# Занятие 4. Особенности применения программного кода в системе.

### §4.1. Встроенный язык системы

До сих пор при разработке приложения мы использовали лишь стандартные механизмы «1С: Предприятия», без использования программного кода. Система многие события обрабатывает самостоятельно путем уже имеющихся механизмов. Тем не менее, существуют моменты, когда стандартного поведения системы оказывается недостаточно для реализации требуемой функциональности.

В нашем случае явным примером является добавление элементов в справочник

«Номенклатура». Вы наверняка заметили, что для каждой номенклатурной позиции нам необходимо явно указывать вид номенклатуры. А это может показаться нелогичным: мы и так находимся в определенной группе товаров, так почему система не может это проанализировать и самостоятельно заполнить вид номенклатуры? Стандартного механизма для такого анализа у системы нет. Для того чтобы систему «научить» выполнять такой анализ, необходимо использовать встроенный язык системы.

Встроенный язык системы является некоторой смесью множества известных вам языков программирования, включая объектно-ориентированное программирование. Главное отличие заключается в том, что весь программный код пишется на русском языке. Возможно использование англоязычных аналогов операторов, но, в большинстве случаев, программисты все же пишут в «1С» на русском языке.

*Переменные, операторы*

Объявление переменной: **Перем <*имя\_переменной*>;**

Присвоение переменной значения: **<*имя\_переменной*>** = <***значение***>;

Отметим, что в системе нет предопределенного задания типа и жесткой типизации переменных. Тип определяется по первому операнду в выражении, после чего производится попытка автоматического приведения остальных операндов к данному типу. Для объявления переменных, содержащих структурированные данные (коллекции значений, например, массив) используется более сложная конструкция - оператор **Новый**. Например, для объявления массива: <***имя\_переменной***> = **Новый Массив**;

Запись даты: **Сегодня = ‘20140221122045’** (12:25:45 21 февраля 2014 года). Можно написать: **Сегодня = ‘2014-02-21 12:25:45’** – все незначащие символы (в данном случае не цифры) игнорируется т.к. в одинарных кавычках пишется только дата. Если не указывать время, то оно будет равно 00:00:00. Есть понятие **ПустаяДата** – 00:00:00 на 01:01:01

Дата хранится в виде числа секунд от 00:00:00 на 01:01:01. Соответственно с датой можно производить арифметические операции сложения, вычитания для получения другой даты. Можно из одной даты вычесть другую и получить разницу в секундах.

Строки всегда записываются в двойных кавычках. Если строка очень длинная, к примеру, текст запроса, тогда имеется возможность переноса при помощи вертикальных разделителей:

**A = “Строка1**

**| Строка2**

**| Строка3**

**….**

**| СтрокаN”;**

Для объединения нескольких строк используется конкатенация символом «+».

Числа записываются как обычно, без всяких спецсимволов (10, 12.5). Основные операции: сложение, умножение, вычитание, деление, % (остаток от деления нацело).

Переменные булевого типа принимают два значения: Истина или Ложь.

Все операторы отделяются друг от друга точкой с запятой. Все незначащие символы (пробелы, переносы строк и т.д.) – просто игнорируются. Например, запись **A = 10%4** будет эквивалентна

## A =

**10**

**%4;**

Комментарии обозначаются символом // в начале каждой строки комментария.

*Стандартные алгоритмические конструкции*

1. Условный оператор:

Если (Условие) Тогда

[ИначеЕсли (Условие) Тогда]

….

[Иначе]

….

КонецЕсли;

1. Оператор цикла с параметром:

Для Каждого <***переменная***> Из <***СписокПеременных***> Цикл

….

КонецЦикла;

1. Оператор цикла с условием:

Пока <***условие***> Цикл

….

КонецЦикла;

1. Процедуры и функции:

Процедура ИмяПроцедуры ([Параметр1, Параметр2,…])

….

КонецПроцедуры;

Вызов процедуры: ИмяПроцедуры (***значение1***, ***значение2***);

По умолчанию передача параметров происходит по ссылке, т.е. значения внутри процедуры, функции могут изменяться. Для передачи параметров по значению (для сохранения значения при выполнении), перед именем параметра указывается ключевое слово **Знач**.

Для вызова процедур, функций в других модулях (а не там, где они описаны) используется ключевое слово **Экспорт**:

Процедура ИмяПроцедуры ([Параметр1, Параметр2,…]) Экспорт

Отличие функции от процедуры – в конце присутствует ключевое слово **Возврат** для передачи управления в точку вызова. Результат работы функции должен быть записан в некоторую переменную.

1. Булевы операции: возвращают результат «Истина» или «Ложь». В результате операций сравнения переменная получает значение типа булево: например, переменная **A** в результате операции **A = 10<3** примет значение «Истина». Для описания операции отрицания используется конструкция «НЕ»: **B = НЕ А**. Есть операторы логических связок «И», «ИЛИ», «НЕ». Пишутся такие операторы подряд: **A = 10 = 10 ИЛИ 3<>5**. Если необходимо сформировать приоритеты операций используются круглые скобки.

### §4.2. Виды программных модулей

Система «1С: Предприятие» является событийной. Т.е. все, что происходит, не происходит просто так, а лишь при наступлении определенных событий. Например, пользователь нажимает кнопку создания элемента справочника, и система реагирует на это действие – возникает событие и производится его обработка. Поэтому при использовании встроенного языка программный код не пишется линейно, чтобы некоторые действия выполнялись по заданному алгоритму: любой программный код привязывается к конкретному событию системы, связанному либо с объектами (включая элементы интерфейса), либо с поведением системы в целом. В зависимости от видов событий в системе «1С: Предприятие» выделяется ряд программных модулей, каждый из которых отвечает за обработку тех или иных событий и имеет определенное предназначение. Выделяются следующие виды программных модулей.

***Модуль формы***. Здесь происходит обработка событий, связанная с элементами формы (интерфейсом приложения). Форма – основной объект для интерактивного отображения данных. К примеру, когда мы нажимаем кнопку создания нового элемента справочника, открывается форма элемента для объекта типа «Справочник» со своим набором данных, свойств, методов. Модуль формы связан с интерактивными действиями пользователя (ввод данных в поля формы, выбор данных, и т.д.). Тип формы зависит от отображаемых данных. Существуют различные виды форм, связанные с одним и тем же объектом: форма элемента, форма списка, форма выбора, а также собственные разработанные программистом формы. Соответственно, каждая форма будет иметь свой перечень элементов и свой модуль, содержащий обработчики событий для данной формы.

***Модуль объекта***. В модуле объекта происходит обработка событий, связанная с объектом в целом, вне зависимости от формы. Такие события обычно связаны с записью информации в таблицы базы данных и не зависят от интерактивных действий пользователя. В отличие от модуля формы, модуль объекта содержит обработчики, реагирующие на любые программные обращения к объекту. Если рассматривать последовательность обработки событий, то вначале происходит обработка событий,

возникающих в какой-либо форме объекта (интерактивные события), а потом уже обработка событий объекта (программные события).

*Примечание*. Модуль объекта отсутствует у констант, перечислений, регистров. У регистров вместо модуля объекта присутствует ***модуль набора записей***.

***Модуль менеджера***. Появился только в версии 8.2. Очень похож на модуль объекта, но модуль объекта содержит реализацию методов и обработчиков событий, относящихся к конкретному экземпляру объекта, а модуль менеджера содержит методы, которые расширяют/переопределяют методы, относящиеся к всему набору экземпляров объекта, без привязки к конкретному экземпляру. Модуль менеджера есть у всех прикладных объектов.

***Модуль управляемого приложения***. Содержит обработчики событий, возникающих при старте системы в режиме «Управляемого приложения» (начинается с версии 8.2). Также **может** содержать процедуры и функции, вызываемые в других частях приложения, хотя для этого, преимущественно все же используются другие модули.

***Модуль сеанса***. Содержит обработчики событий, связанные с приложением в целом, вне зависимости от его типа (обычное или управляемое). Происходит установка параметров сеанса до старта системы.

***Общие модули***. Содержат экспортные процедуры и функции, доступные и вызываемые из всех других модулей приложения. Очень близки к понятию «внешние модули» (UNITS) в классических языках программирования.

### §4.3. Работа приложения в управляемом режиме. Управляемые формы.

Теперь пришло время обратиться к тому, как система устроена изнутри, как она получает данные, обрабатывает их и сохраняет.

Начиная с версии 8.2 основной режим работы «1С: Предприятие» – это работа в режиме «Управляемого приложения» с использованием управляемых форм. Управляемые формы – это такие формы, которые разработчиком не создаются самостоятельно «с нуля», как это было раньше (в предыдущих версиях системы). Система сама, на основании типа объекта и типа формы уже «знает», как форма должна выглядеть, как функционировать, какие события у нее должны быть и какие обработчики должны быть связаны с этими событиями. От программиста единственное что требуется – это указать системе, какие элементы на форме расположить и где. Остальное система все сделает сама. И только в том случае, если стандартное поведение системы не будет устраивать, только тогда программист может изменять, добавлять требуемую функциональность.

Внутри, приложение, разработанное на базе «1С: Предприятия», функционирует по модели «клиент-сервер». «Клиент» содержит все интерактивные элементы управления (кнопки, меню, поля ввода и т.д.), но сам по себе не содержит никаких данных. Все данные находятся на «сервере» - в таблицах базы данных: непосредственно с клиента мы не можем обратиться к ним. И только при специальных серверных вызовах эти данные с сервера поступают к клиенту.

Форма (отображающая те или иные данные), как программный объект, достаточно уникальна. Она одновременно существует и на «клиенте», и на «сервере». Если быть более точным, то на сервере она создается (программная копия), после чего передается клиенту и отображается (интерактивная копия, визуальная проекция). У формы есть реквизиты, имеющие тип «Данные формы». Один из реквизитов является основным (у формы лишь один реквизит может быть основным) и связан с таблицами базы данных. При серверном вызове происходит:

* 1. формирование специального SQL-запроса к таблицам базы данных;
  2. извлечение требуемых данных;
  3. преобразование данных из формата базы данных к типу «Данные формы»;
  4. помещение данных в соответствующие реквизиты основного реквизита формы. Кроме основного, форма также может иметь произвольное число других реквизитов. Данные реквизиты (не основной) не связаны с таблицами базы данных, а содержащаяся в них информация, формируется в результате работы различных программных алгоритмов при вызове или интерактивной работе с формой. При закрытии формы информация, хранящаяся этих реквизитах, никуда не записывается, а пропадает (если не реализовано

какого-либо программного алгоритма по передаче указанных данных).

Для отображения содержимого реквизитов формы используются элементы формы, имеющие тип «Элементы формы». Элементы формы – это те интерактивные элементы (поля ввода, надписи, кнопочки, переключатели и т.д.), которые видит пользователь при интерактивной работе с формой и которыми он управляет.

Рассмотрим пример. Допустим, мы открываем уже созданный ранее элемент справочника. При этом формируется команда на открытие формы элемента данного справочника. На сервере создается форма, и извлекаются данные из таблиц базы данных (этап 1, рис. 19).

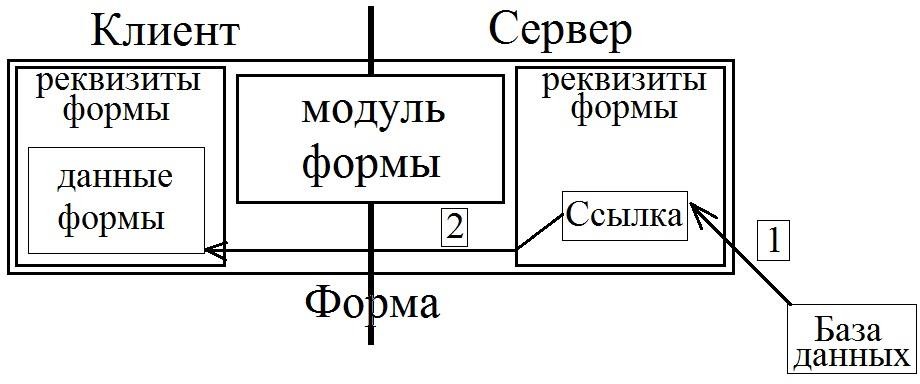


Рис. 19. Принцип открытия управляемой формы

Следует отметить, что при этом извлекаются не все данные, а лишь ссылка и ее представление (Код или Наименование). Ссылка нужна для того, чтобы впоследствии к этим данным можно было обратиться, а представление – для их вывода в интерфейсе. После этого форма «переезжает» на клиент; извлеченные данные преобразуются и помещаются в основной реквизит формы (Данные формы) (этап 2, рис. 24). Затем выполняется команда открытия формы, в результате которой элементы формы отображают содержимое реквизитов формы. Элементы формы связаны с данными формы через специальное свойство «Путь к данным».

Далее, допустим, в уже открытой на клиенте форме пользователь что-то изменил. Вопрос: эти изменения происходят где: на клиенте или на сервере? Конечно же, то, что пользователь изменил, т.е. ввел в интерактивные элементы формы, производится на клиенте. Сервер пока что «не знает», что на клиенте что-то поменялось. Однако если происходит серверный вызов, то измененные данные с клиента «переезжают» на сервер и происходит синхронизация данных на клиенте и на сервере.

Если же при работе с формой, пользователь на клиенте запросил какие-то данные с сервера, то происходит следующее. Реквизиты формы на клиенте содержат измененные данные из интерактивных элементов (рис. 20).

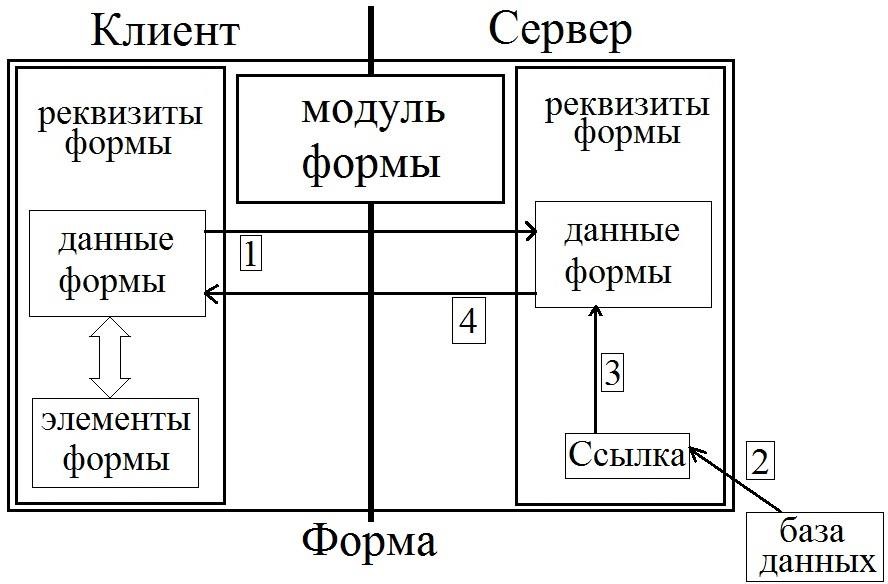


Рис. 20. Работа управляемой формы при серверном вызове

Измененная копия формы «переезжает» на сервер и происходит синхронизация с реквизитами формы на сервере (этап 1, рис. 20). Далее выполняется требуемый серверный вызов, например, запрос к базе данных (этап 2, рис. 20). Из базы данных извлекается требуемая информация и помещается в реквизиты формы на сервере (этап 3, рис. 20). После чего форма обратно «переезжает» на клиент, и происходит снова синхронизация данных на клиенте и на сервере (этап 4, рис. 20). В самом конце измененные реквизиты отображаются в соответствующих элементах формы. При обращении к базе данных (например, при записи данных), данные формы преобразуются к формату таблиц базы данных при помощи специального метода **ДанныеФормыВЗначение()**, при помещении данных из базы данных в реквизиты формы происходит обратное преобразование при помощи метода **ЗначениеВДанныеФормы()**.

Модуль формы существует одновременно и на клиенте и на сервере. Для того чтобы система знала, к данным и методам, клиента или сервера, в данный момент следует обращаться, существует 4 директивы, предваряющие любые процедуры и функции, описанные в модуле формы:

* **&НаКлиенте**: доступны все элементы и реквизиты формы на стороне клиента, нет доступа к базе данных – необходим серверный вызов;
* **&НаСервере**: доступны все реквизиты формы на стороне сервера, есть доступ к базе данных;
* **&НаСервереБезКонтекста**: доступны только те реквизиты формы на стороне сервера, которые были явно переданы в качестве параметров вызова, есть доступ к базе данных;
* **&НаКлиентеНаСервереБезКонтекста**: доступны все элементы формы и те реквизиты формы, которые были явно переданы, есть доступ к базе данных;

Основное отличие в директивах &НаСервере и &НаСервереБезКонтекста – это объем передаваемых данных. При контекстном вызове передаются все данные и описание формы. При безконтекстном вызове передаются лишь необходимые данные, без описания. Это обеспечивает сокращение передаваемых по сети данных и, как следствие, увеличение быстродействия системы.

В процессе разработки управляемых форм чаще всего используются директивы &НаКлиенте и &НаСервереБезКонтекста.

### §4.4. Использование встроенного языка. Создание приветствия пользователя

Теперь, когда мы в общих чертах разобрали особенности функционирования системы, попробуем написать небольшой программный код и посмотреть, как он будет работать.

Самое элементарное, что можно попытаться сделать – это поприветствовать пользователя, запустившего систему. При запуске системы происходит выполнение действий, описанных в модуле управляемого приложения. Т.к. мы будем разрабатывать только управляемые приложения (обычных приложений и обычных форм мы касаться не будем), то это как раз то, что нам нужно.

Для того чтобы открыть модуль управляемого приложения необходимо вызвать контекстное меню конфигурации и выбрать соответствующий пункт. В результате откроется пустой текст модуля. Как было отмечено ранее, система является событийной. Соответственно, необходимо описать события, которые и будут выполняться в данном модуле. Для того, чтобы посмотреть список всех событий и выбрать нужное, используется специальная кнопка на панели инструментов (выделена красным, рис. 21).

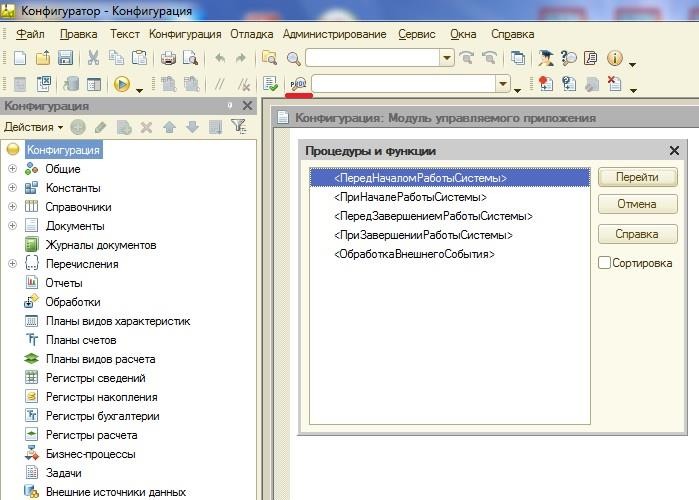


Рис. 21. События модуля управляемого приложения

Т.к. мы хотим выводить приветствие при загрузке разработанного решения, то нам необходимо событие «При начале работы системы». В результате выбора сформируется шаблон процедуры, которую необходимо заполнить. В настоящий момент процедура пустая. Но это не значит, что данное событие никак не обрабатывается. Система производит обработку по умолчанию. В том случае, если обработчик события будет

изменен (добавлен программный текст), тогда обработка по умолчанию будет замещена созданной обработкой. Если в дальнейшем, созданная обработка будет удалена, или закомментирована, тогда система снова будет производить обработку по умолчанию.

Добавим в обработчик события следующий текст:

|  |  |
| --- | --- |
| Сообщить (“Сегодня ”+ТекущаяДата()); | Л.1 |

Данная строка будет выводить сообщение с текущей датой в отдельном окне. ТекущаяДата() – это функция получения текущей системной даты. Обратите внимание, что здесь мы использовали конкатенацию строк, и система автоматические производит преобразование даты к строке. Попробуйте поменять дату и текстовую строку местами и посмотрите на результат. Для проверки правильности синтаксиса написанного программного кода используется кнопка , расположенная рядом с кнопкой вызова списка всех возможных событий.



Запустим систему в режиме отладки и посмотрим на результат.

Немножко усложним задачу. Кроме приветствия пользователя, покажем ему список тех сотрудников из соответствующего справочника, у кого сегодня день рождения. Данные по всем сотрудникам хранятся в базе данных, т.е. «на сервере». Вспомним, что модуль управляемого приложения исполняется на стороне клиента. Соответственно, непосредственно в модуле управляемого приложения мы не можем получить доступ к данным – нам необходимо делать специальный серверный вызов для получения требуемой информации.

Для решения поставленной задачи самый лучший способ – это создать общий модуль, который будет исполняться на сервере. В этом модуле следует описать функцию, получающую список именинников. А в модуле управляемого приложения, соответственно, сделать вызов данной функции (серверный вызов) т.к. общие модули являются местом описания процедур и функций, доступных из любой точки программы.

Создадим общий модуль (раскроем в дереве конфигурации ветвь «Общие» и найдем соответствующий класс объектов) и назовем его «**Общие механизмы**» (рис. 22).

Для любого общего модуля обязательным является указание: глобальный он или нет (настройка «Глобальный», рис. 22). Процедуры и функции глобальных общих модулей доступны в любой части приложения и вызываются просто по имени. При этом имена процедур и функций, описанных в разных глобальных общих модулях не должны пересекаться. В неглобальных общих модулях процедуры и функции могут иметь одинаковые имена, а для их вызова вначале указывается имя общего модуля, а затем через точку имя вызываемой процедуры/функции.

Затем галочками необходимо отметить, где выполняются процедуры и функции данного модуля: на клиенте, на сервере или и там и там (рис. 22). Если отметить что-то одно, то перед именами процедур/функций не нужно будет указывать предваряющую их директиву. Если же отметить обе галочки (клиент, сервер), то перед каждой процедурой/функцией необходимо будет располагать директиву, указывающую, где данная процедура/функция выполняется.

Процедуры/функции исполняемые на клиенте доступны для вызова из клиентских модулей, на сервере – соответственно из серверных. Однако если на клиенте необходимо сделать вызов серверной процедуры/функции, описанной в общем модуле, то в настройке модуля необходимо отметить галочку «Вызов сервера» (рис. 22).

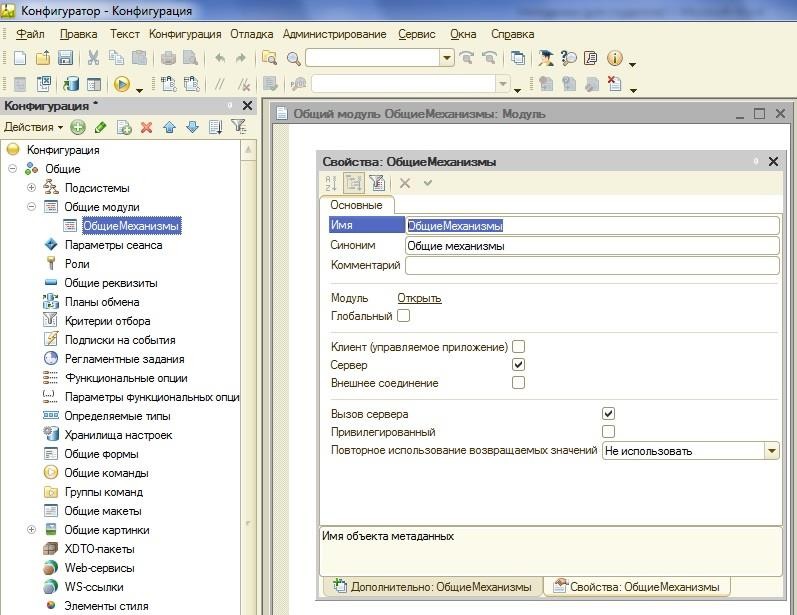


Рис. 22. Создание общего модуля

Общий модуль не содержит никаких событий – только те процедуры/функции, которые будут вызываться в других частях приложения. Соответственно, их компиляция производится только в момент вызова. Опишем в общем модуле следующую функцию:

|  |  |
| --- | --- |
| Функция ПолучитьСписокИменинников () Экспорт ТД\_День = День(ТекущаяДата());  ТД\_Месяц = Месяц(ТекущаяДата()); Мас = Новый Массив;  Выборка = Справочники.Сотрудники.Выбрать(); Пока Выборка.Следующий() Цикл  ДР\_День = День(Выборка.ДатаРождения); ДР\_Месяц = Месяц(Выборка.ДатаРождения);  Если ДР\_День = ТД\_День И ДР\_Месяц = ТД\_Месяц Тогда Мас.Добавить(Выборка.Наименование);  КонецЕсли; КонецЦикла;  Возврат Мас; КонецФункции | Л.2 |

При описании процедуры/функции, даже если она не содержит принимаемых параметров, все равно указываются открывающая и закрывающая скобки. Также обратите внимание: после имени функции идет ключевое слово «Экспорт». Оно необходимо для того, чтобы данная процедура/функция была видна в других частях приложения. Согласитесь, бывает необходимость в создании служебных процедур/функций, которые

будут вызываться другими процедурами/функциями только этого модуля для каких-то промежуточных действий.

Поясним написанный текст. Первые две строчки содержат функции получения дня и месяца из текущей даты. Далее создается массив, который будет заполняться списком имен тех сотрудников, у кого сегодня день рождения. Следующая строчка формирует выборку всех элементов справочника «Сотрудники» и помещает их в переменную типа

«Выборка данных». Переменная данного типа содержит ссылки на все элементы выбранных данных. Далее следует цикл по всем элементам сформированной выборки. Для перехода к следующему элементу используется специальный метод ***Следующий()***, который получает ссылку на следующий элемент выборки. Соответственно, по данной ссылке мы можем обратиться ко всем реквизитам справочника. Так мы получаем день и месяц рождения сотрудника (очередного элемент выборки). Затем проверяем: если день и месяц рождения сотрудника совпадает с текущим днем и месяцем, тогда в описанный ранее массив добавляем имя сотрудника (реквизит «Наименование»). После чего переходим к следующему элементу выборки, и т.д. пока метод ***Следующий()*** не вернет признак конца выборки т.е. значение NULL. В самом конце – функция в качестве возвращаемого значения передает весь сформированный массив.

Теперь в модуле управляемого приложения необходимо получить сформированный массив и вывести его в информационное окно. Для этого изначально мы обращаемся к описанной функции соответствующего общего **неглобального** модуля. Затем формируем цикл обхода всех элементов полученного массива, в котором выводим очередной элемент на экран при помощи функции ***Сообщить()***. Соответствующий программный код будет следующим:

|  |  |
| --- | --- |
| СписокИменниников = ОбщиеМеханизмы.ПолучитьСписокИменинников(); Для Каждого Именинник ИЗ СписокИменниников Цикл  Сообщить("Сегодня День рождения у "+Именинник); КонецЦикла; | Л.3 |

Запустим систему в режиме отладки и проверим работу механизма (при необходимости для ряда сотрудников укажем в качестве даты рождения текущую дату). Для большего понимания написанного программного кода, вам следует самостоятельно обратиться к соответствующим разделам Синтакс-помощника.

### §4.5. Использование отладки. Работа с данными заполнения

Теперь после первоначального знакомства со встроенным языком, можно вернуться к первоначальной задаче, из-за которой мы и стали рассматривать доработку стандартных механизмов платформы «1С: Предприятия».

Напомним задачу: как сделать так, что система автоматически заполняла реквизит

«**Вид номенклатуры**» на основании того, в какой номенклатурной группе происходит создание очередного элемента справочника?

Первое, что необходимо сделать, это реализовать возможность заполнять реквизит

«**Вид номенклатуры**» не только у отдельных элементов, но и у групп элементов. Соответственно, при создании очередного элемента внутри группы, данный реквизит будет «наследоваться» от «Родителя». Для этого перейдем к настройке реквизита «**Вид номенклатуры**» справочника «**Номенклатура**», и на вкладке «Использование» укажем

«Использование для группы и элемента». Запустим приложение в режиме отладки, перейдем к справочнику Номенклатура и откроем любую имеющуюся группу номенклатурных позиций (допустим, «**Товары**») в режиме редактирования (используем соответствующую пункт меню в группе команд «Еще») и заполним данный реквизит.

Для всех групп внутри выбранной группы не произойдет автоматического заполнения указанного реквизита т.к. данные группы были созданы ранее, а в системе отсутствует

соответствующий механизм. Поэтому на данном этапе вручную отредактируем уже имеющиеся группы товаров, и заполним данный реквизит соответствующими значениями. Затем попытаемся в любой номенклатурной группе создать новую позицию. Мы увидим, что реквизит «**Вид номенклатуры**» все равно не заполняется автоматически т.к.

стандартный механизм «не знает» как это делать.

Нашей целью является заполнение соответствующего реквизита при создании нового элемента. При этом нам неважно, какая форма будет открываться при его добавлении. Также обратим внимание на то, что при создании элемента мы обращаемся к данным группы, в которой этот элемент создается, т.е. фактически к тем данным, которые уже хранятся в базе данных, т.е. «на сервере». Из всего этого можно сделать вывод, что требуемый программный код не имеет смысла располагать в модуле формы т.к. она связана с интерактивной работой пользователя. Соответственно, требуемую обработку имеет смысл поместить в модуль объекта справочника «**Номенклатура**».

Открыть модуль объекта можно двумя способами:

* Открыть окно редактирования объекта, перейти на вкладку «Прочее» и нажать на кнопку «Модуль объекта»;
* Открыть контекстное меню объекта в дереве конфигурации и выбрать соответствующий пункт меню;

Откроем модуль объекта и посмотрим список событий, связанных с данным объектом (аналогично тому, как мы это делали при работе с модулем управляемого приложения). Среди всех событий нас будет интересовать событие ***ОбработкаЗаполнения()***. Данное событие возникает при создании новых экземпляров объекта (не при редактировании существующих). В этот момент реквизиты вновь создаваемого объекта заполняются данными из специального набора реквизитов, называемого «ДанныеЗаполнения». Соответственно, для того, чтобы объект получал требуемые значения, необходимо добавить в «ДанныеЗаполнения» требуемую информацию.

Перед тем, как добавить информацию в «ДанныеЗаполнения», необходимо определить

* какой тип этих данных и что они уже содержат. Для этого воспользуемся еще одним необходимым инструментом разработчика – отладкой. Для этого в любом месте обработчика события ***ОбработкаЗаполнения()*** добавим текст, к примеру, ***A = 1;***. После чего обновим конфигурацию базы данных, и затем при помощи контекстного меню установим в данной части процедуры точку останова (рис. 23).

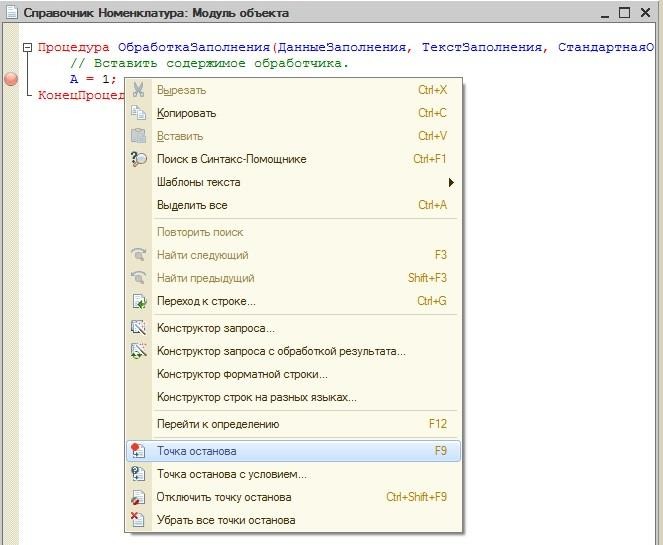


Рис. 23. Добавление точки останова для отладки приложения

Запустим приложение в режиме отладки. Откроем справочник «**Номенклатура**» и попытаемся создать новый элемент. При этом система автоматически перебрасывает нас в

«Конфигуратор», в то самое место, где мы поставили точку останова. На панели инструментов становятся доступны инструменты отладки (рис. 24). Соответствующие инструменты также доступны через контекстное меню в модуле и через меню «Отладка».



Рис. 24. Панель отладки

В процессе отладки возможна установка новых точек останова без перезапуска приложения, движение по точкам останова, переход к следующим строчкам кода и т.д. В наибольшей степени нас будут интересовать следующие инструменты (рис. 24):

* + «**вычислить выражение**» - используется для определения значения различных объектов в текущий момент выполнения программы (можно выделить непосредственно объект, переменную и система покажет данные выделенного объекта); можно определять получает ли программа в нужный момент времени

доступ к определенным данным, а также, какие данные содержатся в тех или иных объектах, их корректность и т.д.;

* + «**табло**» - служит для добавления данных, которые могут меняться в процессе работы программы, тем самым имеется возможность отслеживать правильность выполнения кода, заполнения реквизитов, выполнения расчетов и т.д.;
  + «**шагнуть через**» - переход к следующей строчке программного кода и ее выполнение;
  + «**шагнуть в**» и «**шагнуть из**» связаны с переходом в/из некоторого цикла, содержащего программный код;
  + «**продолжить отладку**» - вернуться в режим исполнения для дальнейших действий пользователя;

Выделим переменную «ДанныеЗаполнения» (в параметрах обработчика события) и нажмем инструмент «Вычислить выражение». В результате появится окно (рис. 25).

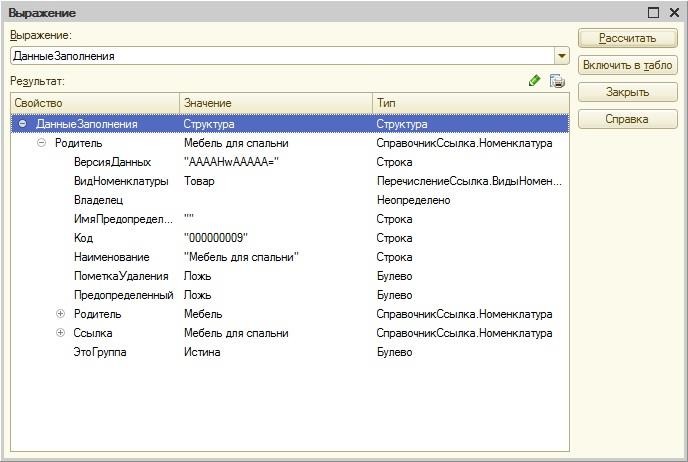


Рис. 25. Окно «Вычислить выражение» в режиме отладки

На основе данных, имеющихся в этом окне можно сделать ряд важных выводов:

* Переменная «ДанныеЗаполнения» имеет особый тип – Структура;
* в «ДанныеЗаполнения» передается лишь значение поля «Родитель»;

Поле «Родитель» - это реквизит ссылочного типа, соответственно, имеется возможность его «раскрыть» и подробно посмотреть его содержимое (значения всех реквизитов). В том случае, если в оке «Вычислить выражение» оказываются не только реквизиты ссылочного типа, а, табличные части, то для просмотра содержимого табличной части необходимо ее выделить, после чего нажать кнопку «Показать значение в отдельном окне» (рис. 25). Для удобства, аналогичные действия можно выполнить с любыми данными в окне «Вычислить выражение».



Закроем окно «Вычислить выражение» и завершим отладку (через соответствующий пункт меню «Отладка»).

Самый простой способ решить поставленную задачу – это принудительное заполнение значения реквизита «Вид номенклатуры» таким же значением, как и у группы, в которой данный элемент создается, т.е. у «Родителя». У Родителя значение реквизита ВидНоменклатуры можно получить из «ДанныхЗаполнения» (рис. 25). Добавим в обработчик следующий код:

|  |  |
| --- | --- |
| ВидНоменклатуры = ДанныеЗаполнения.Родитель.ВидНоменклатуры; | Л.4 |

Проверим, что точка останова находится там же, где и раньше (рис. 23). Запустим систему в режиме отладки. В момент остановки выполнения, запустим инструмент

«Вычислить выражение», в поле «Выражение» введем имя реквизита, значение которого мы хотим увидеть (**ВидНоменклатуры**), и нажмем кнопку «Рассчитать» (см. рис. 25). В результате увидим, что значение реквизита заполнилось верными данными.

Этот код достаточно прост и правильно работает. А теперь попытаемся создать новый элемент справочника в его корне (там, где у нас имеются предопределенные группы

«**Товары**» и «**Услуги**»). Возникающая ошибка (рис. 26) связана с тем, что у элементов, находящихся в корне справочника поле «Родитель» не заполнено, а «ДанныеЗаполнения» не определены.

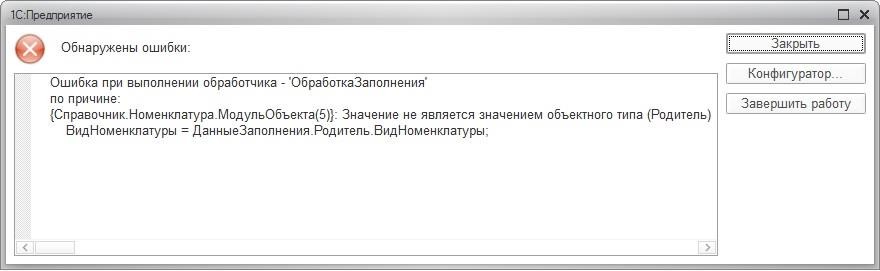


Рис. 26. Ошибка при создании элемента справочника в его корне

Для исключения данной ошибки перед заполнением реквизита «**Вид номенклатуры**» необходимо проверить – заполнена структура «**ДанныеЗаполнения**» или нет. Для этого выше добавим следующий код:

|  |  |
| --- | --- |
| Если ДанныеЗаполнения = Неопределено Тогда Возврат;  КонецЕсли; | Л.5 |

Напоминаем, что НЕОПРЕДЕЛЕНО означает, что значение принципиально может существовать, но не заполнено в данный момент. «**Возврат**» означает прерывание работы процедуры/функции в которой указан данный метод, в данном случае процедура ***ОбработкаЗаполнения()*** прерывается, и мы продолжаем работу с системой. Запустите систему в режиме отладки и проверьте работоспособность. Обратите внимание, что процедура действительно прерывается т.к. мы не доходим до точки останова (мы ее пока что не отключали).

На текущем этапе мы разобрались с тем, что же такое «ДанныеЗаполнения», для чего они нужны. Но пока что мы никак не воспользовались информацией о том, что это Структура. На самом деле, «ДанныеЗаполнения» могут содержать то, что в них будет передано. И возможно возникновение ситуации, когда «ДанныеЗаполнения» не будут

иметь тип «Неопределено», т.е. в них будут переданы какие-то данные, но среди этих данных будет отсутствовать поле «**Родитель**». Такое возможно при различных программных обращениях к справочнику «**Номенклатура**», когда отключается стандартное поведение системы и разработчик меняет функциональность по своему усмотрению.

Соответственно, задача дополнительно сводится к тому, чтобы проверить – имеется ли в «ДанныхЗаполнения» поле «**Родитель**». Для решения данной задачи вначале следует обратиться к Синтакс-помощнику и определить, что представляет собой тип данных

«Структура» (мы его найдем среди «Универсальных коллекций значений»). Открыв соответствующий раздел, мы увидим, что структура – это набор «ключ - значение». В качестве ключей используются имена полей (реквизитов), в нашем случае – это поле

«Родитель». В качестве значения – значения конкретных реквизитов. Таким образом, при создании нового элемента, структура «ДанныеЗаполнения» должна, к примеру, выглядеть следующим образом:

ДанныеЗаполнения = («Родитель», Мебель для спальни)

Понятно, что «**Мебель для спальни**» - это представление. Реально же там содержится

ссылка в виде хэш-строки. Также в справке по типу данных «Структура» мы видим, что к значениям, хранящимся в ней, можно обращаться по имени ключа, что мы и неосознанно сделали, когда написали самый первый программный код по заполнению реквизита «**Вид номенклатуры**». Также объекты типа «Структура» имеют ряд методов, в частности, метод «Свойство», который определяет, присутствует ли заданный ключ в структуре или нет, что требуется для проверки того, что в «ДанныеЗаполнения» попадает поле

«**Родитель**» со всеми его реквизитами. Поэтому после проверки на наличие

«ДанныеЗаполнения» добавим следующую проверку перед заполнением реквизита «**Вид номенклатуры**»:

|  |  |
| --- | --- |
| Если ДанныеЗаполнения.Свойство("Родитель") Тогда  ВидНоменклатуры = ДанныеЗаполнения.Родитель.ВидНоменклатуры; КонецЕсли; | Л.6 |

Запустим систему в режиме отладки и посмотрим, что все наши действия привели к желаемому результату. Если точка останова все еще имеется, то ее можно удалить вместе с содержащимся в ней кодом, после чего обновить конфигурацию базы данных.

### Темы для самостоятельного изучения:

1. Обычное приложение и обычные формы: отличия от современной концепции управляемого приложения.
2. Типы данных для управления прикладными объектами (<Объект>Менеджер,

<Объект>Объект, <Объект>Выборка, <Объект>Ссылка и т.д.).

1. Универсальные коллекции значений.
2. Создание обработчика для события «Обработка Проверки Заполнения».