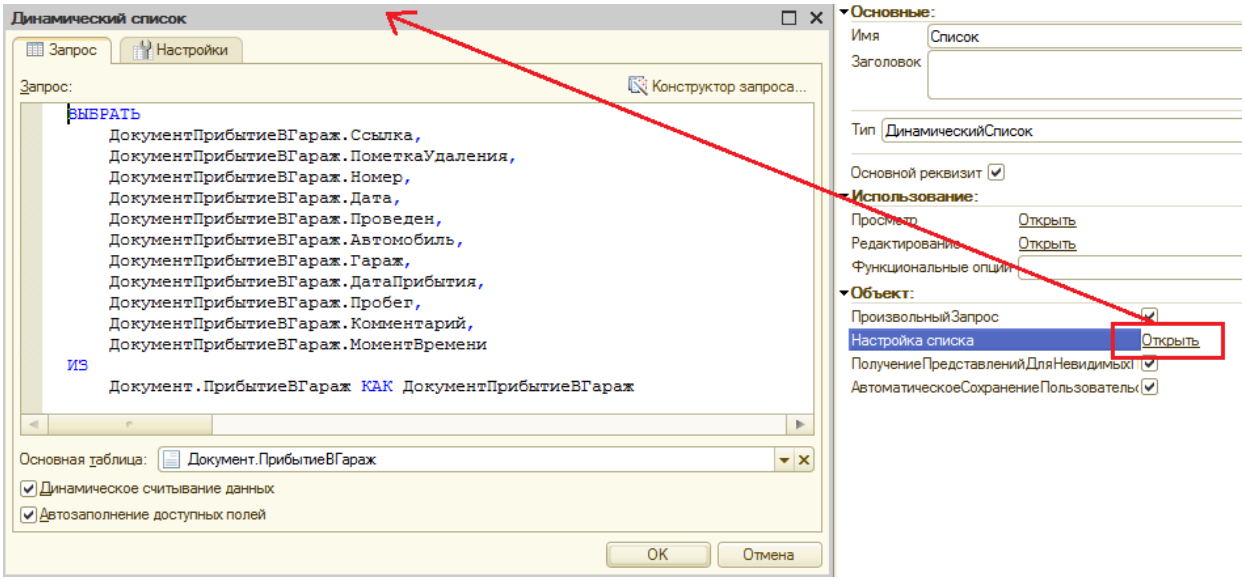


Рис. 11.3.5

В случае произвольного запроса мы можем изменить набор данных, который должен отображаться в динамическом списке. Но в этом случае все равно рекомендуется указывать основную таблицу, необходимо это делать для того, что бы платформа могла определить, какая информация в этом динамическом списке главная, а какая второстепенная, и исходя из этого настроить стандартные команды. В том случае, если у Вас несколько таблиц и они связаны каким-то соединением, то все равно рекомендуется всегда указывать основную таблицу, потому что в противном случае производительность основного списка будет желать лучшего.

Обратите внимание на свойства «Динамическое считывание данных», которое есть и в свойствах динамического списка, когда указана основная таблица (рис. 11.3.2), и в том случае, когда указан произвольный запрос (см. рис. 11.3.5). Если это свойство установлено, то считывание данных происходит порциями, т.е. выбираются только те данные, которые необходимы для отображения на экране. В целях улучшения производительности лучше всегда это свойство устанавливать.

Добавим в имеющийся динамический список поля *ГосНомер* и *ГодВыпуска* реквизита *Автомобиль* (сделайте это сами, воспользовавшись кнопкой «Конструктора запроса»).

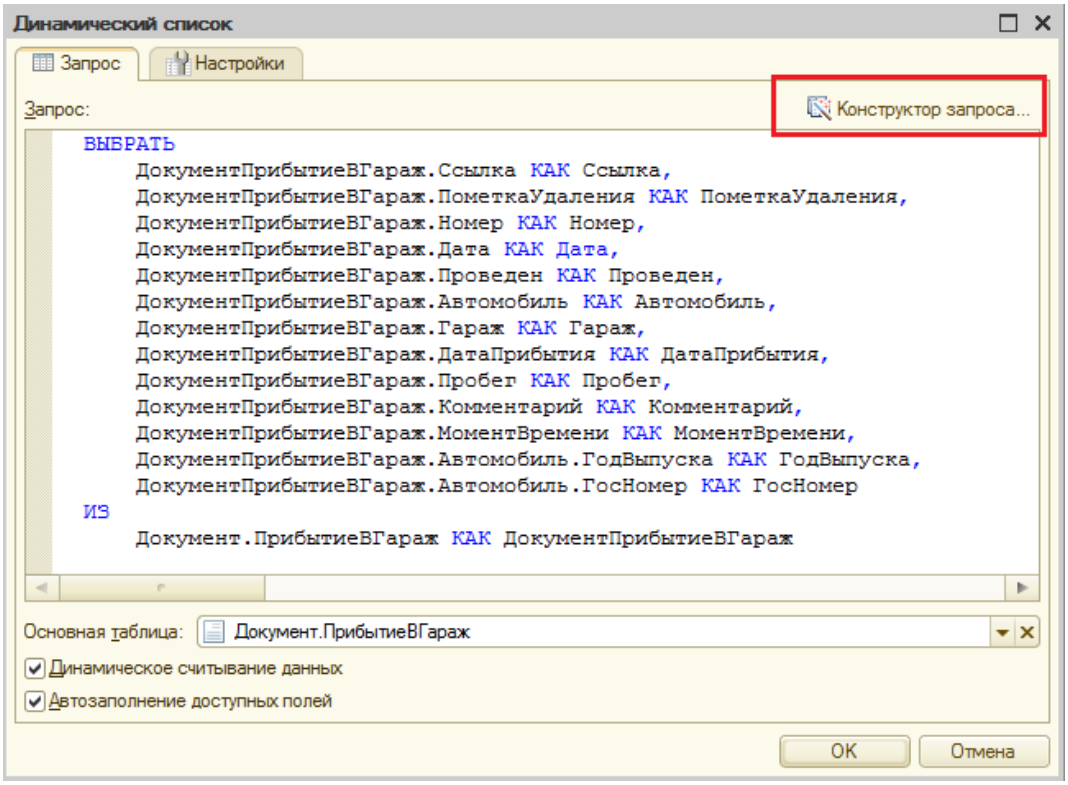


Рис. 11.3.6

Сохраним эту настройку списка, обновим конфигурацию и посмотрим, как отобразится список документов *ПрибытиеВГараж* (удалите имеющийся элемент «Список» и добавьте новый, расположите колонки, как Вам больше нравится).

Номер	Дата	Автомобиль	Год выпуска	Гос номер	Гараж	Дата прибытия	Ко
000000001	04.12.2017 12:00:00	Дежурный автом...	10.05.2015	кк123ка18	Гараж №1-1	04.12.2017 0:00:00	
000000002	06.12.2017 12:00:00	Автомобиль глав...	01.01.2016	х001кк18RU	Гараж №1-1	06.12.2017 0:00:00	
000000008	10.12.2017 0:00:00	Дежурный автом...	10.05.2015	кк123ка18	Гараж №1-2	12.12.2017 0:00:00	
000000007	12.12.2017 0:00:00	Дежурный автом...	10.05.2015	кк123ка18	Гараж №1-1	12.12.2017 0:00:00	
000000005	12.12.2017 12:53:24	Дежурный автом...	10.05.2015	кк123ка18	Гараж №1-1	12.12.2017 0:00:00	
000000006	15.12.2017 12:54:57	Дежурный автом...	10.05.2015	кк123ка18	Гараж №1-1	15.12.2017 9:26:04	

Рис. 11.3.7

Подробно о работе с динамическим списком читайте в книге «Основы разработки в 1С: Такси. Разработка в управляемом приложении за 12 шагов».

А теперь попробуем использовать динамический список как замену таблицы значений в наших тренировках с запросами. Для этого сделаем обработку, в которой создадим динамический список, выводящий список автомобилей.

Создайте новую обработку и форму обработки, а на форме создайте реквизит *Список* с типом *ДинамическийСписок*.

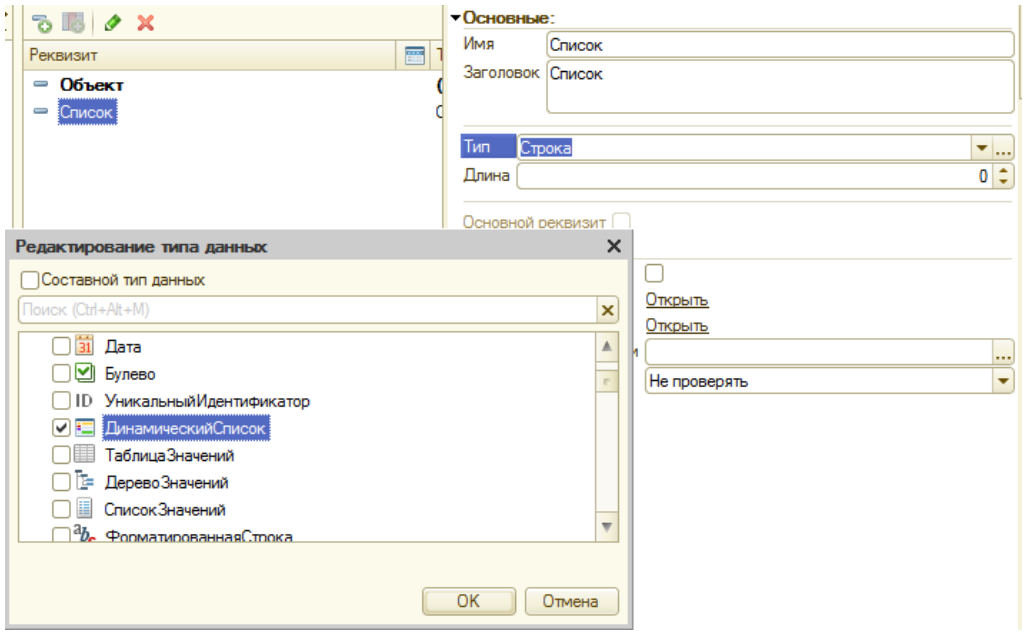


Рис. 11.3.8

Установим свойство *ПроизвольныйЗапрос* и в настройках списка сделаем запрос по таблице «Автомобили», не забыв указать её в качестве основной таблицы и не забыв поставить флаг «Динамическое считывание данных».

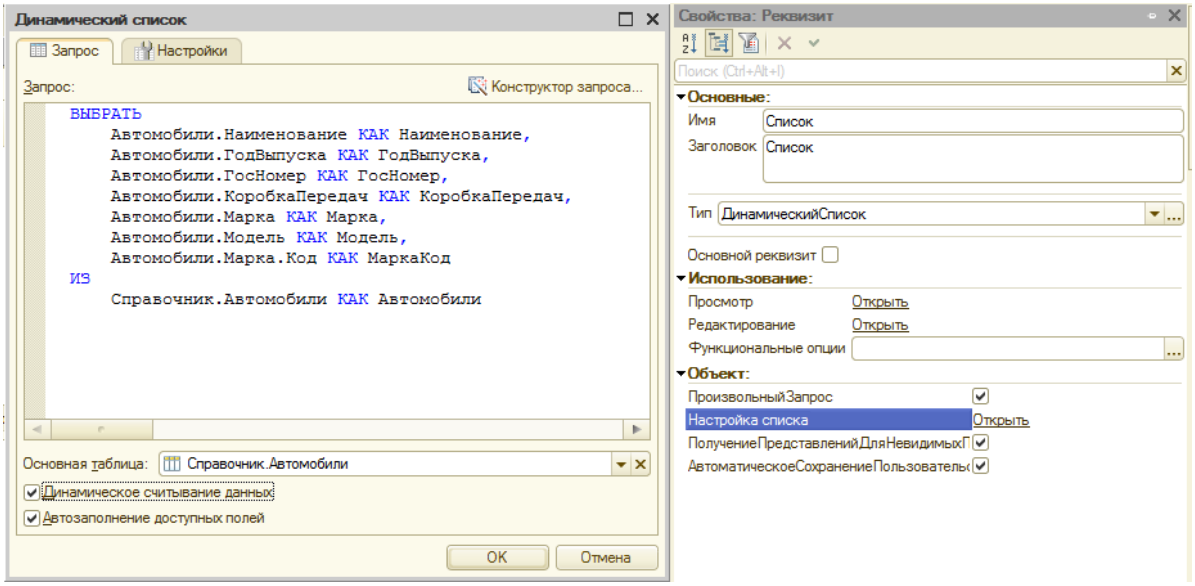


Рис. 11.3.9

Сохраним эту настройку, и у нас сразу же нужные поля появятся в списке.

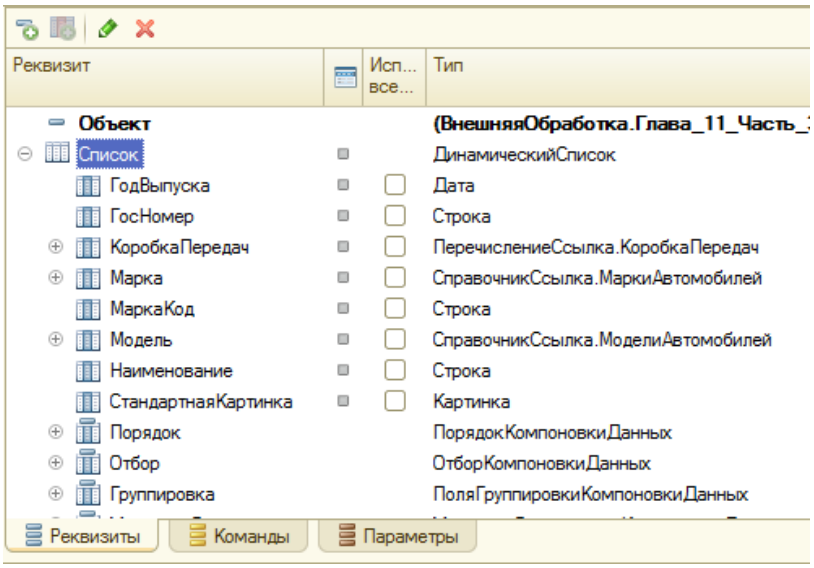


Рис. 11.3.10

Перетащим динамический список на форму, он отобразится в виде элемента *Таблица* (самостоятельно сделайте для Вас удобное расположение полей).

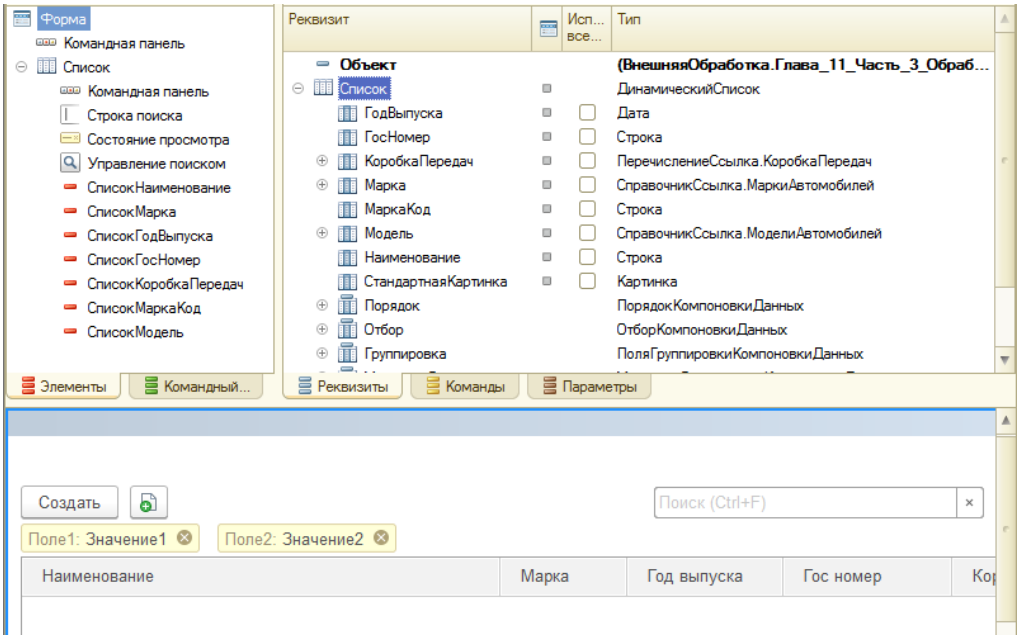


Рис. 11.3.11

Сохраним обработку и откроем её в «1С:Предприятии».

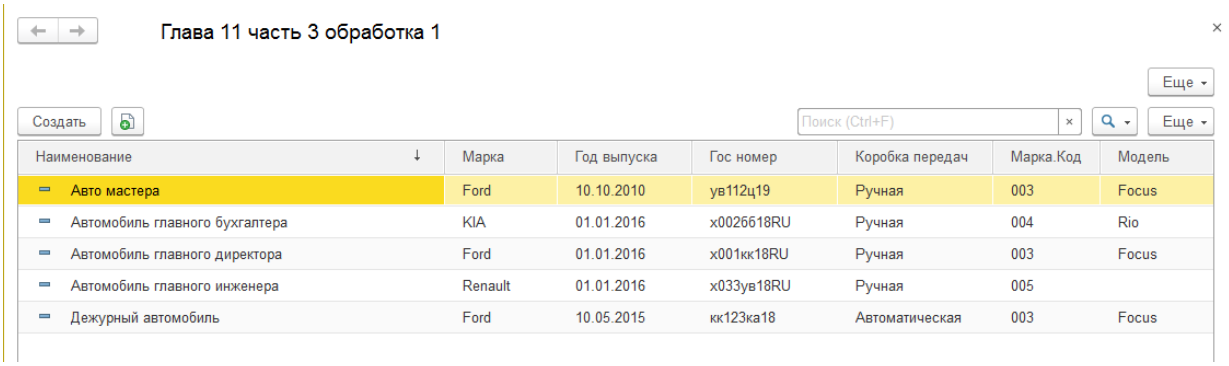


Рис. 11.3.12

Как видите, выводить данные при помощи динамических списков гораздо удобнее.

На этом мы закончим изучать работу с динамическими списками. Эта часть носит больше ознакомительный характер, чтобы Вы знали, что такой инструмент имеется и его очень удобно использовать. Больше информации о работе с динамическими списками есть в книге [«Основы разработки в 1С: Такси. Разработка в управляемом приложении за 12 шагов»](#).

Резюме

На этом мы закончим изучать основные механизмы вывода информации на экран в системе «1С:Предприятия». Их довольно много, и мы разобрали только основные из них. Вся информация этой главы носит ознакомительный характер, цель которой дать Вам первоначальное представление о различных способах отображения информации. В дальнейшем в процессе работы (учебы) Вы сможете углубить Ваши знания.

Заключение

Поздравляю! Вы дочитали до конца эту книгу. Если Вы осилили весь материал, прорешали все примеры и задачи, то у Вас есть отличные шансы начать успешно программировать в среде 1С. Но главное - не останавливаться на достигнутом: постоянно двигайтесь вперед, постоянно получайте новые для Вас знания и умения. И постоянно занимайтесь программированием и разработкой. Если Вы после прочтения этой книги не будете применять эти знания на практике, то всё, что Вы изучили, забудется через полгода.

И я очень надеюсь, что материал этой книги пригодится Вам в практическом применении, что Вы станете программировать, возможно, устроитесь работать программистом 1С (пусть даже стажером) и начнете свой профессиональный рост.

У Вас всё получится!