ПРОФЕССИОНАЛЬНАЯ РАЗРАБОТКА

Е.Ю. Хрусталева

ТЕХНОЛОГИИ ИНТЕГРАЦИИ 1С:ПРЕДПРИЯТИЯ 8.3

1С ПАБЛИШИНГ

Хрусталева Е.Ю.

Технологии интеграции 1С:Предприятия 8.3

Электронная книга в формате pdf; ISBN 978-5-9677-2964-5.

Книга адресована специалистам, имеющим опыт разработки на платформе «1С:Предприятие». С ее помощью можно освоить механизмы «1С:Предприятия 8.3», предназначенные для обмена данными в распределенных системах, а также механизмы и технологии, позволяющие интегрировать прикладные решения с другими информационными системами, не использующими «1С:Предприятие».

В период активного перехода на удаленную работу особую ценность приобретают доработки, которые реализуют функции автоматического взаимодействия прикладного решения с внешними системами и ресурсами. Например, важным ресурсом являются Интернет-сайты, поскольку основной инструмент потенциальных клиентов в этот период – браузер. Если, заходя на сайт, клиенты смогут, например, самостоятельно размещать заказы в программе 1С или получать из нее некоторую информацию, это может значительно облегчить взаимодействие с ними без использования офисной телефонной связи. В этой книге как раз подробно рассматриваются все ключевые технологии, которые есть в системе «1С:Предприятие» для интеграции с различными внешними системами.

По сравнению с предыдущим изданием «Технологии интеграции "1С:Предприятия 8.2"») книга дополнена описанием интернет-технологий, которые появились в платформе 8.3 или не были описаны раньше:

- JSON;
- HTTP-сервисы (REST);
- HTTP-запросы;
- автоматический REST-интерфейс (OData);
- Web-сервисы;
- FTP-соединение;
- электронная почта.

Для создания демонстрационных примеров использована версия 8.3.16.1030 платформы «1С:Предприятие 8».

Книга выпущена под редакцией Максима Радченко.

Дополнительные материалы

Информационные базы с примерами, описанными в книге, опубликованы на портале 1С:ИТС. Вы можете скачать их по адресу http://its.1c.ru/book_demo/.

Интернет-конференция для начинающих разработчиков http://devtrainingforum.v8.1c.ru/forum.

Оглавление

Введение

В книге собрана и систематизирована наиболее важная информация, которая может понадобиться разработчику прикладных решений «1С:Предприятия 8.3».

Несмотря на то что в одной книге невозможно рассмотреть все ситуации, возникающие при разработке прикладных решений, на большинство вопросов в книге можно найти ответы. Причем читать будет одинаково интересно как начинающим, так и продвинутым разработчикам.

При написании мы стремились к тому, чтобы книга стала серьезным инструментом для разработчиков: к ней всегда можно было бы обратиться в случае затруднений, узнать что-то новое о хорошо известной предметной области или познакомиться с новым взглядом на привычные вещи.

При подготовке материала были использованы различные источники информации:

- опыт преподавания на учебных курсах по платформе и прикладным решениям «1С:Предприятия 8»;
- опыт внедрения прикладных решений;
- опыт, накопленный разработчиками фирмы «1С»;

- материалы информационно-технологической поддержки (ИТС);
- материалы форума партнеров-разработчиков на сайте http://partners.v8.1c.ru;
- общение на партнерских семинарах, проводимых фирмой «1С».

Глава 1. Интернет-технологии

JSON

В настоящее время в веб-приложениях широко используется формат обмена данными JSON. JSON (JavaScript Object Notation) – это текстовый формат обмена данными, с которым могут работать все браузеры. Этот формат похож на XML, но по сравнению с XML он является более лаконичным и требует меньше места.

Общая информация

Есть несколько причин для широкого использования этого формата на уровне платформы. Во-первых, JSON – это современный формат, с помощью которого прикладные решения «1С:Предприятия» могут осуществлять интеграцию со сторонними приложениями. Во-вторых, JSON активно используется в HTTP-интерфейсах, а платформа «1С:Предприятие 8» как раз предоставляет два способа реализации таких интерфейсов – это REST-интерфейс, который автоматически формируется для всего прикладного решения, и HTTP-сервисы, которые можно создавать самостоятельно.

Существует несколько основных сценариев использования JSON:

- Интеграция с внешними системами через их HTTP-интерфейсы: Google Calendar, Salesforce.com, REST-интерфейс «1С:Предприятия», SharePoint и т. д.
- Организация собственного HTTP-интерфейса прикладного решения.
- Обмен файлами JSON с внешними системами. Формирование конфигурационных, настроечных файлов. Использование их в процедурах обмена данными, например с интернет-магазинами.
- Использование файлов JSON для обмена данными между разными приложениями «1С:Предприятия».

JSON – это текстовый формат, поэтому данные в формате JSON могут содержать:

- *Объект* – неупорядоченное множество пар <имя свойства>:<значение>, заключенный в фигурные скобки ({}). Пары <имя свойства>:<значение> разделяются запятыми (,).
- *Массив* – множество значений. Массив заключается в квадратные скобки ([]). Значения разделяются запятыми (,).
- *Значение* – может быть строкой, числом, объектом, массивом или литералом `true`, `false`, `null`.
  - `Строка` – набор символов, заключенный в двойные кавычки («»).
  - `Число` – сериализуется с разделителем точка (.). Точность числа не ограничена.

Таким образом, с помощью вышеперечисленных элементов допускается описание объектов любой сложности для представления в формате JSON.

В платформе реализовано несколько слоев работы с JSON. Самые универсальные и гибкие – это низкоуровневые средства потоковой записи и чтения. Более высокоуровневые и не такие универсальные – средства сериализации в JSON примитивных типов и коллекций «1С:Предприятия». И, наконец, третий слой – это средства, позволяющие сериализовать/десериализовать прикладные типы «1С:Предприятия»: ссылки, объекты, наборы записей и вообще любые типы, для которых поддерживается XDTO-сериализация (про XDTO-сериализацию рассказывается в разделе «XDTO-сериализация»).

ПОДРОБНЕЕ

Более подробно про работу с форматом данных JSON рассказано в главе 1 «Интернет-технологии».

Подробнее познакомиться с примерами сериализации/десериализации данных в формате JSON можно в демонстрационной конфигурации «Примеры работы», которая прилагается к книге.

Потоковая работа

Для потоковой работы предназначены объекты `ЧтениеJSON` и `ЗаписьJSON`. Они последовательно, от значения к значению, читают JSON из файла или строки или последовательно записывают JSON в файл или строку. Таким образом, чтение и запись JSON происходят без формирования всего документа в памяти. В результате потоковая техника может дать существенный выигрыш, особенно если объем документа потенциально большой или же если этот объем заранее неизвестен.

Запись и чтение

Для того чтобы выполнить потоковую запись JSON-документа, необходимы записываемые данные и объект `ЗаписьJSON`.

Допустим, нам нужно записать в JSON-документ некоторый объект, обладающий свойствами Код, Наименование, Телефоны (массив строк), ОбъемПродаж, Поставщик?.

Один JSON-файл содержит одно значение. Это то самое значение, о котором говорилось выше: объект, массив, строка, число или один из литералов. Поэтому записываем в JSON наш объект следующим образом (листинг 1.1).

Листинг 1.1. Пример потоковой записи JSON-документа

```
&НаСервереБезКонтекста
Процедура ПотоковаяЗаписьНаСервере()

    // Создать объект записи и открыть файл, в который будет выполняться запись.
    Запись = Новый ЗаписьJSON;
    Запись.ОткрытьФайл("c:\temp\streamWrite.json", , , Новый ПараметрыЗаписиJSON(, Символы.Таб));

    // Выполнить запись значений с помощью объекта записи (Запись).
    // Записать начало нашего объекта.
    Запись.ЗаписатьНачалоОбъекта();

    // Заполнить свойство Код типа Строка.
    Запись.ЗаписатьИмяСвойства("Код");
    Запись.ЗаписатьЗначение("000000017");

    // Заполнить свойство Наименование типа Строка.
    Запись.ЗаписатьИмяСвойства("Наименование");
    Запись.ЗаписатьЗначение("ОАО Топаз");

    // Заполнить свойство Телефоны типа Массив.
```

```
        // Поэтому после имени свойства записываем массив, состоящий из значений – строк.
        Запись.ЗаписатьИмяСвойства("Телефоны");
        Запись.ЗаписатьНачалоМассива();
        Запись.ЗаписатьЗначение("8-999-777-55-33");
        Запись.ЗаписатьЗначение("+71112223344");
        Запись.ЗаписатьКонецМассива();

        // Заполнить свойство ОбъемПродаж типа Число.
        Запись.ЗаписатьИмяСвойства("ОбъемПродаж");
        Запись.ЗаписатьЗначение(5000000);

        // Заполнить свойство Поставщик? типа Булево.
        Запись.ЗаписатьИмяСвойства("Поставщик?");
        Запись.ЗаписатьЗначение(Ложь);

        // Записать конец нашего объекта.
        Запись.ЗаписатьКонецОбъекта();

        // Завершить работу с файлом.
        Запись.Закрыть();

КонецПроцедуры
```

Содержимое процедуры понятно из комментариев в тексте листинга. Поэтому не будем еще раз на этом останавливаться. Про параметры записи JSON, переданные четвертым параметром в метод `ОткрытьФайл()`, будет рассказано ниже, в разделе «Параметры записи JSON».

В результате сформированный JSON-документ будет иметь следующий вид (листинг 1.2).

Листинг 1.2. Содержимое JSON-документа

```
{
        "Код": "000000017",
        "Наименование": "ОАО Топаз",
        "Телефоны": [
                "8-999-777-55-33",
                "+71112223344"
        ],
        "ОбъемПродаж": 5000000,
        "Поставщик?": false
}
```

Теперь представим, что мы хотим записать не какой-то абстрактный объект, а объект, обладающий собственным именем – Контрагент1. Возможно, вместе с ним мы захотим записать и другой аналогичный объект с именем Контрагент2. Как поместить это в один JSON-файл?

Поскольку, как мы говорили, JSON-файл может содержать только одно значение, мы запишем в него корневой объект с единственным свойством.

Имя этого свойства – это имя нашего объекта, а значение этого свойства – это совокупность свойств, которыми обладает наш объект (листинг 1.3).

Листинг 1.3. Пример потоковой записи JSON-документа

```
&НаСервереБезКонтекста
Процедура ПотоковаяЗаписьНаСервере()

	// Создать объект записи и открыть файл, в который будет выполняться запись.
	Запись = Новый ЗаписьJSON;
	Запись.ОткрытьФайл("c:\temp\streamWrite_2.json", , , Новый ПараметрыЗаписиJSON(, Символы.Таб));

	// Выполнить запись значений с помощью объекта записи (Запись).
	// Записать начало корневого объекта.
	Запись.ЗаписатьНачалоОбъекта();

	// Записать имя свойства корневого объекта.
	Запись.ЗаписатьИмяСвойства("Контрагент1");

	// Записать начало нашего объекта.
	Запись.ЗаписатьНачалоОбъекта();

	// Заполнить свойство Код типа Строка.
	Запись.ЗаписатьИмяСвойства("Код");
	Запись.ЗаписатьЗначение("000000017");

	// Заполнить свойство Наименование типа Строка.
	Запись.ЗаписатьИмяСвойства("Наименование");
	Запись.ЗаписатьЗначение("ОАО Топаз");

	// Заполнить свойство Телефоны типа Массив.
	// Поэтому после имени свойтва записываем массив, состоящий из значений – строк.
	Запись.ЗаписатьИмяСвойства("Телефоны");
	Запись.ЗаписатьНачалоМассива();
	Запись.ЗаписатьЗначение("8-999-777-55-33");
	Запись.ЗаписатьЗначение("+71112223344");
	Запись.ЗаписатьКонецМассива();

	// Заполнить свойство ОбъемПродаж типа Число.
	Запись.ЗаписатьИмяСвойства("ОбъемПродаж");
	Запись.ЗаписатьЗначение(5000000);

	// Заполнить свойство Поставщик? типа Булево.
	Запись.ЗаписатьИмяСвойства("Поставщик?");
	Запись.ЗаписатьЗначение(Ложь);

	// Записать конец нашего объекта.
	Запись.ЗаписатьКонецОбъекта();

	// Записать конец корневого объекта.
	Запись.ЗаписатьКонецОбъекта();

	// Завершить работу с файлом.
	Запись.Закрыть();

КонецПроцедуры
```

В результате сформированный JSON-документ будет иметь следующий вид (листинг 1.4).

Листинг 1.4. Содержимое JSON-документа

```
{
	"Контрагент1": {
		"Код": "000000017",
		"Наименование": "ОАО Топаз",
		"Телефоны": [
			"8-999-777-55-33",
			"+71112223344"
		],
		"ОбъемПродаж": 5000000,
		"Поставщик?": false
	}
}
```

Если требуется записать в этот же файл второй объект Контрагент2, то нам нужно просто добавить к корневому объекту еще одно свойство (Контрагент2) и его значение (листинг 1.5).

Листинг 1.5. Пример потоковой записи JSON-документа

```
&НаСервереБезКонтекста
Процедура ПотоковаяЗаписьНаСервере()

	// Создать объект записи и открыть файл, в который будет выполняться запись.
	Запись = Новый ЗаписьJSON;
	Запись.ОткрытьФайл("c:\temp\streamWrite_3.json", , , Новый ПараметрыЗаписиJSON(, Символы.Таб));

	// Выполнить запись значений с помощью объекта записи (Запись).
	// Записать начало корневого объекта.
	Запись.ЗаписатьНачалоОбъекта();

	// Записать имя первого свойства корневого объекта.
	Запись.ЗаписатьИмяСвойства("Контрагент1");

	// Записать начало первого объекта.
	Запись.ЗаписатьНачалоОбъекта();

	...

	// Записать конец первого объекта.
	Запись.ЗаписатьКонецОбъекта();

	// Записать имя второго свойства корневого объекта.
	Запись.ЗаписатьИмяСвойства("Контрагент2");

	// Записать начало второго объекта.
	Запись.ЗаписатьНачалоОбъекта();

	...
```

```
    // Записать конец второго объекта.
    Запись.ЗаписатьКонецОбъекта();

    // Записать конец корневого объекта.
    Запись.ЗаписатьКонецОбъекта();

    // Завершить работу с файлом.
    Запись.Закрыть();

КонецПроцедуры
```

В результате сформированный JSON-документ будет иметь следующий вид (листинг 1.6).

Листинг 1.6. Содержимое JSON-документа

```
{
    "Контрагент1": {
        "Код": "000000017",
        "Наименование": "ОАО Топаз",
        "Телефоны": [
            "8-999-777-55-33",
            "+71112223344"
        ],
        "ОбъемПродаж": 5000000,
        "Поставщик?": false
    },
    "Контрагент2": {
        "Код": "000000018",
        "Наименование": "ОАО Алмаз",
        "Телефоны": [
            "8-111-222-33-44",
            "+78886664422"
        ],
        "ОбъемПродаж": 6000000,
        "Поставщик?": true
    }
}
```

Потоковое чтение JSON-документа выполняется с помощью объекта `ЧтениеJSON`. Документ открывается для чтения и поэлементно считывается в цикле. При этом разработчик определяет, что считано, и соответствующим образом обрабатывает считываемые данные.

Простейший вариант чтения JSON-документа может быть выполнен с помощью следующей процедуры (листинг 1.7).

Листинг 1.7. Пример потокового чтения JSON-документа

```
&НаСервереБезКонтекста
Процедура ПотоковоеЧтениеНаСервере()

	Сообщение = Новый СообщениеПользователю;

	// Создать объект чтения и открыть файл, из которого будет выполняться чтение.
	Чтение = Новый ЧтениеJSON;
	Чтение.ОткрытьФайл("c:\temp\streamWrite_2.json");

	// Выполнить чтение поэлементно в цикле.
	Пока Чтение.Прочитать() Цикл

		Если Чтение.ТипТекущегоЗначения = ТипЗначенияJSON.ИмяСвойства Тогда
			Сообщение.Текст = "Имя = " + Чтение.ТекущееЗначение;
			Сообщение.Сообщить();
		КонецЕсли;

		Если Чтение.ТипТекущегоЗначения = ТипЗначенияJSON.Булево Или
			Чтение.ТипТекущегоЗначения = ТипЗначенияJSON.Строка Или
			Чтение.ТипТекущегоЗначения = ТипЗначенияJSON.Число Или
			Чтение.ТипТекущегоЗначения = ТипЗначенияJSON.Комментарий Тогда
			Сообщение.Текст = "Значение = " + Чтение.ТекущееЗначение;
			Сообщение.Сообщить();
		КонецЕсли;
	КонецЦикла;

	// Завершить работу с файлом.
	Чтение.Закрыть();

КонецПроцедуры
```

Результат чтения JSON-документа, содержимое которого показано в листинге 1.4, будет выглядеть следующим образом (рис. 1.1).

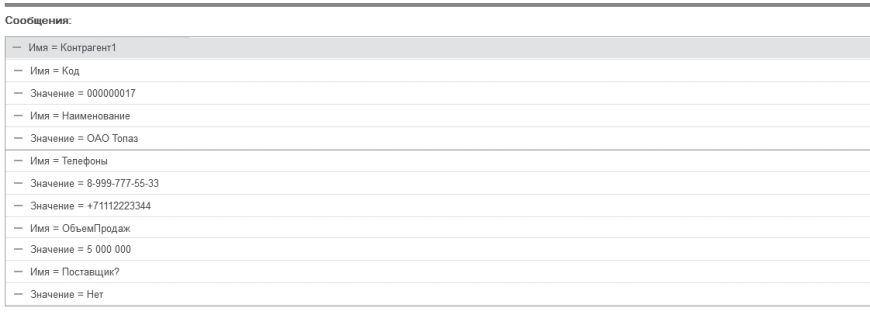


Рис. 1.1. Потоковое чтение JSON-документа

Работа со строкой JSON

Данные в формате JSON можно получать не только в виде файла, но и в виде строки. Например, при работе с HTTP-сервисами очень часто нужно работать с телом HTTP-запроса, которое представляет собой строку JSON. В таких случаях можно писать данные не в файл, а в строку. И потом подставлять эту строку в качестве тела HTTP-запроса. И, наоборот, брать строку, которая является телом HTTP-запроса, и читать из нее данные с помощью объекта `ЧтениеJSON`.

Для этого нужно использовать метод `УстановитьСтроку()` объектов `ЗаписьJSON`/`ЧтениеJSON`. После вызова этого метода для получения строки со сформированными данными JSON достаточно просто завершить запись документа методом `Закрыть()` объекта `ЗаписьJSON` (листинг 1.8).

Листинг 1.8. Запись и чтение содержимого JSON в/из строки

```
&НаСервереБезКонтекста
Процедура РаботаСоСтрокойНаСервере()

	Сообщение = Новый СообщениеПользователю;

	// Создать объект записи и записать строковое значение в строку JSON.
	Запись = Новый ЗаписьJSON;
	Запись.УстановитьСтроку();
	Запись.ЗаписатьЗначение("строковое значение");
	СтрокаJSON = Запись.Закрыть();

	// Показать результат.
	Сообщение.Текст = СтрокаJSON;
	Сообщение.Сообщить();

	// Создать объект чтения и прочитать JSON из строки.
	Чтение = Новый ЧтениеJSON;
	Чтение.УстановитьСтроку(СтрокаJSON);
	Пока Чтение.Прочитать() Цикл

		Если Чтение.ТипТекущегоЗначения = ТипЗначенияJSON.ИмяСвойства Тогда
			Сообщение.Текст = "Имя = " + Чтение.ТекущееЗначение;
			Сообщение.Сообщить();
		КонецЕсли;

		Если Чтение.ТипТекущегоЗначения = ТипЗначенияJSON.Булево Или
			Чтение.ТипТекущегоЗначения = ТипЗначенияJSON.Строка Или
			Чтение.ТипТекущегоЗначения = ТипЗначенияJSON.Число Или
			Чтение.ТипТекущегоЗначения = ТипЗначенияJSON.Комментарий Тогда
			Сообщение.Текст = "Значение = " + Чтение.ТекущееЗначение;
			Сообщение.Сообщить();
		КонецЕсли;
	КонецЦикла;

	// Завершить чтение.
	Чтение.Закрыть();

КонецПроцедуры
```

Проверка структуры записываемого JSON-документа

Следует иметь в виду, что при потоковой записи разработчик самостоятельно формирует структуру JSON-документа. При записи данных объект `ЗаписьJSON` автоматически проверяет правильность записываемой структуры. За это отвечает свойство `ПроверятьСтруктуру`. По умолчанию оно истинно, поэтому если при записи нарушена целостность структуры JSON, то будет сгенерировано исключение и запись выполнена не будет. Но для увеличения скорости работы эту проверку можно отключить (листинг 1.9).

Листинг 1.9. Отключение проверки структуры записываемого JSON-документа

```
ЗаписьJSON.ПроверятьСтруктуру = Ложь;
```

Параметры записи JSON

При записи можно управлять некоторыми параметрами формируемого текста, например: использованием двойных кавычек, переносом строк, символами отступа и экранированием символов. Набор параметров записи задается в объекте `ПараметрыЗаписиJSON` и затем используется при открытии для записи JSON-документа.

Перенос строк

По умолчанию при записи JSON используется автоматический перенос строк. Это удобно для визуального контроля получившегося результата. Но иногда те или иные HTTP-интерфейсы требуют, чтобы JSON был записан в одну строку. В таких случаях можно в параметрах записи JSON отключить перенос строк (листинг 1.10). Кроме того, отключение переноса строк полезно в тех случаях, когда нужно экономить объем передаваемых данных: если не переносить строки, объем уменьшится в среднем на 20 %.

Листинг 1.10. Отключение переноса строк записываемого JSON-документа

```
...
    // Задать параметры записи JSON.
    ПараметрыJSON = Новый ПараметрыЗаписиJSON(ПереносСтрокJSON.Нет, " ", Истина);

    // Создать объект записи и открыть файл, в который будет выполняться запись.
    Запись = Новый ЗаписьJSON;
    Запись.ОткрытьФайл("c:\temp\streamWrite_4.json", , , ПараметрыJSON);`
...
```

В результате сформированный JSON-документ будет иметь следующий вид (листинг 1.11).

Листинг 1.11. Содержимое JSON-документа

```
{"Контрагент1":{"Код":"000000017","Наименование":"ОАО Топаз","Телефоны":
                                ["8-999-777-55-33","+71112223344"],"ОбъемПродаж":5000000,"Поставщик?":false}}
```

Символы отступа

По умолчанию перед строками JSON не вставляется синтаксический отступ. Но для визуального контроля удобно, когда строки форматируются с помощью синтаксического отступа. Поэтому в примерах выше мы использовали для такого отступа символ табуляции (листинг 1.12).

Листинг 1.12. Набор параметров записи JSON-документа с синтаксическим отступом

```
ПараметрыJSON = Новый ПараметрыЗаписиJSON(, Символы.Таб);
Запись = Новый ЗаписьJSON;
Запись.ОткрытьФайл("c:\temp\streamWrite.json", , , ПараметрыJSON);
```

Если бы мы этого не сделали (листинг 1.13), результат выглядел бы следующим образом (листинг 1.14).

Листинг 1.13. Запись JSON-документа без синтаксического отступа

```
...
        Запись.ОткрытьФайл("c:\temp\streamWrite_5.json");
...
```

Листинг 1.14. Содержимое JSON-документа

```
{
"Контрагент1": {
"Код": "000000017",
"Наименование": "ОАО Топаз",
"Телефоны": [
"8-999-777-55-33",
"+71112223344"
],
"ОбъемПродаж": 5000000,
"Поставщик?": false
}
}
```

Экранирование символов

Внешние системы могут использовать разные нотации JSON, могут считать те или иные символы недопустимыми и так далее. Для этих целей у конструктора `ПараметрыЗаписиJSON` и у самого объекта есть возможность управлять экранированием таких символов, как амперсанд, одинарные

кавычки, разделители строк, слеш и угловые скобки. Кроме того, есть возможность для строковых значений использовать одинарные кавычки вместо двойных.

Например, прямой слеш при записи в JSON не экранируется по умолчанию (листинг 1.15).

Листинг 1.15. Запись строки JSON без экранирования символов

```
...
	Сообщение = Новый СообщениеПользователю;

	// Записать строковое значение в JSON.
	Запись = Новый ЗаписьJSON;
	Запись.УстановитьСтроку();
	Запись.ЗаписатьЗначение("путь к файлу C:/файл.txt");
	СтрокаJSON = Запись.Закрыть();

	// Показать результат.
	Сообщение.Текст = СтрокаJSON;

...
```

В результате выполнения приведенного выше фрагмента кода будет получена следующая строка (листинг 1.16).

Листинг 1.16. Результирующая строка без экранирования слеша

```
"путь к файлу C:/файл.txt"
```

Однако эту возможность можно включить (листинг 1.17).

Листинг 1.17. Запись строки JSON с экранированием символов

```
...
	Сообщение = Новый СообщениеПользователю;
	ПараметрыJSON = Новый ПараметрыЗаписиJSON( , , , , , , , ,Истина);

	// Записать строковое значение в JSON.
	Запись = Новый ЗаписьJSON;
	Запись.УстановитьСтроку(ПараметрыJSON);
	Запись.ЗаписатьЗначение("путь к файлу C:/файл.txt");
	СтрокаJSON = Запись.Закрыть();

	// Показать результат.
	Сообщение.Текст = СтрокаJSON;
	Сообщение.Сообщить();
...
```

В результате выполнения приведенного выше фрагмента кода будет получена следующая строка (листинг 1.18).

Листинг 1.18. Результирующая строка с экранированием слеша

```
"путь к файлу C:\/файл.txt"
```

Сериализация коллекций значений (объектная техника)

Во многих случаях (например, при обмене информацией с внешними системами, чтении конфигурационных файлов в формате JSON и др.) проще и удобнее использовать так называемую объектную технику работы с JSON с помощью методов глобального контекста `ПрочитатьJSON()`, `ЗаписатьJSON()` и объектов `ЧтениеJSON`, `ЗаписьJSON`.

Эта техника позволяет избежать рутинной работы по чтению/записи каждого отдельного значения или свойства. При чтении документы JSON отображаются в фиксированный набор типов платформы: `Строка`, `Число`, `Булево`, `Неопределено`, `Массив`, `ФиксированныйМассив`, `Структура`, `ФиксированнаяСтруктура`, `Соответствие`, `Дата`. При записи можно сформировать в памяти и быстро записать структуру в файл JSON.

Таким образом, объектная техника предполагает достаточно простую работу с данными, однако платой за это является расход памяти, так как весь JSON-документ обрабатывается целиком в оперативной памяти. Кроме того, так можно сериализовать только примитивные типы данных и коллекции значений. Прикладные типы данных так сериализовать не получится (о сериализации прикладных типов «1С:Предприятия» рассказывается в разделе «Сериализация прикладных типов "1С:Предприятия"»).

Запись и чтение

Для того чтобы выполнить запись коллекции значений в формате JSON, нужен сам объект низкоуровневой записи `ЗаписьJSON`, собственно, записываемая структура данных. И затем все вышеперечисленные объекты нужно передать в качестве параметров в метод глобального контекста `ЗаписатьJSON()`.

Рассмотрим следующий пример записи в JSON-документ структуры, состоящей из примитивных типов данных (листинг 1.19).

Листинг 1.19. Пример сериализации структуры в JSON-документ

```
&НаСервереБезКонтекста
Процедура СериализацияПростыхТиповНаСервере()

    // Создать структуру с данными контрагента.
    Данные = Новый Структура;
    Данные.Вставить("Контрагент", "ОАО Топаз");
    Данные.Вставить("ОбъемПродаж", 5000000);
```

```
    // Добавить элемент структуры Телефоны типа Массив.
    Телефоны = Новый Массив;
    Телефоны.Добавить("+71112223344");
    Телефоны.Добавить("+79998887766");
    Данные.Вставить("Телефоны", Телефоны);
    Данные.Вставить("Поставщик", Ложь);

    // Создать объект записи и открыть файл, в который будет выполняться запись.
    Запись = Новый ЗаписьJSON;
    ПараметрыЗаписиJSON = Новый ПараметрыЗаписиJSON(ПереносСтрокJSON.Авто, Символы.Таб);
    Запись.ОткрытьФайл("c:\temp\Serialisation.json",,, ПараметрыЗаписиJSON);

    // Выполнить запись данных (Данные) с помощью объекта записи (Запись).
    ЗаписатьJSON(Запись, Данные);

    // Завершить работу с файлом.
    Запись.Закрыть();

КонецПроцедуры
```

В результате сформированный JSON-документ будет иметь следующий вид (листинг 1.20).

Листинг 1.20. Содержимое JSON-документа

```
{
    "Контрагент": "ОАО Топаз",
    "ОбъемПродаж": 5000000,
    "Телефоны": [
        "+71112223344",
        "+79998887766"
    ],
    "Поставщик": false
}
```

Чтение данных в объектной технике выглядит аналогично записи. Рассмотрим пример чтения JSON-документа, содержимое которого показано выше (листинг 1.21).

Листинг 1.21. Пример десериализации структуры из JSON-документа

```
&НаСервереБезКонтекста
Процедура ДесериализацияПростыхТиповНаСервере()

    // Создать объект чтения и открыть файл, из которого будет выполняться чтение.
    Чтение = Новый ЧтениеJSON;
    Чтение.ОткрытьФайл("c:\temp\Serialisation.json");

    // Выполнить чтение данных в структуру Данные с помощью объекта чтения (Чтение).
    Данные = ПрочитатьJSON(Чтение);
```

```
        // Завершить работу с файлом.
        Чтение.Закрыть();

        // Вывести результат чтения в сообщение.
        Сообщение = Новый СообщениеПользователю;
        Сообщение.Текст = "Контрагент: " + Данные.Контрагент + ", Объем продаж: " + Данные.ОбъемПродаж +
                          ", Поставщик?: "+ Данные.Поставщик + ", Телефоны: ";

        // Обойти в цикле элемент структуры данных Телефоны типа Массив.
        Для Каждого Телефон Из Данные.Телефоны Цикл
                Сообщение.Текст = Сообщение.Текст + Телефон + ", ";
        КонецЦикла;

        Сообщение.Сообщить();

КонецПроцедуры
```

Результат чтения JSON-документа, содержимое которого показано в листинге 1.20, будет выглядеть следующим образом (рис. 1.2).

Рис. 1.2. Чтение примитивных типов из JSON-документа

Чтение в соответствие

По умолчанию чтение данных методом `ПрочитатьJSON()` выполняется в структуру. Но некоторые внешние системы могут формировать имена свойств, которые, с точки зрения «1С:Предприятия», содержат недопустимые символы. Например, свойство может называться «user.first» или «user.last». С точки зрения платформы это недопустимое значение для строкового ключа (т. к. имя свойства не может содержать точку).

Например, JSON-документ содержит соответствие, имена свойств которого включают пробел и другие недопустимые символы (листинг 1.22).

Листинг 1.22. Содержимое JSON-документа

```
{
        "Контрагент №2": "ОАО Алмаз",
        "Объем Продаж": 6000000,
        "Мобильный тел.": "+79998887766",
        "Поставщик?": true
}
```

При стандартном чтении такого документа в структуру будет получена ошибка (рис. 1.3).

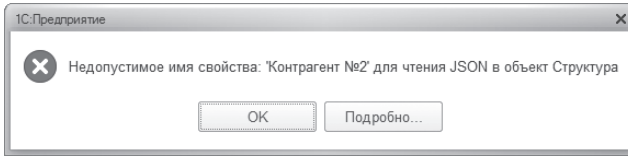


Рис. 1.3. Ошибка при чтении в структуру JSON-документа

В этом случае надо читать данные в соответствие, потому что ключи (имена свойств) соответствия могут быть любыми. Для этого вторым параметром в метод `ПрочитатьJSON(ЧтениеJSON, Истина)` надо передать истину (листинг 1.23).

Листинг 1.23. Пример десериализации соответствия из JSON-документа

```
&НаСервереБезКонтекста
Процедура РаботаССоответствиемНаСервере()

	// Создать объект чтения и открыть файл, из которого будет выполняться чтение.
	Чтение = Новый ЧтениеJSON;
	Чтение.ОткрытьФайл("c:\temp\Serialisation_8.json");

	// Выполнить чтение данных в соответствие Данные с помощью объекта чтения (Чтение).
	Данные = ПрочитатьJSON(Чтение, Истина);
	// Данные = ПрочитатьJSON(Чтение);

	// Завершить работу с файлом.
	Чтение.Закрыть();

	// Вывести результат чтения в сообщение.
	Сообщение = Новый СообщениеПользователю;
	Сообщение.Текст = "Контрагент: " + Данные["Контрагент №2"]
	                  + ", Объем продаж: " + Данные["Объем Продаж"] +
	                  ", Поставщик?: " + Данные["Поставщик?"] + ", Телефон моб.: " + Данные["Мобильный тел."];
	Сообщение.Сообщить();

КонецПроцедуры
```

Результат чтения JSON-документа, содержимое которого показано в листинге 1.22, будет выглядеть следующим образом (рис. 1.4).

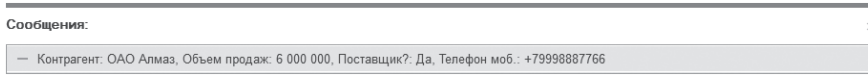


Рис. 1.4. Чтение данных из JSON-документа в соответствие

Функции преобразования и восстановления

Однако на практике далеко не всегда данные могут быть автоматически сериализованы в / десериализованы из JSON. Если данные обладают сложной структурой или требуют выполнения дополнительных преобразований при записи или чтении, то можно воспользоваться собственными функциями преобразования (при записи) или восстановления (при чтении) в/из JSON.

Для этого в методы `ЗаписатьJSON()` / `ПрочитатьJSON()` нужно передать в качестве параметров имя функции преобразования/восстановления, модуль, в котором эта функция находятся, а также, при необходимости, дополнительные параметры, которые будут переданы в эту функцию.

Необходимо помнить, что функции преобразования/восстановления должны быть объявлены с указанием ключевого слова `Экспорт`. Кроме того, если эти функции находятся в модуле формы, то они могут быть описаны только в «контекстной» части модуля, то есть с использованием директив компиляции `&НаКлиенте` или `&НаСервере`.

Функция преобразования

Функция преобразования, которая используется при сериализации данных в JSON, будет вызываться для всех свойств, тип которых не поддерживает преобразование в формат JSON напрямую. В этой функции разработчик может самостоятельно проанализировать переданный объект и вернуть платформе значение, которое может быть сериализовано в JSON, либо вообще отказаться от его записи.

Например, структура с информацией о контрагенте помимо примитивных типов данных содержит объект встроенного языка `ДанныеАдреса`. JSON ничего не знает о подобных типах, поэтому если попытаться записать данные в JSON обычным образом, без использования функции преобразования, то будет получена ошибка из-за недопустимых для сериализации типов данных. Поэтому в метод `ЗаписатьJSON()` мы передаем имя функции преобразования и указываем, что она находится в том же модуле формы (`ЭтотОбъект`), что и процедура сериализации данных (листинг 1.24).

Листинг 1.24. Пример сериализации в JSON-документ с использованием функции преобразования

```
&НаСервере
Процедура СериализацияСФункциейПреобразованияНаСервере()

    // Создать структуру с данными контрагента.
    Данные = Новый Структура;
    Данные.Вставить("Контрагент", "ОАО Топаз");
    Данные.Вставить("ОбъемПродаж", 5000000);
```

```
    // Записать свойство Адрес, имеющее тип ДанныеАдреса.
    ДопИнфо = Новый ДанныеАдреса(Новый Структура(
                "Город, Страна, Индекс", "Москва", "Россия", "112233"));
    Данные.Вставить("Адрес", ДопИнфо);
    Данные.Вставить("Поставщик", Ложь);

    // Создать объект записи и открыть файл, в который будет выполняться запись.
    Запись = Новый ЗаписьJSON;
    ПараметрыЗаписиJSON = Новый ПараметрыЗаписиJSON(ПереносСтрокJSON.Авто, Символы.Таб);
    Запись.ОткрытьФайл("c:\temp\Serialisation_4.json",,, ПараметрыЗаписиJSON);

    // Выполнить запись данных (Данные) с помощью объекта записи (Запись).
    ЗаписатьJSON(Запись, Данные, , "ФункцияПреобразованияЗаписи", ЭтотОбъект);

    // Завершить работу с файлом.
    Запись.Закрыть();

КонецПроцедуры
```

Отличие от обычной записи данных в JSON-документ заключается в том, что для всех свойств, тип которых не поддерживает автоматическую сериализацию в JSON, вызывается функция преобразования `ФункцияПреобразованияЗаписи()`.

В этой функции в параметрах `Свойство` и `Значение` содержатся соответственно имя и значение того поля структуры данных, которое не может быть автоматически сериализовано в JSON (листинг 1.25).

Листинг 1.25. Функция преобразования

```
&НаСервере
Функция ФункцияПреобразованияЗаписи(Свойство, Значение, ДополнительныеПараметры, Отказ) Экспорт

    Если Свойство = "Адрес" Тогда
        Адрес = "Страна: " + Значение.Страна + ", Индекс: " + Значение.Индекс + "
                                , Город: " + Значение.Город;
        Возврат Адрес;
    КонецЕсли;
    Отказ = Истина;

КонецФункции
```

В функции преобразования мы извлекаем данные из объекта `ДанныеАдреса` в строку, и эта строка возвращается в метод `ЗаписатьJSON()`.

В результате сформированный JSON-документ будет иметь следующий вид (листинг 1.26).

Листинг 1.26. Содержимое JSON-документа

```
{
	"Контрагент": "ОАО Топаз",
	"ОбъемПродаж": 5000000,
	"Адрес": "Страна: Россия, Индекс: 112233, Город: Москва",
	"Поставщик": false
}
```

Функция восстановления

При чтении из JSON функция восстановления может использоваться для того, чтобы преобразовать данные JSON в типы «1С:Предприятия», которые не могут являться результатом автоматического преобразования, либо чтобы просто изменить получаемые данные, если есть такая необходимость.

При использовании функции восстановления разработчик должен знать, какие данные и в каком виде находятся JSON-документе.

Например, требуется прочитать из JSON-документа, показанного в листинге 1.27, информацию о контрагенте и при этом представить телефоны контрагента в привычном формате: «999-99-99».

Листинг 1.27. Содержимое JSON-документа

```
{
	"Контрагент": "ОАО Топаз",
	"ОбъемПродаж": 5000000,
	"Телефоны": [
		"+71112223344",
		"+79998887766"
	],
	"Поставщик": false
}
```

Для этого в метод `ПрочитатьJSON()` мы передаем имя функции восстановления и указываем, что она находится в том же модуле формы (`ЭтотОбъект`), что и процедура десериализации (листинг 1.28).

Листинг 1.28. Пример десериализации данных из JSON-документа с функцией восстановления

```
&НаСервере
Процедура ДесериализацияСФункциейВосстановленияНаСервере()

	// Создать объект чтения и открыть файл, из которого будет выполняться чтение.
	Чтение = Новый ЧтениеJSON;
	Чтение.ОткрытьФайл("c:\temp\Serialisation_6.json");

	// Выполнить чтение данных в структуру Данные с помощью объекта чтения (Чтение).
```

```
    Данные = ПрочитатьJSON(Чтение, Ложь, , , "ФункцияВосстановленияЧтения", ЭтотОбъект);

    // Завершить работу с файлом.
    Чтение.Закрыть();


    // Вывести результат чтения в сообщение.
    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = "Контрагент: " + Данные.Контрагент + ", Объем продаж: " + Данные.ОбъемПродаж +
                      ", Поставщик?: "+ Данные.Поставщик + ", Телефоны: ";
    Для Каждого Телефон Из Данные.Телефоны Цикл
        Сообщение.Текст = Сообщение.Текст + Телефон + ", ";
    КонецЦикла;
    Сообщение.Сообщить();

КонецПроцедуры
```

Отличие от обычного чтения из JSON-документа заключается в том, что при чтении вызывается функция восстановления `ФункцияВосстановленияЧтения()`. Преобразованные с помощью функции восстановления данные сохраняются в структуру `Данные`. После этого структура данных выводится в сообщение пользователю.

В функции восстановления мы обходим массив телефонов и представляем каждый телефон в привычном виде. Результат восстановления возвращается в метод `ПрочитатьJSON()` (листинг 1.29).

Листинг 1.29. Функция восстановления

```
&НаСервере
Функция ФункцияВосстановленияЧтения(Свойство, Значение, ДополнительныеПараметры) Экспорт

    Если Свойство = "Телефоны" И ТипЗнч(Значение) = Тип("Массив") Тогда
        Телефоны = Новый Массив;
        Для Каждого Телефон Из Значение Цикл
            СтрокаТелефона = Лев(Телефон,2) + "-" + Сред(Телефон, 3, 3)
                + "-" + Сред(Телефон, 6, 3) + "-" +
                "-" + Сред(Телефон, 9, 2) + "-" + Прав(Телефон,2);
            Телефоны.Добавить(СтрокаТелефона);
        КонецЦикла;
        Возврат Телефоны;
  КонецЕсли;

КонецФункции
```

Результат чтения JSON-документа, содержимое которого показано в листинге 1.27, будет выглядеть следующим образом (рис. 1.5).

Рис. 1.5. Чтение из JSON-документа с функцией восстановления

Однако необходимо учитывать, что функция восстановления будет вызвана для всех свойств, которые будут обнаружены в JSON-документе. Это не всегда удобно и, кроме того, существенно снижает производительность чтения JSON-документа, особенно если он большой.

Для того чтобы ускорить чтение JSON, можно передать в метод `ПрочитатьJSON()` массив имен свойств, для которых требуется вызывать функцию восстановления (листинг 1.30).

Листинг 1.30. Пример десериализации данных из JSON-документа с функцией восстановления

```
&НаСервере
Процедура ДесериализацияСФункциейВосстановленияНаСервере()

	// Заполнить массив имен реквизитов для восстановления.
	РеквизитыДляВосстановления = Новый Массив;
	РеквизитыДляВосстановления.Добавить("Телефоны");

	// Создать объект чтения и открыть файл, из которого будет выполняться чтение.
	Чтение = Новый ЧтениеJSON;
	Чтение.ОткрытьФайл("c:\temp\Serialisation_6.json");

	// Выполнить чтение данных в структуру Данные с помощью объекта чтения (Чтение).
	Данные = ПрочитатьJSON(Чтение, Ложь, , , "ФункцияВосстановленияЧтения"
				, ЭтотОбъект, , РеквизитыДляВосстановления);

	// Завершить работу с файлом.
	Чтение.Закрыть();

	// Вывести результат чтения в сообщение.
	Сообщение = Новый СообщениеПользователю;
	Сообщение.Текст = "Контрагент: " + Данные.Контрагент + ", Объем продаж: " + Данные.ОбъемПродаж +
			", Поставщик?: "+ Данные.Поставщик + ", Телефоны: ";
	Для Каждого Телефон Из Данные.Телефоны Цикл
		Сообщение.Текст = Сообщение.Текст + Телефон + ", ";
	КонецЦикла;
	Сообщение.Сообщить();

КонецПроцедуры
```

В результате чтение данных из JSON-документа будет выполнено быстрее, так как функция восстановления будет вызвана только для свойств, имена которых перечислены в массиве `РеквизитыДляВосстановления`.

Сериализация типа «Дата»

JSON не стандартизует формат представления даты. Поэтому разработчик может самостоятельно определить формат представления даты в JSON-документе исходя из своих предпочтений и требований принимающей стороны.

«1С:Предприятие» поддерживает несколько форматов представления даты, которые можно задать с помощью системного перечисления ФорматДатыJSON:

- Формат ISO. В этом случае дата сериализуется следующим образом: «2019-03-15T00:00:00+0400».
- Формат JavaScript. В этом случае дата сериализуется следующим образом: «new Date(1234656000000)».
- Формат Microsoft. В этом случае дата сериализуется следующим образом: «\/Date(1234656000000)\/» или «/Date(1234656000000)/» (в зависимости от режима экранирования символов).

Дата может записываться в нескольких вариантах (для примера используется дата 10 марта 2019 13:14:15 в зоне UTC+4):

- как локальная дата: 2019-03-10T13:14:15;
- как локальная дата с указанием смещения: 2019-03-10T13:14:15+04:00;
- как дата в UTC: 2019-03-10T09:14:15Z.

Управлять этим можно с помощью системного перечисления ВариантЗаписиДатыJSON. Дату в варианте UTC можно записать в любом формате (ISO, JavaScript и Microsoft), остальные варианты представления даты возможны только в том случае, если сериализация выполняется в формате ISO.

Для установки желаемого формата и варианта записи даты в JSON-документ у метода `ЗаписатьJSON()` можно использовать параметр `НастройкиСериализации`.

При чтении даты из JSON-документа нужно учитывать, что значения даты представляются в нем строкой. В параметре `ИменаСвойствСоЗначениямиДата` метода `ПрочитатьJSON()` можно перечислить те свойства JSON, значения которых нужно преобразовать в дату «1С:Предприятия». А в параметре `ОжидаемыйФорматДаты` можно указать, в каком формате эти данные содержатся в JSON (ISO, JavaScript или Microsoft). Однако если окажется, что в какой-то момент формат данных JSON не совпадает с ожидаемым форматом, то будет вызвано исключение.

Поясним вышесказанное на примерах. Сначала попробуем записать значение даты обычным образом (листинг 1.31).

Листинг 1.31. Пример сериализации даты в JSON-документ

```
&НаСервереБезКонтекста
Процедура СериализацияТипаДатаНаСервере()

	// Создать структуру с данными контрагента.
	Данные = Новый Структура;
	Данные.Вставить("Контрагент", "ОАО Фонтан");
	Данные.Вставить("Телефон", "81234567788");
	Данные.Вставить("ДатаИзменения", ТекущаяДата());

	// Создать объект записи и открыть файл, в который будет выполняться запись.
	Запись = Новый ЗаписьJSON;
	ПараметрыЗаписиJSON = Новый ПараметрыЗаписиJSON(ПереносСтрокJSON.Авто, Символы.Таб);
	Запись.ОткрытьФайл("c:\temp\Serialisation_2.json",,, ПараметрыЗаписиJSON);

	// Выполнить запись данных (Данные) с помощью объекта записи (Запись).
	ЗаписатьJSON(Запись, Данные);

	// Завершить работу с файлом.
	Запись.Закрыть();

КонецПроцедуры
```

В результате сформированный JSON-документ будет иметь следующий вид (листинг 1.32).

Листинг 1.32. Содержимое JSON-документа

```
{
	"Контрагент": "ОАО Фонтан",
	"Телефон": "81234567788",
	"ДатаИзменения": "2019-02-28T20:48:54"
}
```

То есть стандартно дата сериализуется как локальная дата в формате ISO. Прочитать такую дату можно следующим образом (листинг 1.33).

Листинг 1.33. Пример десериализации даты из JSON-документа

```
&НаСервереБезКонтекста
Процедура ДесериализацияТипаДатаНаСервере()

	// Создать объект чтения и открыть файл, из которого будет выполняться чтение.
	Чтение = Новый ЧтениеJSON;
	Чтение.ОткрытьФайл("c:\temp\Serialisation_2.json");

	// Выполнить чтение данных в структуру Данные с помощью объекта чтения (Чтение).
	Данные = ПрочитатьJSON(Чтение, Ложь, "ДатаИзменения", ФорматДатыJSON.ISO);

	// Завершить работу с файлом.
	Чтение.Закрыть();
```

```
    // Вывести результат чтения в сообщение.
    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = "Контрагент: " + Данные.Контрагент + ", Телефон: " + Данные.Телефон +
                      ", Дата изменения: "+ Данные.ДатаИзменения;
    Сообщение.Сообщить();

КонецПроцедуры
```

В процедуре десериализации при вызове метода `ПрочитатьJSON()` мы указываем имя свойства, содержащее дату («ДатаИзменения»), и ожидаемый формат даты – значение ISO перечисления ФорматДатыJSON.

Результат чтения JSON-документа, содержимое которого показано в листинге 1.32, будет выглядеть следующим образом (рис. 1.6).

Рис. 1.6. Чтение даты из JSON-документа

Однако если указать другой ожидаемый формат даты, например ФорматДатыJSON.JavaScript, то будет получена ошибка из-за неверного формата даты.

Чтобы этого избежать, можно считывать данные обычным образом, а затем использовать так называемую постобработку при помощи функции `ПрочитатьДатуJSON()` (листинг 1.34).

Листинг 1.34. Пример десериализации даты из JSON-документа

```
&НаСервереБезКонтекста
Процедура ДесериализацияТипаДатаНаСервере()

    // Создать объект чтения и открыть файл, из которого будет выполняться чтение.
    Чтение = Новый ЧтениеJSON;
    Чтение.ОткрытьФайл("c:\temp\Serialisation_2.json");

    // Выполнить чтение данных в структуру Данные с помощью объекта чтения (Чтение).
    Данные = ПрочитатьJSON(Чтение);

    // Завершить работу с файлом.
    Чтение.Закрыть();

    // Прочитать дату JSON в нужном формате.
    ДатаИзменения = ПрочитатьДатуJSON(Формат(Данные.ДатаИзменения, "ЧГ=")
                                      , ФорматДатыJSON.ISO);

    // Вывести результат чтения в сообщение.
    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = "Контрагент: " + Данные.Контрагент + ", Телефон: " + Данные.Телефон +
                      ", Дата изменения: " + ДатаИзменения;
    Сообщение.Сообщить();

КонецПроцедуры
```

Результат чтения будет аналогичен показанному на рис. 1.6.

Теперь укажем формат и вариант записи даты при помощи параметра `НастройкиСериализации` (листинг 1.35).

Листинг 1.35. Пример сериализации даты в JSON-документ

```
&НаСервереБезКонтекста
Процедура СериализацияТипаДатаНаСервере()

	// Создать структуру с данными контрагента.
	Данные = Новый Структура;
	Данные.Вставить("Контрагент", "ОАО Фонтан");
	Данные.Вставить("Телефон", "81234567788");
	Данные.Вставить("ДатаИзменения", ТекущаяДата());

	// Создать настройки сериализации для записи дат.
	НастройкиСериализации = Новый НастройкиСериализацииJSON;
	НастройкиСериализации.ФорматСериализацииДаты = ФорматДатыJSON.JavaScript;
	НастройкиСериализации.ВариантЗаписиДаты
	                                    = ВариантЗаписиДатыJSON.УниверсальнаяДата;

	// Создать объект записи и открыть файл, в который будет выполняться запись.
	Запись = Новый ЗаписьJSON;
	ПараметрыЗаписиJSON = Новый ПараметрыЗаписиJSON(ПереносСтрокJSON.Авто, Символы.Таб);
	Запись.ОткрытьФайл("c:\temp\Serialisation_3.json",,, ПараметрыЗаписиJSON);

	// Выполнить запись данных (Данные) с помощью объекта записи (Запись).
	ЗаписатьJSON(Запись, Данные, НастройкиСериализации);

	// Завершить работу с файлом.
	Запись.Закрыть();

КонецПроцедуры
```

В результате сформированный JSON-документ будет иметь следующий вид (листинг 1.36).

Листинг 1.36. Содержимое JSON-документа

```
{
	"Контрагент": "ОАО Фонтан",
	"Телефон": "81234567788",
	"ДатаИзменения": "new Date(1551382927000)"
}
```

В показанном примере дата сериализуется как универсальная дата в формате JavaScript. Прочитать такую дату можно следующим образом (листинг 1.37).

Листинг 1.37. Пример десериализации даты из JSON-документа

```
&НаСервереБезКонтекста
Процедура ДесериализацияТипаДатаНаСервере()

	// Создать объект чтения и открыть файл, из которого будет выполняться чтение.
	Чтение = Новый ЧтениеJSON;
	Чтение.ОткрытьФайл("c:\temp\Serialisation_3.json");

	// Выполнить чтение данных в структуру Данные с помощью объекта чтения (Чтение).
	Данные = ПрочитатьJSON(Чтение, Ложь, "ДатаИзменения", ФорматДатыJSON.JavaScript);

	// Завершить работу с файлом.
	Чтение.Закрыть();

	// Вывести результат чтения в сообщение.
	Сообщение = Новый СообщениеПользователю;
	Сообщение.Текст = "Контрагент: " + Данные.Контрагент + ", Телефон: " + Данные.Телефон +
			", Дата изменения: "+ Данные.ДатаИзменения;
	Сообщение.Сообщить();

КонецПроцедуры
```

В процедуре десериализации при вызове метода `ПрочитатьJSON()` мы указываем имя свойства, содержащее дату («ДатаИзменения»), и ожидаемый формат даты – значение JavaScript перечисления ФорматДатыJSON.

Результат чтения будет аналогичен показанному на рис. 1.6.

Этого же результата можно добиться, выполнив постобработку при помощи функции `ПрочитатьДатуJSON()` (листинг 1.38).

Листинг 1.38. Пример десериализации даты из JSON-документа

```
&НаСервереБезКонтекста
Процедура ДесериализацияТипаДатаНаСервере()

	// Создать объект чтения и открыть файл, из которого будет выполняться чтение.
	Чтение = Новый ЧтениеJSON;
	Чтение.ОткрытьФайл("c:\temp\Serialisation_2.json");

	// Выполнить чтение данных в структуру Данные с помощью объекта чтения (Чтение).
	Данные = ПрочитатьJSON(Чтение);

	// Завершить работу с файлом.
	Чтение.Закрыть();

	// Прочитать дату JSON в нужном формате.
	ДатаИзменения = ПрочитатьДатуJSON(Формат(Данные.ДатаИзменения, "ЧГ=")
			, ФорматДатыJSON.JavaScript);

	// Вывести результат чтения в сообщение.
	Сообщение = Новый СообщениеПользователю;
	Сообщение.Текст = "Контрагент: " + Данные.Контрагент + ", Телефон: " + Данные.Телефон +
			", Дата изменения: " + ДатаИзменения;
	Сообщение.Сообщить();

КонецПроцедуры
```

Вместо постобработки данных в процедуре чтения лучше и методологически более правильно использовать чтение с функцией восстановления.

Например, дата в JSON-документе может быть представлена в виде числа (листинг 1.39).

Листинг 1.39. Содержимое JSON-документа

```
{
	"Контрагент": "ОАО Фонтан",
	"Телефон": "81234567788",
	"ДатаИзменения": 63661814555
}
```

Чтобы прочитать такую дату, можно использовать чтение с функцией восстановления (листинг 1.40).

Листинг 1.40. Пример десериализации даты из JSON-документа с функцией восстановления

```
&НаСервереБезКонтекста
Процедура ДесериализацияТипаДатаНаСервере()

	// Заполнить массив имен реквизитов для восстановления.
	РеквизитыДляВосстановления = Новый Массив;
	РеквизитыДляВосстановления.Добавить("Телефон");
	РеквизитыДляВосстановления.Добавить("ДатаИзменения");

	// Создать объект чтения и открыть файл, из которого будет выполняться чтение.
	Чтение = Новый ЧтениеJSON;
	Чтение.ОткрытьФайл("c:\temp\Serialisation_7.json");

	// Выполнить чтение данных в структуру Данные с помощью объекта чтения (Чтение).
	Данные = ПрочитатьJSON(Чтение, Ложь, , , "ФункцияВосстановленияЧтения"
	                                      , ЭтотОбъект, , РеквизитыДляВосстановления);

	// Завершить работу с файлом.
	Чтение.Закрыть();

	// Вывести результат чтения в сообщение.
	Сообщение = Новый СообщениеПользователю;
	Сообщение.Текст = "Контрагент: " + Данные.Контрагент + ", Телефон: " + Данные.Телефон +
	                  ", Дата изменения: "+ Данные.ДатаИзменения;
	Сообщение.Сообщить();

КонецПроцедуры
```

В функции восстановления дата, выраженная простым числом, с помощью функции `ПрочитатьДатуJSON()` приводится к формату JavaScript (листинг 1.41).

Листинг 1.41. Функция восстановления

```
&НаСервере
Функция ФункцияВосстановленияЧтения(Свойство, Значение, ДополнительныеПараметры) Экспорт

    Если Свойство = "ДатаИзменения" Тогда
        Возврат ПрочитатьДатуJSON("new Date(" + Формат(Значение, "ЧГ=")+")"
                                  , ФорматДатыJSON.JavaScript);
    КонецЕсли;
    Если Свойство = "Телефон" Тогда
        СтрокаТелефона = Лев(Значение,1) + " " + Сред(Значение, 2, 3) + "-" + Сред(Значение, 5, 3) + "-" +
            Сред(Значение, 8, 2) + "-" + Прав(Значение,2);
        Возврат СтрокаТелефона;
    КонецЕсли;

КонецФункции
```

Результат чтения JSON-документа, содержимое которого показано в листинге 1.39, будет выглядеть следующим образом (рис. 1.7).

Рис. 1.7. Чтение даты из JSON-документа с функцией восстановления

При сериализации/десериализации даты с использованием потоковой техники также можно использовать методы глобального контекста `ЗаписатьДатуJSON()` / `ПрочитатьДатуJSON()`.

Например, в следующем фрагменте потоковой записи дата сериализуется как универсальная дата в формате ISO (листинг 1.42).

Листинг 1.42. Сериализация даты в JSON-документ с использованием потоковой техники

```
...
    Запись.ЗаписатьИмяСвойства("ДатаИзменения");
    Запись.ЗаписатьЗначение(ЗаписатьДатуJSON(ТекущаяДата(), ФорматДатыJSON.ISO
                                , ВариантЗаписиДатыJSON.УниверсальнаяДата));
...
```

В результате в сформированном JSON-документе значение даты будет выглядеть следующим образом (листинг 1.43).

Листинг 1.43. Фрагмент сформированного JSON-документа

```
...
    "ДатаИзменения": "2019-03-22T12:50:16Z"
...
```

Прочитать такую дату можно следующим образом (листинг 1.44).

Листинг 1.44. Десериализация даты из JSON-документа с использованием потоковой техники

```
...
    Если Чтение.ТекущееЗначение = "ДатаИзменения" Тогда
        Чтение.Прочитать();
        Сообщение.Текст = "Значение = " +
            ПрочитатьДатуJSON(Формат(Чтение.ТекущееЗначение, "ЧГ=")
                , ФорматДатыJSON.ISO);
        Сообщение.Сообщить();
    КонецЕсли;
...
```

Результат чтения JSON-документа, фрагмент которого показан в листинге 1.43, будет выглядеть следующим образом (рис. 1.8).

Рис. 1.8. Потоковое чтение даты из JSON-документа

Сериализация прикладных типов «1С:Предприятия»

Помимо потоковой записи/чтения и сериализации примитивных типов и коллекций в платформе разработаны средства, позволяющие сериализовать/десериализовать прикладные типы «1С:Предприятия»: ссылки, объекты, наборы записей и вообще любые типы, для которых поддерживается XDTO-сериализация.

В основном XDTO-сериализацию в JSON рекомендуется использовать при обмене данными между двумя прикладными решениями «1С:Предприятия». Преимущества обмена через формат JSON состоят в том, что по сравнению с обменом через XML этот обмен работает быстрее, а сами файлы обмена в формате JSON записываются более компактно.

Кроме того, этот механизм можно использовать и для обмена с внешними системами, готовыми принимать типы данных «1С:Предприятия». Например, XDTO-сериализацию в JSON можно использовать для организации собственного HTTP-интерфейса прикладного решения. Сервис на платформе «1С:Предприятия» будет формировать ответ в памяти в виде строки JSON. А затем передавать ее при помощи объекта `HTTPСервисОтвет`. Подробнее об этом будет рассказано в разделе «HTTP-запросы».

Запись

JSON-сериализация «эмулирует» XML-сериализацию, в силу чего получающийся JSON-документ внешне выглядит очень похоже на соответствующий XML-документ. В JSON-документ могут быть помещены любые объекты системы «1С:Предприятие», для которых указано, что они могут быть сериализованы в XDTO.

Общие принципы сериализации в JSON идентичны XDTO-сериализации в XML. В частности:

- структура данных соответствует структуре XML-документа;
- имеются незначительные отличия, связанные с особенностями хранения типов и представлением массивов в JSON;
- порядок и состав свойств JSON соответствует структуре объекта конфигурации и не может изменяться.

Сериализация прикладных типов в JSON-документ происходит с помощью метода `ЗаписатьJSON()` объекта глобального контекста `СериализаторXDTO`.

При этом существует одна важная особенность. В отличие от XML для записи начала объекта JSON нет необходимости указывать его имя. Поэтому сериализация/десериализация в/из JSON может выполняться одним из двух способов.

Во-первых, при сериализации в параметре `НазначениеТипаXML` метода `ЗаписатьJSON()` можно указать явное назначение типа (`НазначениеТипаXML.Явное`). Десериализовать такой объект можно без указания типа считываемого значения (листинг 1.45).

Листинг 1.45. Пример сериализации элемента справочника в JSON-документ с указанием типа

```
&НаСервереБезКонтекста
Процедура СериализацияПрикладныхТиповНаСервере()

	// Получить записываемый объект.
	ССылкаНаОбъект = Справочники.Сотрудники.НайтиПоКоду("000000003");
	Объект = ССылкаНаОбъект.ПолучитьОбъект();

	// Создать объект записи и открыть файл, в который будет выполняться запись.
	Запись = Новый ЗаписьJSON;
	ПараметрыЗаписиJSON = Новый ПараметрыЗаписиJSON(ПереносСтрокJSON.Авто, Символы.Таб);
	Запись.ОткрытьФайл("c:\temp\SerialisationXDTO.json",,, ПараметрыЗаписиJSON);

	// Выполнить запись данных (Объект) с помощью объекта записи (Запись).
	СериализаторXDTO.ЗаписатьJSON(Запись, Объект, НазначениеТипаXML.Явное);

	// Завершить работу с файлом.
	Запись.Закрыть();

КонецПроцедуры
```

В процедуре записи ссылка на объект справочника Сотрудники находится по коду, а затем данные объекта записываются в JSON-документ с указанием его типа.

В результате сформированный JSON-документ будет иметь следующий вид (листинг 1.46).

Листинг 1.46. Содержимое JSON-документа

```
{
	"#type": "jcfg:CatalogObject.Сотрудники",
	"#value": {
		"Ref": "0b6cd7dc-36a0-11e9-8bb2-642737df2048",
		"DeletionMark": false,
		"Code": "000000003",
		"Description": "Артемов Игорь Владимирович",
		"ДатаРождения": "1970-01-31T00:00:00",
		"КоличествоДетей": 2,
		"Работает": true,
		"Стаж": "25.03.20"
	}
}
```

Из соображений компактности сформированного JSON-документа для встроенных типов платформы пространства имен не записываются полностью в отдельном свойстве JSON, а указываются в виде префикса перед типом, например jcfg (см. листинг 1.46).

В данном случае префикс cfg обозначает пространство имен http://v8.1c.ru/8.1/data/enterprise/current-config. Соответствие префиксов пространствам имен «зашито» в платформу. Поэтому обмен данными между прикладными решениями «1С:Предприятия» выполняется стандартным образом. А при обмене с внешними системами для правильной десериализации типов платформы внешней системе понадобится полная таблица соответствия, приведенная в документации.

Во-вторых, можно выполнить сериализацию в JSON без явного назначения типа (листинг 1.47). Но тогда тип нужно будет указать при десериализации (см. листинг 1.50).

Листинг 1.47. Пример сериализации элемента справочника в JSON-документ без указания типа

```
&НаСервереБезКонтекста
Процедура СериализацияПрикладныхТиповНаСервере()

	// Получить записываемый объект.
	СсылкаНаОбъект = Справочники.Сотрудники.НайтиПоКоду("000000002");
	Объект = СсылкаНаОбъект.ПолучитьОбъект();
```

```
    // Создать объект записи и открыть файл, в который будет выполняться запись.
    Запись = Новый ЗаписьJSON;
    ПараметрыЗаписиJSON = Новый ПараметрыЗаписиJSON(ПереносСтрокJSON.Авто, Символы.Таб);
    Запись.ОткрытьФайл("c:\temp\SerialisationXDTO_2.json",,, ПараметрыЗаписиJSON);

    // Выполнить запись данных (Объект) с помощью объекта записи (Запись).
    СериализаторXDTO.ЗаписатьJSON(Запись, Объект);

    // Завершить работу с файлом.
    Запись.Закрыть();

КонецПроцедуры
```

В результате сформированный JSON-документ будет иметь следующий вид (листинг 1.48).

Листинг 1.48. Содержимое JSON-документа

```
{
    "#value": {
        "Ref": "0b6cd7db-36a0-11e9-8bb2-642737df2048",
        "DeletionMark": false,
        "Code": "000000002",
        "Description": "Смирнова Светлана Ивановна",
        "ДатаРождения": "1990-02-22T00:00:00",
        "КоличествоДетей": 0,
        "Работает": false,
        "Стаж": "03.10.27"
    }
}
```

Кроме того, нужно учитывать, что сериализация значений типа `Дата` выполняется в формате ISO. Это определяется механизмом XDTO и не управляется при записи данных. Также при операции сериализации не поддерживается использование функции преобразования, в отличие от потоковой и объектной техник.

Чтение

Общие принципы десериализации также связаны с использованием механизма XDTO. В частности:

- десериализуются только те типы, которые поддерживают XDTO-сериализацию;
- порядок свойств JSON должен соответствовать структуре объекта конфигурации.

Десериализация прикладных типов из JSON-документа происходит с помощью метода `ПрочитатьJSON()` объекта глобального контекста `СериализаторXDTO`. При этом может быть либо указан тип считываемого

значения, либо нет – в зависимости от того, указан ли тип прикладного объекта при сериализации.

Например, чтение JSON-документа, содержимое которого показано в листинге 1.46, может быть выполнено без указания типа считываемого значения (листинг 1.49).

Листинг 1.49. Пример десериализации элемента справочника из JSON-документа без указания типа

```
&НаСервереБезКонтекста
Процедура ДесериализацияПрикладныхТиповНаСервере()

	// Создать объект чтения и открыть файл, из которого будет выполняться чтение.
	Чтение = Новый ЧтениеJSON;
	Чтение.ОткрытьФайл("c:\temp\SerialisationXDTO.json");

	// Выполнить чтение данных в структуру Данные с помощью объекта чтения (Чтение).
	Данные = СериализаторXDTO.ПрочитатьJSON(Чтение);

	// Завершить работу с файлом.
	Чтение.Закрыть();

	// Вывести результат чтения в сообщение.
	Сообщение = Новый СообщениеПользователю;
	Сообщение.Текст = "Код: " + Данные.Code + ", ФИО: " + Данные.Description + ", ДР:
			"+ Данные.ДатаРождения +
			", Детей: " + Данные.КоличествоДетей + ", Работает: " + Данные.Работает + "
			, Стаж: " + Данные.Стаж;
	Сообщение.Сообщить();

КонецПроцедуры
```

Результат чтения элемента справочника Сотрудники из JSON-документа будет выглядеть следующим образом (рис. 1.9).

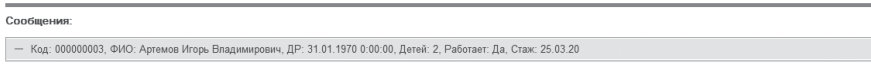


Рис. 1.9. Чтение элемента справочника из JSON-документа

Если же тип прикладного объекта не указан при сериализации (см. листинги 1.47, 1.48), то при чтении JSON-документа тип считываемого значения должен быть указан в методе `ПрочитатьJSON()` объекта `СериализаторXDTO` (листинг 1.50).

Листинг 1.50. Пример десериализации элемента справочника из JSON-документа с указанием типа

```
&НаСервереБезКонтекста
Процедура ДесериализацияПрикладныхТиповНаСервере()

	// Создать объект чтения и открыть файл, из которого будет выполняться чтение.
	Чтение = Новый ЧтениеJSON;
	Чтение.ОткрытьФайл("c:\temp\SerialisationXDTO_2.json");

	// Выполнить чтение данных в структуру Данные с помощью объекта чтения (Чтение).
	Данные = СериализаторXDTO.ПрочитатьJSON(Чтение, Тип("СправочникОбъект.Сотрудники"));

	// Завершить работу с файлом.
	Чтение.Закрыть();

	// Вывести результат чтения в сообщение.
	Сообщение = Новый СообщениеПользователю;
	Сообщение.Текст = "Код: " + Данные.Code + ", ФИО: " + Данные.Description
			+ ", ДР: "+ Данные.ДатаРождения +
			", Детей: " + Данные.КоличествоДетей + ", Работает: " + Данные.Работает
			+ ", Стаж: " + Данные.Стаж;
	Сообщение.Сообщить();

КонецПроцедуры
```

Десериализация прикладных типов с помощью функции восстановления

При десериализации, так же как и в случае с примитивными типами и коллекциями, есть возможность изменить получаемые данные. Для этого можно использовать функцию восстановления.

Например, при переносе элемента справочники Сотрудники из одной базы в другую требуется каким-то образом изменить его код. Пусть сформированный JSON-документ имеет следующий вид (листинг 1.51).

Листинг 1.51. Содержимое JSON-документа

```
{
	"#type": "jcfg:CatalogObject.Сотрудники",
	"#value": {
		"Ref": "9575f8ed-369f-11e9-8bb2-642737df2048",
		"DeletionMark": false,
		"Code": "000000001",
		"Description": "Алексеев Сергей Иванович",
		"ДатаРождения": "1980-12-10T00:00:00",
		"КоличествоДетей": 1,
		"Работает": true,
		"Стаж": "10.05.01",
		"Должность": "88d54406-36a1-11e9-8bb2-642737df2048"
	}
}
```

Для этого в метод `ПрочитатьJSON()` мы передаем имя функции восстановления и массив имен свойств, для которых она должна вызываться (листинг 1.52).

Листинг 1.52. Пример десериализации элемента справочника с функцией восстановления

```
&НаСервере
Процедура ВосстановлениеПриДесериализацииХДТОНаСервере()

	// Создать объект чтения и открыть файл, из которого будет выполняться чтение.
	Чтение = Новый ЧтениеJSON;
	Чтение.ОткрытьФайл("c:\temp\SerialisationXDTO_3.json");

	// Заполнить массив имен реквизитов для восстановления.
	РеквизитыДляВосстановления = Новый Массив;
	РеквизитыДляВосстановления.Добавить("Code");

	// Выполнить чтение данных в структуру Данные с помощью объекта чтения (Чтение).
	Данные = СериализаторXDTO.ПрочитатьJSON(Чтение, , "ФункцияВосстановленияЧтенияXDTO"
	                                        , ЭтотОбъект, , , РеквизитыДляВосстановления);

	// Завершить работу с файлом.
	Чтение.Закрыть();

	// Вывести результат чтения в сообщение.
	Сообщение = Новый СообщениеПользователю;
	Сообщение.Текст = "Код: " + Данные.Code + ", ФИО: " + Данные.Description
		+ ", ДР: "+ Данные.ДатаРождения +
		", Детей: " + Данные.КоличествоДетей + ", Работает: " + Данные.Работает
		+ ", Стаж: " + Данные.Стаж +
		", Должность: " + Данные.Должность;
	Сообщение.Сообщить();

КонецПроцедуры
```

Функцию восстановления заполним следующим образом (листинг 1.53).

Листинг 1.53. Функция восстановления

```
&НаСервере
Функция ФункцияВосстановленияЧтенияXDTO(Свойство, Тип, Значение, ДополнительныеПараметры) Экспорт

	Если Свойство = "Code" Тогда
		КодОбмена = "НМК-" + Прав(Значение.Значение, 5);
		ТипXDTOКод = ФабрикаXDTO.Тип("http://www.w3.org/2001/XMLSchema","string");
		Возврат ФабрикаXDTO.Создать(ТипXDTOКод, КодОбмена);
	КонецЕсли;

КонецФункции
```

В функции восстановления в параметре `Значение` будет содержаться объект XDTO. Его значение (`Значение.Значение`) мы изменяем нужным образом и возвращаем в метод `ПрочитатьJSON()`.

Результат чтения JSON-документа, содержимое которого показано в листинге 1.51, будет выглядеть следующим образом (рис. 1.10).

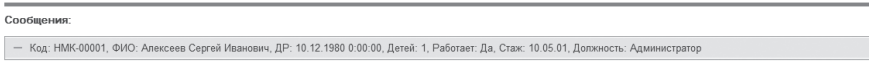


Рис. 1.10. Чтение элемента справочника из JSON-документа с функцией восстановления

Смешанная техника работы

Для упрощения работы с JSON можно совмещать различные техники при формировании одного документа. Например, необходимо сформировать документ, который содержит некоторый набор структур и массив. В этом случае можно все оформление документа выполнять с помощью потоковой техники, а уже готовые структуры и массив записывать с помощью объектной техники.

Например, нам нужно записать в JSON-документ список оплат поставщикам за поступившие товары. Этот список должен содержать дату формирования и набор оплат (соответствующий некоторому критерию).

Каждая оплата поставщику характеризуется следующими параметрами:

- номер оплаты;
- дата совершения оплаты;
- поставщик по оплате;
- сумма оплаты;
- ссылка на оплату, по которой впоследствии можно получить всю информацию по оплате поставщику.

Процедура, которая формирует JSON-документ, может иметь следующий вид (листинг 1.54).

Листинг 1.54. Пример сериализации в JSON-документ с применением смешанной техники

```
&НаСервереБезКонтекста
Процедура СовмещениеТехникНаСервере()

    СписокОплат = Документы.ОплатаПоставщикам.Выбрать();

    // Создать настройки сериализации для записи дат.
```

```
    НастройкиСериализации = Новый НастройкиСериализацииJSON;
    НастройкиСериализации.ВариантЗаписиДаты = ВариантЗаписиДатыJSON.УниверсальнаяДата;
    НастройкиСериализации.ФорматСериализацииДаты = ФорматДатыJSON.ISO;
    НастройкиСериализации.СериализовыватьМассивыКакОбъекты = Истина;

    // Задать параметры записи JSON.
    ПараметрыJSON = Новый ПараметрыЗаписиJSON(ПереносСтрокJSON.Авто, " ", Истина);

    // Создать объект записи и открыть файл, в который будет выполняться запись.
    Запись = Новый ЗаписьJSON;
    Запись.ПроверятьСтруктуру = Истина;
    Запись.ОткрытьФайл("c:\temp\combinedWrite.json", , , ПараметрыJSON);

    // Выполнить запись значений с помощью объекта записи (Запись).
    // Записать начало корневого объекта.
    Запись.ЗаписатьНачалоОбъекта();

    Запись.ЗаписатьИмяСвойства("ДатаФормирования");
    Запись.ЗаписатьЗначение(ЗаписатьДатуJSON(ТекущаяДата(), ФорматДатыJSON.ISO
                                , ВариантЗаписиДатыJSON.УниверсальнаяДата));

    Запись.ЗаписатьИмяСвойства("Оплаты");

    // Создать массив структур с данными документа ОплатаПоставщикам.
    Оплаты = Новый Массив;
    Пока СписокОплат.Следующий() Цикл
        ОплатаПоставщику = Новый Структура("Ссылка, Номер, Дата, Поставщик, Сумма");
        ОплатаПоставщику.Ссылка = Строка(СписокОплат.Ссылка);
        ОплатаПоставщику.Номер = СписокОплат.Номер;
        ОплатаПоставщику.Дата = СписокОплат.Дата;
        ОплатаПоставщику.Поставщик = Строка(СписокОплат.Поставщик);
        ОплатаПоставщику.Сумма = СписокОплат.Сумма;

        Оплаты.Добавить(ОплатаПоставщику);
    КонецЦикла;

    // Выполнить запись данных (Оплаты) с помощью объекта записи (Запись).
    ЗаписатьJSON(Запись, Оплаты, НастройкиСериализации);

    // Записать конец корневого объекта.
    Запись.ЗаписатьКонецОбъекта();

    // Завершить работу с файлом.
    Запись.Закрыть();

КонецПроцедуры
```

В процедуре сначала создается объект `Запись`, который открывает файл для низкоуровневой записи. Затем с помощью потоковой техники в файл записывается дата формирования списка оплат как универсальная дата в формате ISO.

После этого создается массив `Оплаты`. Выборка документов ОплатаПоставщикам обходится в цикле, и на каждом шаге цикла структура `ОплатаПоставщику`, заполненная данными этой выборки, добавляется в массив `Оплаты`.

Затем с помощью объектной техники методом `ЗаписатьJSON()` массив `Оплаты` записывается в JSON-документ.

В результате сформированный JSON-документ будет иметь следующий вид (листинг 1.55).

Листинг 1.55. Содержимое JSON-документа

```
{
 "ДатаФормирования": "2019-03-22T15:06:10Z",
 "Оплаты": {
  "0": {
   "Ссылка": "Оплата поставщикам 000000003 от 05.03.2019 12:00:00",
   "Номер": "000000003",
   "Дата": "2019-03-05T09:00:00Z",
   "Поставщик": "ООО \"Топаз\"",
   "Сумма": 500500
  },
  "1": {
   "Ссылка": "Оплата поставщикам 000000001 от 13.03.2019 12:00:00",
   "Номер": "000000001",
   "Дата": "2019-03-13T09:00:00Z",
   "Поставщик": "ООО \"Стиль\"",
   "Сумма": 300300
  },
  "2": {
   "Ссылка": "Оплата поставщикам 000000002 от 20.03.2019 12:00:00",
   "Номер": "000000002",
   "Дата": "2019-03-20T09:00:00Z",
   "Поставщик": "ОАО \"Фонтан\"",
   "Сумма": 700700
  }
 }
}
```

HTTP-сервисы (REST)

Платформа «1С:Предприятие» позволяет прикладному разработчику реализовать собственные HTTP-сервисы. С помощью HTTP-сервисов можно создать в прикладном решении дополнительный прикладной интерфейс, доступ к которому из внешних систем осуществляется посредством HTTP-запросов.

Общая информация

Одним из распространенных способов взаимодействия приложений в Интернете является использование веб-сервисов. Одно приложение (поставщик веб-сервиса) публикует свой веб-сервис по уникальному веб-адресу в Интернете, а другие приложения (потребители веб-сервиса) могут обратиться к этому веб-сервису, для того чтобы получить какие-то данные от поставщика, передать ему какие-то данные или заставить поставщика выполнить какие-то действия.

Существуют разные протоколы и разные архитектуры, с помощью которых реализуются веб-сервисы. В платформе поддерживается работа с двумя видами веб-сервисов: веб-сервисы, реализованные в архитектуре REST (в «1С:Предприятии» они называются HTTP-сервисы), и веб-сервисы, работающие по протоколу SOAP (в «1С:Предприятии» они называются Web-сервисы).

В этом разделе мы будем рассматривать HTTP-сервисы (REST) как наиболее популярные и часто используемые. Про Web-сервисы (SOAP) вы можете прочитать далее, в разделе «Web-сервисы».

Не вдаваясь в технические подробности, можно сказать, что HTTP-сервисы пользуются популярностью благодаря следующим качествам:

- простота программирования клиента таких сервисов;
- потенциально меньший объем передаваемых данных;
- потенциально меньшая вычислительная нагрузка;
- HTTP-сервисы ориентированы на «ресурсы», в то время как SOAP-сервисы ориентированы на «действия».

Первые три фактора особенно важны для приложений, работающих на мобильных устройствах.

Обращение к HTTP-сервису выполняется по некоторому URL. В качестве примера можно привести HTTP-сервис, который при обращении по одному URL возвращает список каких-либо документов (например, расходных накладных), а при обращении по другому URL возвращает конкретную накладную. Действие, которое следует выполнить серверу, определяется тем, какой HTTP-метод адресован серверу. Например, если на сервер отправляется GET-запрос, то выполняется получение каких-либо данных, DELETE-запрос приводит к удалению данных и т. д.

При реализации HTTP-сервиса прикладной разработчик должен определить следующие элементы сервиса:

- Базовую часть URL, по которому будет выполняться обращение к сервису.
- Состав предоставляемой функциональности и структуру ресурсов, на которых эта функциональность будет отображаться.
- Действия (HTTP-методы), которые можно будет выполнить при обращении к тому или иному ресурсу.
- Для каждого выбранного действия разработчик должен создать специальный метод, написанный на встроенном языке платформы «1С:Предприятие», который и реализует необходимую функциональность. Метод располагается в специальном модуле, связанном с HTTP-сервисом.

После реализации HTTP-сервиса необходимо опубликовать его на веб-сервере с помощью стандартного механизма публикации.

Для обращения к HTTP-сервису нужно сформировать URL, который в общем виде выглядит следующим образом: `http://host/base/hs/<корневойURL>/<относительныйURL>`. Рассмотрим более подробно составные части адреса:

- `http://host/base` – это обычный URL, по которому выполняется доступ, например, к информационной базе с помощью веб-клиента. Необходимо помнить, что при использовании информационной базы, в которой настроено разделение данных, значения разделителей можно указывать только в URL информационной базы. Указание разделителей с помощью параметра Z не поддерживается;
- `hs` – признак того, что выполняется обращение к HTTP-сервису (в отличие от `ws`, который определяет доступ к Web-сервису);
- `<корневойURL>` – имя ресурса, которое определяет группу ресурсов, связанных общим смыслом. Задается в свойстве `КорневойURL` объекта `HTTP-сервис`;
- `<относительныйURL>` – определяет ресурс, к которому будет выполняться обращение. Относительный URL, указанный в запросе, будет использован для определения конкретного ресурса, к которому выполняется обращение. Правило сопоставления задается в свойстве `Шаблон` объекта `Шаблон URL`, подчиненного объекту `HTTP-сервис`.

ПОДРОБНЕЕ

Более подробно HTTP-сервисы описаны в документации «1С:Предприятия» в разделе «Глава 17. Механизмы интернет-сервисов – HTTP-сервисы – HTTP-сервисы».

Разработка HTTP-сервиса

Для создания HTTP-сервисов, которые будет поставлять прикладное решение, есть специальный объект конфигурации `HTTP-сервис`. Такие объекты добавляются в ветку Общие > HTTP-сервисы.

При создании HTTP-сервиса следует установить свойство Корневой URL. Оно определяет группу ресурсов, объединенных общим смыслом. Например, если необходимо создать несколько ресурсов, которые работают с заказами, то корневой URL в этом случае может выглядеть как `order`. Тогда начало URL при обращении к такому сервису будет выглядеть следующим образом: `http://host/base/hs/order`.

Затем в каждом сервисе необходимо создать определенный шаблон URL (группу ресурсов). Таких групп ресурсов может быть несколько. При создании шаблона URL в свойстве Шаблон описываются возможные адреса ресурсов, которые можно будет использовать для обращения к HTTP-сервису. При создании шаблона можно использовать следующие символы:

- любые символы, допустимые в идентификаторах языка «1С:Предприятия»;
- символ «/»;
- символы «{}» с непустым текстом между ними;
- символ *.

Например, шаблон может выглядеть следующим образом:

- `/query`
- `/documents/{id}/props/{PropertyName}/*`

В последнем примере `props` – это непараметризованный сегмент, который должен быть перенесен в URL дословно. А заключенный в фигурные скобки `{PropertyName}` – параметризованный сегмент, который фактически описывает переменную с указанным именем, к которой можно получить доступ из встроенного языка.

И затем для созданного шаблона необходимо определить HTTP-методы, которые могут быть использованы при работе с ресурсом, совпадающим с шаблоном. Для этого необходимо создать нужное количество объектов `Метод`, подчиненных объекту `Шаблон URL`. Для каждого метода необходимо указать свойство `HTTP-метод` и создать обработчик. В качестве HTTP-метода можно выбрать один из следующих методов: GET, POST, PUT, DELETE, PATCH, MERGE, CONNECT, OPTIONS, TRACE, PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK. Или можно выбрать значение Любой – в этом случае можно будет использовать любой метод из приведенного списка.

При обращении к HTTP-сервису платформа сначала попытается сопоставить URL, по которому произошло обращение, с одним из имеющихся шаблонов и методов. Если сопоставить не удалось, то платформа выдаст код ответа: 404 – «Not Found». Если подходящий метод будет найден, то платформа начнет выполнение его обработчика.

Обработчиком того или иного HTTP-метода является функция, которая получает на вход один параметр типа `HTTPСервисЗапрос` и должна вернуть объект типа `HTTPСервисОтвет`.

Входной параметр полностью описывает HTTP-запрос, поступивший в систему. С помощью объекта `HTTPСервисЗапрос` можно определить:

- какой HTTP-метод используется в запросе (свойство HTTPМетод);
- на какой URL поступил запрос (свойство БазовыйURL) – часть полного URL запроса, не включающая относительный URL и параметры запроса;
- относительный URL (свойство ОтносительныйURL) – часть полного URL запроса, которая использовалась для определения группы ресурсов и правил обработки входящего HTTP-запроса;
- какие заголовки (свойство Заголовки) содержит HTTP-запрос;
- какие параметризованные сегменты (свойство ПараметрыURL) выделены из входящего HTTP-запроса и их значения;
- какие параметры (свойство ПараметрыЗапроса) перечислены в URL HTTP-запроса после символа «?» и их значения.

Также существует возможность получать тело HTTP-запроса в виде двоичных данных или строки (в зависимости от передаваемой информации).

После того как прикладное решение выполнило все действия, которые определены для данного ресурса и HTTP-метода, необходимо сформировать ответ сервиса. Для этого необходимо создать объект типа `HTTPСервисОтвет`. В зависимости от результата обработки следует указать свойство КодСостояния, которое описывает стандартный код возврата HTTP. Затем, при необходимости, следует сформировать тело ответа HTTP-сервиса. Тело может быть установлено как двоичные данные, как строка (в формате JSON или XML) или с помощью указания имени файла, откуда будет загружено тело ответа.

Примеры реализации HTTP-сервисов

Рассмотрим все вышесказанное на конкретных примерах. Создадим два HTTP-сервиса, с помощью которых мы будем получать информацию о сотрудниках и информацию о ценах товаров из нашей демонстрационной базы. Расположим примеры по мере усложнения и прокомментируем.

ПОДРОБНЕЕ

Подробнее познакомиться с реализацией и использованием HTTP-сервисов можно в демонстрационной конфигурации «Примеры работы», которая прилагается к книге.

Информация о сотрудниках

Итак, в ветке Общие > HTTP-сервисы добавим HTTP-сервис с именем Сотрудники, который будет получать информацию из справочника Сотрудники. Свойство Корневой URL для сервиса установим как `employees` (рис. 1.11).

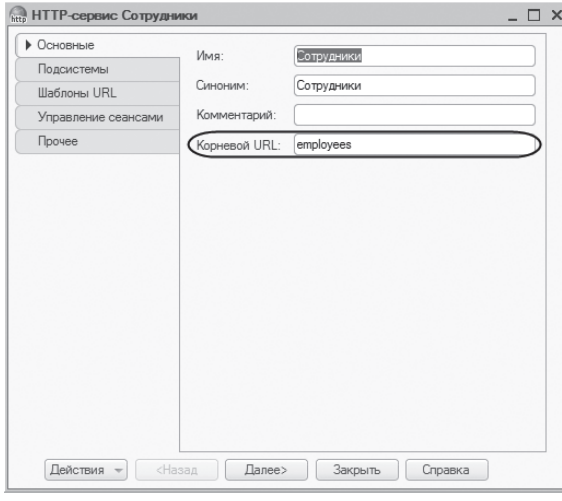


Рис. 1.11. Основные свойства HTTP-сервиса «Сотрудники»

Поскольку в дальнейшем наш HTTP-сервис будет опубликован на локальном сервере в каталоге «REST», то базовая часть URL для всех шаблонов URL (групп ресурсов) при обращении к сервису будет выглядеть как `http://localhost/REST/hs/employees`.

Предположим, нам нужно, чтобы при обращении по одному URL наш сервис возвращал бы список всех сотрудников из справочника Сотрудники (возможно, с отбором по признаку работы), а при обращении по другому URL мы получали бы все данные о конкретном сотруднике. И, кроме того, мы хотели бы иметь возможность удаления конкретных сотрудников из справочника.

Для этого нам потребуется создать у нашего HTTP-сервиса два подчиненных объекта `Шаблон URL` с соответствующими шаблонами, с помощью которых можно было бы сопоставить URL, по которому произошло обращение к сервису, с одним из имеющихся HTTP-методов.

Итак, на закладке Шаблоны URL добавим у нашего HTTP-сервиса шаблон URL с именем Список (свойство которого Шаблон оставим по умолчанию в значении `/*`) и шаблон URL Сотрудник с шаблоном `/code/{Код}` (рис. 1.12).

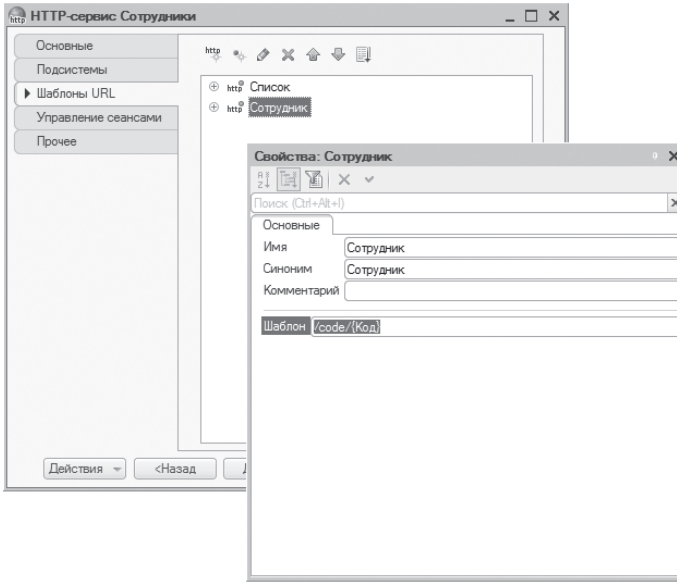


Рис. 1.12. Шаблоны URL HTTP-сервиса «Сотрудники»

Ниже мы разберем каждый шаблон URL более подробно.

Затем для каждого шаблона URL добавим метод, который будет вызван при сопоставлении этого шаблона с URL запроса к HTTP-сервису. У каждого метода выберем нужный HTTP-метод (GET, POST, PUT и т. п.) и создадим его обработчик (рис. 1.13).

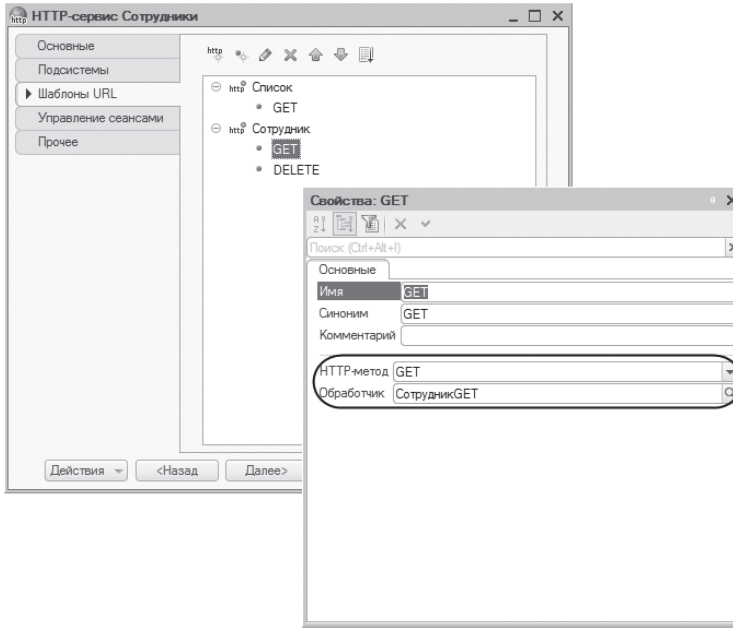


Рис. 1.13. Шаблоны URL HTTP-сервиса «Сотрудники»

Получить список сотрудников

Итак, чтобы получить список сотрудников с возможным отбором из справочника Сотрудники, добавим у нашего HTTP-сервиса объект `Шаблон URL` Список с шаблоном `/*`. Затем у этого объекта добавим подчиненный объект `Метод` GET, в качестве HTTP-метода оставим предложенное по умолчанию значение GET. Далее, нажав на кнопку открытия в поле Обработчик, создадим обработчик этого метода (см. рис. 1.13). Имя обработчика автоматически формируется путем соединения имени шаблона URL и имени самого метода. Это имя можно изменить в дальнейшем по своему желанию, хотя делать это не рекомендуется.

В модуле нашего HTTP-сервиса будет создан шаблон функции-обработчика `СписокGET()`, которая получает на вход параметр типа `HTTPСервисЗапрос` и должна вернуть объект типа `HTTPСервисОтвет`. Заполним эту функцию следующим образом (листинг 1.56).

Листинг 1.56. Функция «СписокGET»

```
Функция СписокGET(Запрос)

	// Сформировать ответ, возвращаемый HTTP-сервисом.
	Ответ = Новый HTTPСервисОтвет(200);

	// Получить из запроса параметр URL *.
	Признак = Запрос.ПараметрыURL["*"];

	// Сформировать выборку сотрудников.
	Если Признак = "" Тогда
		// Без отбора.
		Выборка = Справочники.Сотрудники.Выбрать();
	Иначе
		// С отбором по признаку работы.
		Работает = ?(ВРег(Прав(Признак,2)) = "ДА", Истина, Ложь);
		Выборка = Справочники.Сотрудники.Выбрать(,, Новый Структура("Работает", Работает));
	КонецЕсли;

	// В цикле обхода выборки записать в JSON список сотрудников.
	// Создать объект записи и записать строковое значение в строку JSON.
	Запись = Новый ЗаписьJSON;
	Запись.УстановитьСтроку();

	// Записать начало корневого объекта.
	Запись.ЗаписатьНачалоОбъекта();
	Пока Выборка.Следующий() Цикл
		Запись.ЗаписатьИмяСвойства(Выборка.Наименование);
		Запись.ЗаписатьНачалоОбъекта();
		Запись.ЗаписатьИмяСвойства("Код");
		Запись.ЗаписатьЗначение(Выборка.Код);
		Запись.ЗаписатьИмяСвойства("Должность");
		Запись.ЗаписатьЗначение(Строка(Выборка.Должность));
		Запись.ЗаписатьКонецОбъекта();
	КонецЦикла;
	// Записать конец корневого объекта.
	Запись.ЗаписатьКонецОбъекта();

	// Записать результат записи в строку JSON.
	Результат = Запись.Закрыть();

	// Установить тело ответа из строки Результат.
	Ответ.УстановитьТелоИзСтроки(Результат);
	Ответ.Заголовки.Вставить("Content-type", "application/json");

	Возврат Ответ;

КонецФункции
```

Прокомментируем код функции.

В переменной Ответ мы сразу же создаем объект типа HTTPСервисОтвет, который содержит ответ, формируемый HTTP-сервером на поступивший запрос. Ответ имеет стандартный код возврата HTTP (КодСостояния=200). Код состояния 200 сообщает об успешном выполнении запроса. Поскольку в функции мы описываем простейшее действие – получение выборки данных из справочника, то для простоты мы не будем здесь анализировать ошибочные состояния, которые могут возникнуть при выполнении этого обработчика.

ПОДРОБНЕЕ

> Подробнее о кодах состояния, возвращаемых HTTP-сервером, рассказывается в разделе «Коды состояния в ответах HTTP-сервера».

Родительский объект функции-обработчика – это шаблон URL Список, который содержит строку шаблона /*. Этот шаблон означает, что в относительном URL на месте звездочки могут оказаться любые допустимые символы, в том числе возможно и отсутствие этих символов.

Поэтому мы получаем и анализируем те символы, которые в реальном URL присутствуют на месте звездочки. Для этого мы получаем параметры URL запроса, который передается в функцию в параметре Запрос с помощью конструкции Запрос.ПараметрыURL[«*»].

Если относительный URL пустой, то мы формируем выборку из справочника Сотрудники без отбора. Если на месте звездочки находится строка «ДА», то она интерпретируется как Истина, любые другие символы интерпретируется как Ложь. Соответственно этому признаку накладывается отбор на поле справочника Работает.

В цикле обхода выборки мы заполняем строку JSON списком сотрудников и затем записываем эту строку в переменную Результат. После этого с помощью метода УстановитьТелоИзСтроки() устанавливаем тело ответа HTTP-сервиса из строки Результат и возвращаем ответ сервиса в переменной Ответ.

При этом заголовок «Content-type» объекта HTTPСервисОтвет мы устанавливаем как «application/json». Такой заголовок помогает клиенту понять, что же за данные к нему пришли, кроме того, из него определяется кодировка ответа, если она не задана явно.

Теперь посмотрим, как это работает.

Прежде всего нам нужно опубликовать наш HTTP-сервис на веб-сервере. Для этого в диалоге публикации информационной базы на веб-сервере на закладке HTTP сервисы установим флажок Публиковать HTTP сервисы

по умолчанию и отметим имеющиеся HTTP-сервисы, к которым мы хотим предоставить доступ из внешних систем (рис. 1.14).

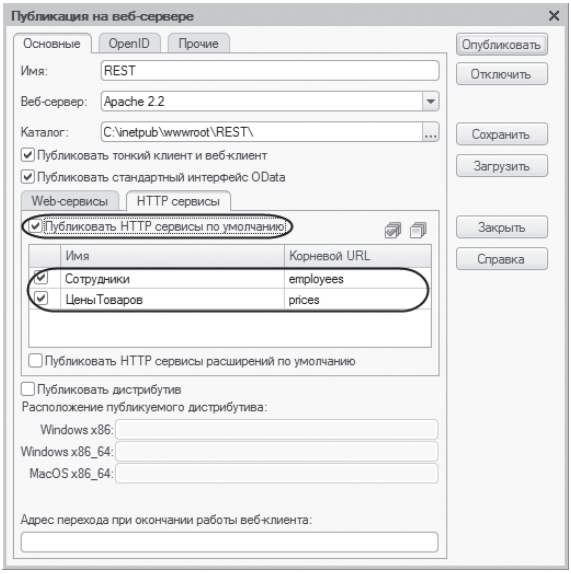


Рис. 1.14. Публикация HTTP-сервиса на веб-сервере

И все, больше делать ничего не надо.

Поскольку в большинстве наших примеров мы будем просто получать данные информационной базы методом GET, то URL запроса к HTTP-сервису будем набирать непосредственно в адресной строке браузера и сразу же в окне браузера будем видеть содержимое ответа сервиса.

Заметим, что в этом разделе таким способом (получение данных через браузер) мы будем пользоваться только для упрощения примеров. На самом же деле к подобным HTTP-сервисам обращаются с помощью HTTP-запросов. Про то, как в «1С:Предприятии» написать собственный HTTP-запрос и с его помощью обратиться к HTTP-сервису, будет рассказано в следующем разделе «HTTP-запросы».

Итак, наберем в адресной строке браузера следующий URL (листинг 1.57).

Листинг 1.57. URL запроса

```
http://localhost/REST/hs/employees
```

Как видите, для получения списка сотрудников из справочника мы указываем только базовую часть URL запроса к HTTP-сервису, относительная часть URL отсутствует. Поэтому такой URL будет сопоставлен с шаблоном URL Список (строка шаблона – /*) и будет вызван обработчик этого шаблона СписокGET(). В результате в окне браузера мы увидим список всех сотрудников из справочника Сотрудники (листинг 1.58).

Листинг 1.58. Содержимое ответа сервиса

```
{
"Алексеев Сергей Иванович": {
"Код": "000000001",
"Должность": "Администратор"
},
"Артемов Игорь Владимирович": {
"Код": "REST-0003",
"Должность": "Новая должность - REST"
},
"Смирнова Светлана Ивановна": {
"Код": "000000002",
"Должность": "Кассир"
}
}
```

Теперь наберем в адресной строке браузера следующий URL (листинг 1.59).

Листинг 1.59. URL запроса

```
http://localhost/REST/hs/employees/да
```

Такой URL будет сопоставлен с шаблоном URL Список (строка шаблона – /*) и будет вызван обработчик этого шаблона СписокGET(). В результате в окне браузера мы увидим список только работающих сотрудников (листинг 1.60).

Листинг 1.60. Содержимое ответа сервиса

```
{
"Алексеев Сергей Иванович": {
"Код": "000000001",
"Должность": "Администратор"
},
"Артемов Игорь Владимирович": {
"Код": "REST-0003",
"Должность": "Новая должность - REST"
}
}
```

Теперь наберем в адресной строке браузера следующий URL (листинг 1.61).

Листинг 1.61. URL запроса

```
http://localhost/REST/hs/employees/нет
```

В результате в окне браузера мы увидим список только неработающих сотрудников (листинг 1.62).

Листинг 1.62. Содержимое ответа сервиса

```
{
"Смирнова Светлана Ивановна": {
"Код": "000000002",
"Должность": "Кассир"
}
}
```

Получить данные о конкретном сотруднике

В этом примере, чтобы получить данные о конкретном сотруднике, нам понадобится воспользоваться другим шаблоном URL нашего HTTP-сервиса. В относительном URL запроса при обращении к сервису должно быть указано некое уникальное свойство (например, Код), по которому сотрудника можно было бы идентифицировать в справочнике. В обработчике, связанном с этим шаблоном, данные о конкретном сотруднике должны быть записаны в строку JSON и подставлены в тело ответа, возвращаемого сервисом.

Итак, добавим у нашего HTTP-сервиса объект `Шаблон URL` Сотрудник с шаблоном `/code/{Код}`. Затем у этого объекта добавим подчиненный метод GET, в качестве HTTP-метода оставим предложенное по умолчанию значение GET. Далее создадим обработчик этого метода.

В модуле нашего HTTP-сервиса будет создан шаблон функции-обработчика `СотрудникGET()`. Заполним эту функцию следующим образом (листинг 1.63).

Листинг 1.63. Функция «СотрудникGET»

```
Функция СотрудникGET(Запрос)

	// Сформировать ответ, возвращаемый HTTP-сервисом.
	Ответ = Новый HTTPСервисОтвет(200);

	// Получить из запроса параметр URL Код.
	Код = Запрос.ПараметрыURL.Получить("Код");
	Если Код = Неопределено Тогда
		Ответ = Новый HTTPСервисОтвет(400);
		Ответ.УстановитьТелоИзСтроки("Не задан параметр Код");
		Ответ.Заголовки.Вставить("Content-type", "application/json");
	Возврат Ответ;
```

```
    КонецЕсли;

    // Найти сотрудника в справочнике по полученному параметру Код.
    СотрудникСсылка = Справочники.Сотрудники.НайтиПоКоду(Код);
    Если СотрудникСсылка = Справочники.Сотрудники.ПустаяСсылка() Тогда
        Ответ = Новый HTTPСервисОтвет(404);
        Ответ.УстановитьТелоИзСтроки("Сотрудник не найден");
        Ответ.Заголовки.Вставить("Content-type", "application/json");
    Возврат Ответ;
    КонецЕсли;

    Сотрудник = СотрудникСсылка.ПолучитьОбъект();

    // Сериализовать данные объекта Сотрудник с помощью объекта записи (Запись).
    ПараметрыЗаписиJSON = Новый ПараметрыЗаписиJSON(ПереносСтрокJSON.Авто, Символы.Таб);
    Запись = Новый ЗаписьJSON;
    Запись.УстановитьСтроку(ПараметрыЗаписиJSON);
    СериализаторXDTO.ЗаписатьJSON(Запись, Сотрудник);

    // Записать результат записи в строку JSON.
    Результат = Запись.Закрыть();

    // Установить тело ответа из строки Результат.
    Ответ.УстановитьТелоИзСтроки(Результат);
    Ответ.Заголовки.Вставить("Content-type", "application/json");

    Возврат Ответ;

КонецФункции
```

Прокомментируем код функции.

Родительский объект функции-обработчика – шаблон URL Сотрудник, описанный с помощью шаблона `/code/{Код}`. Этот шаблон содержит непараметризованный сегмент `code` и параметризованный сегмент `{Код}`. Это означает, что любые символы, которые могут присутствовать в URL запроса к сервису после символов «/code/», будут трактоваться как параметр URL `Код`, который можно получить из переданного в функцию параметра `Запрос`.

Поэтому сначала в обработчике мы получаем значение параметра `Код` (`Запрос.ПараметрыURL.Получить(«Код»)`) и анализируем его. Если этот параметр URL не определен, то мы прекращаем работу функции и возвращаем ответ сервиса с кодом состояния 400, сообщающим об ошибке запроса.

Затем находим сотрудника в справочнике Сотрудники по значению параметра `Код`. Если сотрудник не найден, то мы прекращаем работу функции и возвращаем ответ сервиса с кодом состояния 404, сообщающим об этом.

От ссылки на найденного сотрудника мы получаем объект и сериализуем данные сотрудника в JSON с помощью метода `ЗаписатьJSON()` объекта глобального контекста `СериализаторXDTO`. Результат записи сохраняем в строку JSON в переменную `Результат`. После этого с помощью метода `УстановитьТелоИзСтроки()` устанавливаем тело ответа HTTP-сервиса из строки `Результат` и возвращаем ответ сервиса в переменной `Ответ`.

Теперь посмотрим, как это работает. Наберем в адресной строке браузера следующий URL (листинг 1.64).

Листинг 1.64. URL запроса

```
http://localhost/REST/hs/employees/code/000000001
```

В URL запроса к HTTP-сервису после символов «/code/» присутствует строковый код сотрудника (000000001). Поэтому такой URL будет сопоставлен с шаблоном URL Сотрудник (строка шаблона – `/code/{Код}`) и будет вызван обработчик этого шаблона `СотрудникGET()`. В результате в окне браузера мы увидим содержимое ответа сервиса (листинг 1.65).

Листинг 1.65. Содержимое ответа сервиса

```
{
	"#value": {
		"Ref": "9575f8ed-369f-11e9-8bb2-642737df2048",
		"DeletionMark": false,
		"Code": "000000001",
		"Description": "Алексеев Сергей Иванович",
		"ДатаРождения": "1980-12-10T00:00:00",
		"КоличествоДетей": 1,
		"Работает": true,
		"Стаж": "10.05.01",
		"Должность": "88d54406-36a1-11e9-8bb2-642737df2048"
	}
}
```

Удалить данные о конкретном сотруднике

В этом примере, чтобы иметь возможность удалить данные о конкретном сотруднике, мы добавим еще один метод для ранее созданного шаблона URL Сотрудник нашего HTTP-сервиса.

Итак, добавим этому шаблону еще один подчиненный метод DELETE, в качестве HTTP-метода выберем значение DELETE. Далее создадим обработчик этого метода.

В модуле нашего HTTP-сервиса будет создан шаблон функции-обработчика `СотрудникDELETE()`. Заполним эту функцию следующим образом (листинг 1.66).

Листинг 1.66. Функция «СотрудникDELETE»

```
Функция СотрудникDELETE(Запрос)

	// Сформировать ответ (без тела), возвращаемый HTTP-сервисом.
	Ответ = Новый HTTPСервисОтвет(204);

	// Получить из запроса параметр URL Код.
	Код = Запрос.ПараметрыURL.Получить("Код");
	Если Код = Неопределено Тогда
		Ответ = Новый HTTPСервисОтвет(400);
		Ответ.УстановитьТелоИзСтроки("Не задан параметр Код");
		Ответ.Заголовки.Вставить("Content-type", "application/json");
	Возврат Ответ;
	КонецЕсли;

	// Найти сотрудника в справочнике по полученному параметру Код.
	СотрудникСсылка = Справочники.Сотрудники.НайтиПоКоду(Код);
	Если СотрудникСсылка = Справочники.Сотрудники.ПустаяСсылка() Тогда
		Ответ = Новый HTTPСервисОтвет(404);
		Ответ.УстановитьТелоИзСтроки("Сотрудник не найден");
		Ответ.Заголовки.Вставить("Content-type", "application/json");
	Возврат Ответ;
	КонецЕсли;

	// Установить найденному сотруднику пометку удаления.
	Сотрудник = СотрудникСсылка.ПолучитьОбъект();
	Сотрудник.УстановитьПометкуУдаления(Истина);
	Сотрудник.Записать();

	Возврат Ответ;

КонецФункции
```

Поиск сотрудника по значению параметра URL запроса аналогичен поиску в обработчике `СотрудникGET()` из предыдущего примера (см. листинг 1.63). Затем от ссылки на найденного сотрудника мы получаем объект и устанавливаем ему пометку удаления.

И в заключение возвращаем ответ HTTP-сервиса с кодом состояния 204, который также свидетельствует об успешном выполнении операции на сервере, но сообщает клиенту, что сервер не посылал в ответ никаких данных в теле запроса.

Испытать наш запрос в браузере не получится, так как из браузера отправляются только запросы на чтение (методом `GET`) данных из информационной базы. Чтобы посмотреть результат работы HTTP-сервиса по удалению

данных о конкретном сотруднике в прикладном решении, надо обратиться к нему с помощью собственного HTTP-запроса и выполнить на сервере метод DELETE. Этот пример будет рассмотрен в следующем разделе «HTTP-запросы – Удаление данных».

Информация о ценах на товары

В этом примере мы рассмотрим, как с помощью HTTP-сервиса получать информацию о ценах товаров, которые в нашей демонстрационной конфигурации хранятся в регистре сведений ЦеныТоваров.

Для этого в ветке Общие > HTTP-сервисы добавим HTTP-сервис с именем ЦеныТоваров, свойство которого Корневой URL установим как prices.

Поскольку в дальнейшем наш HTTP-сервис будет опубликован на локальном сервере в каталоге «REST», то базовая часть URL для всех шаблонов URL (групп ресурсов) при обращении к сервису будет выглядеть как http://localhost/REST/hs/prices.

Предположим, нам нужно, чтобы при обращении по одному URL наш сервис возвращал бы список цен всех товаров, актуальных на определенную дату, а при обращении по другому URL мы получали бы последнюю установленную цену на конкретный товар. И, кроме того, мы хотели бы иметь возможность индексации последних цен всех товаров на заданную величину.

Для этого на закладке Шаблоны URL добавим у нашего HTTP-сервиса шаблон URL ПолучитьЦены с шаблоном /date/{Дата}, шаблон URL ПоказатьЦенуТовара с шаблоном /product/{Код} и шаблон URL ПроиндексироватьЦены с шаблоном /percent/{Процент}.

Затем для каждого шаблона URL добавим метод, который будет вызван при сопоставлении этого шаблона с URL запроса к HTTP-сервису. Для методов ПолучитьЦены и ПоказатьЦенуТовара выберем HTTP-метод GET, а для метода ПроиндексироватьЦены выберем HTTP-метод POST (рис. 1.15).

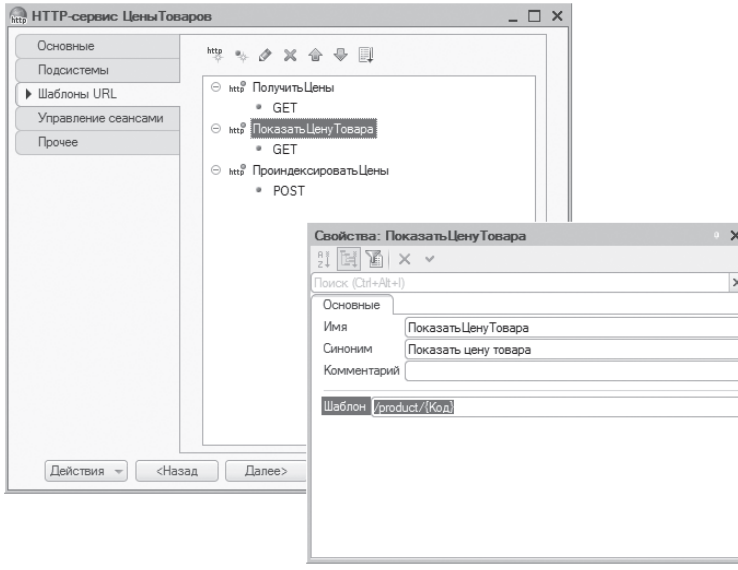


Рис. 1.15. Шаблоны URL и методы HTTP-сервиса «Цены товаров»

Получить цены товаров на определенную дату

В этом примере, чтобы получить актуальные цены товаров на заданную дату, нам понадобится получить срез последних записей на эту дату из регистра сведений ЦеныТоваров и подставить список товаров их цен в тело ответа, возвращаемого сервисом.

Итак, добавим у нашего HTTP-сервиса объект `Шаблон URL` ПолучитьЦены с шаблоном `/date/{Дата}`. Затем у этого объекта добавим подчиненный метод GET, в качестве HTTP-метода оставим предложенное по умолчанию значение GET. Далее создадим обработчик этого метода.

В модуле нашего HTTP-сервиса будет создан шаблон функции-обработчика `ПолучитьЦеныGET()`. Заполним эту функцию следующим образом (листинг 1.67).

Листинг 1.67. Функция «ПолучитьЦеныGET»

```
Функция ПолучитьЦеныGET(Запрос)

    Попытка

        // Получить из запроса параметр URL Дата.
        ДатаЗапроса = Запрос.ПараметрыURL.Получить("Дата");
```

```
    Если ДатаЗапроса = Неопределено Тогда
        Ответ = Новый HTTPСервисОтвет(400);
        Ответ.УстановитьТелоИзСтроки("Не задан параметр Дата");
        Ответ.Заголовки.Вставить("Content-type", "application/json");
        Возврат Ответ;
    КонецЕсли;

    ЦеныНаДату = Дата(ДатаЗапроса);
    // Сформировать запрос к регистру сведений.
    Запрос = Новый Запрос;
    Запрос.Текст = "ВЫБРАТЬ
    |    ЦеныТоваровСрезПоследних.Период КАК Период,
    |    ЦеныТоваровСрезПоследних.Товар КАК Товар,
    |    ЦеныТоваровСрезПоследних.Цена КАК Цена
    |ИЗ
    |    РегистрСведений.ЦеныТоваров.СрезПоследних(&ЦеныНаДату, )
    |                                        КАК ЦеныТоваровСрезПоследних";

    Запрос.УстановитьПараметр("ЦеныНаДату", ЦеныНаДату);
    Выборка = Запрос.Выполнить().Выбрать();

    // В цикле обхода выборки записать в JSON последние цены товаров
    // Создать объект записи и записать строковое значение в строку JSON.
    Запись = Новый ЗаписьJSON;
    Запись.УстановитьСтроку();

    // Записать начало корневого объекта.
    Запись.ЗаписатьНачалоОбъекта();
    Пока Выборка.Следующий() Цикл
        Запись.ЗаписатьИмяСвойства(Строка(Выборка.Товар));
        Запись.ЗаписатьНачалоОбъекта();
        Запись.ЗаписатьИмяСвойства("Период");
        Запись.ЗаписатьЗначение(Строка(Выборка.Период));
        Запись.ЗаписатьИмяСвойства("Цена");
        Запись.ЗаписатьЗначение(Выборка.Цена);
        Запись.ЗаписатьКонецОбъекта();
    КонецЦикла;
    // Записать конец корневого объекта.
    Запись.ЗаписатьКонецОбъекта();

    // Записать результат записи в строку JSON.
    Результат = Запись.Закрыть();

    // Сформировать ответ, возвращаемый HTTP-сервисом.
    Ответ = Новый HTTPСервисОтвет(200);

    // Установить тело ответа из строки Результат.
    Ответ.УстановитьТелоИзСтроки(Результат);

Исключение

    // Вывести стуктурированную информацию об исключении.
    Ответ = Новый HTTPСервисОтвет(500);
    Информация = ИнформацияОбОшибке();
    Сообщение = Информация.Описание;
```

```
            Если Информация.Причина <> Неопределено Тогда
                Сообщение = Сообщение + ":" + Информация.Причина.Описание;
            КонецЕсли;
            Ответ.УстановитьТелоИзСтроки(Сообщение);

        КонецПопытки;

        Ответ.Заголовки.Вставить("Content-type", "application/json");
        Возврат Ответ;

КонецФункции
```

Прокомментируем код функции.

Родительский объект функции-обработчика – шаблон URL ПолучитьЦены, описанный с помощью шаблона `/date/{Дата}`. Этот шаблон содержит непараметризованный сегмент `date` и параметризованный сегмент `{Дата}`. Это означает, что любые символы, которые могут присутствовать в URL запроса к сервису после символов «/date/», будут трактоваться как параметр URL `Дата`, который можно получить из переданного в функцию параметра `Запрос`.

Поэтому сначала в обработчике мы делаем попытку получить значение параметра `Дата` (`Запрос.ПараметрыURL.Получить("Дата")`) и анализируем его. Если этот параметр URL не определен, то мы прекращаем работу функции и возвращаем ответ сервиса с кодом состояния 400, сообщающим об ошибке запроса.

Затем мы преобразуем строковое значение параметра `Дата` в дату `ЦеныНаДату`, которую подставляем в качестве параметра в запрос, обращающийся к срезу последних регистра ЦеныТоваров.

В цикле обхода выборки мы заполняем строку JSON списком последних цен товаров на дату, переданную в параметре запроса, и затем записываем эту строку в переменную `Результат`. После этого создаем объект `HTTPСервисОтвет` с кодом состояния 200, с помощью метода `УстановитьТелоИзСтроки()` устанавливаем тело ответа HTTP-сервиса из строки `Результат` и возвращаем ответ сервиса в переменной `Ответ`.

При этом заголовок «Content-type» объекта `HTTPСервисОтвет` мы устанавливаем как «application/json». Такой заголовок помогает клиенту понять, что же за данные к нему пришли, кроме того, из него определяется кодировка ответа, если она не задана явно.

Если же при попытке получить значение параметра URL из запроса возникает ошибка, то генерируется исключение. С помощью метода глобального контекста `ИнформацияОбОшибке()` мы получаем структурированную

информацию об исключении и возвращаем ответ сервиса с кодом состояния 500, сообщающим о причине ошибки.

Теперь посмотрим, как это работает.

Прежде всего опубликуем наш HTTP-сервис на веб-сервере. О том, как это делается, уже рассказывалось в примере про HTTP-сервис Сотрудники (см. рис. 1.14).

После этого наберем в адресной строке браузера следующий URL (листинг 1.68).

Листинг 1.68. URL запроса

```
http://localhost/REST/hs/prices/date/20190510
```

В URL запроса к HTTP-сервису после символов «/date/» присутствует строковое представление даты – 10/05/2019. Поэтому такой URL будет сопоставлен с шаблоном URL ПолучитьЦены (строка шаблона – `/date/{Дата}`) и будет вызван обработчик этого шаблона `ПолучитьЦеныGET()`. В результате в окне браузера мы увидим содержимое ответа сервиса (листинг 1.69).

Листинг 1.69. Содержимое ответа сервиса

```
{
"Чайник": {
"Период": "07.05.2019 0:00:00",
"Цена": 3000
},
"Тостер": {
"Период": "07.05.2019 0:00:00",
"Цена": 2000
},
"Кофемашина": {
"Период": "07.05.2019 0:00:00",
"Цена": 5000
}
}
```

Получить данные о ценах на конкретный товар

В этом примере, чтобы получить последнюю установленную цену на конкретный товар, нам понадобится воспользоваться другим шаблоном URL нашего HTTP-сервиса. В относительном URL запроса при обращении к сервису должно быть указано некое уникальное свойство (например, Код), по которому товар можно было бы идентифицировать в справочнике товаров. В обработчике, связанном с этим шаблоном, нужно получить последнюю цену товара из регистра сведений ЦеныТоваров и подставить ее в тело ответа, возвращаемого сервисом.

Итак, добавим у нашего HTTP-сервиса объект `Шаблон URL` ПоказатьЦенуТовара с шаблоном `/product/{Код}`. Затем у этого объекта добавим подчиненный метод GET, в качестве HTTP-метода оставим предложенное по умолчанию значение GET. Далее создадим обработчик этого метода.

В модуле нашего HTTP-сервиса будет создан шаблон функции-обработчика `ПоказатьЦенуТовараGET()`. Заполним эту функцию следующим образом (листинг 1.70).

Листинг 1.70. Функция «ПоказатьЦенуТовараGET»

```
Функция ПоказатьЦенуТовараGET(Запрос)

    Попытка

        // Получить из запроса параметр URL Код.
        Код = Запрос.ПараметрыURL.Получить("Код");
        Если Код = Неопределено Тогда
            Ответ = Новый HTTPСервисОтвет(400);
            Ответ.УстановитьТелоИзСтроки("Не задан параметр Код");
            Ответ.Заголовки.Вставить("Content-type", "application/json");
            Возврат Ответ;
        КонецЕсли;

        // Найти товар в справочнике по полученному параметру Код.
        Товар = Справочники.Товары.НайтиПоКоду(Код);
        Если Товар = Справочники.Товары.ПустаяСсылка() Тогда
            Ответ = Новый HTTPСервисОтвет(400);
            Ответ.УстановитьТелоИзСтроки("Товар не найден");
            Ответ.Заголовки.Вставить("Content-type", "application/json");
            Возврат Ответ;
        КонецЕсли;

        // Получить последнюю цену товара из регистра сведений.
        ЦенаТовара = РегистрыСведений.ЦеныТоваров.ПолучитьПоследнее(
                                                    , Новый Структура("Товар", Товар));

        // Создать объект записи и записать строковое значение в строку JSON.
        Запись = Новый ЗаписьJSON;
        Запись.УстановитьСтроку();

        // Записать начало корневого объекта.
        Запись.ЗаписатьНачалоОбъекта();
        Запись.ЗаписатьИмяСвойства("На: ");
        Запись.ЗаписатьЗначение(Строка(ТекущаяДата()));
        Запись.ЗаписатьИмяСвойства("Товар");
        Запись.ЗаписатьЗначение(Строка(Товар));
        Запись.ЗаписатьИмяСвойства("Цена");
        Запись.ЗаписатьЗначение(ЦенаТовара.Цена);
        // Записать конец корневого объекта.
        Запись.ЗаписатьКонецОбъекта();

        // Записать результат записи в строку JSON.
```

```
        Результат = Запись.Закрыть();

        // Сформировать ответ, возвращаемый HTTP-сервисом.
        Ответ = Новый HTTPСервисОтвет(200);

        // Установить тело ответа из строки Результат.
        Ответ.УстановитьТелоИзСтроки(Результат);

    Исключение

        // Вывести стуктурированную информацию об исключении.
        Ответ = Новый HTTPСервисОтвет(500);
        Информация = ИнформацияОбОшибке();
        Сообщение = Информация.Описание;
        Если Информация.Причина <> Неопределено Тогда
            Сообщение = Сообщение + ":" + Информация.Причина.Описание;
        КонецЕсли;
        Ответ.УстановитьТелоИзСтроки(Сообщение);

    КонецПопытки;

    Ответ.Заголовки.Вставить("Content-type", "application/json");
    Возврат Ответ;

КонецФункции
```

Прокомментируем код функции.

Родительский объект функции-обработчика – шаблон URL ПоказатьЦенуТовара, описанный с помощью шаблона `/product/{Код}`. Этот шаблон содержит непараметризованный сегмент `product` и параметризованный сегмент `{Код}`. Это означает, что любые символы, которые могут присутствовать в URL запроса к сервису после символов «/product/», будут трактоваться как параметр URL `Код`, который можно получить из переданного в функцию параметра `Запрос`.

Поэтому сначала в обработчике мы делаем попытку получить значение параметра `Код` (`Запрос.ПараметрыURL.Получить("Код")`) и анализируем его. Если этот параметр URL не определен, то мы прекращаем работу функции и возвращаем ответ сервиса с кодом состояния 400, сообщающим об ошибке запроса.

Затем находим товар в справочнике Товары по значению параметра `Код`. Если товар не найден, то мы прекращаем работу функции и возвращаем ответ сервиса с кодом состояния 404, сообщающим об этом.

После этого с помощью метода `ПолучитьПоследнее()` с отбором по найденному товару мы получаем последнюю цену этого товара из регистра сведений ЦеныТоваров и записываем ее в JSON-строку `Результат`.

И затем создаем объект HTTPСервисОтвет с кодом состояния 200, с помощью метода УстановитьТелоИзСтроки() устанавливаем тело ответа HTTP-сервиса из строки Результат и возвращаем ответ сервиса в переменной Ответ.

Если же при попытке получить значение параметра URL из запроса возникает ошибка, то генерируется исключение. С помощью метода глобального контекста ИнформацияОбОшибке() мы получаем структурированную информацию об исключении и возвращаем ответ сервиса с кодом состояния 500, сообщающим о причине ошибки.

Теперь посмотрим, как это работает. Наберем в адресной строке браузера следующий URL (листинг 1.71).

Листинг 1.71. URL запроса

```
http://localhost/REST/hs/prices/product/000000001
```

В URL запроса к HTTP-сервису после символов «/product/» присутствует строковой код товара (000000001). Поэтому такой URL будет сопоставлен с шаблоном URL ПоказатьЦенуТовара (строка шаблона – /product/{Код}) и будет вызван обработчик этого шаблона ПоказатьЦенуТовараGET(). В результате в окне браузера мы увидим содержимое ответа сервиса (листинг 1.72).

Листинг 1.72. Содержимое ответа сервиса

```
{
"На: ": "21.07.2019 13:06:01",
"Товар": "Чайник",
"Цена": 3300
}
```

Проиндексировать цены на товары

В этом примере, чтобы проиндексировать последние цены товаров на заданную величину, нам понадобится воспользоваться другим шаблоном URL нашего HTTP-сервиса. В обработчике, связанном с этим шаблоном, нам нужно получить срез последних записей из регистра сведений ЦеныТоваров и затем увеличить последние цены товаров на значение, полученное из URL запроса при обращении к сервису. После этого нужно записать в регистр сведений новые цены товаров на текущую дату и затем показать эти новые цены в теле ответа, возвращаемого сервисом.

Итак, добавим у нашего HTTP-сервиса объект Шаблон URL ПроиндексироватьЦены с шаблоном /percent/{Процент}. Затем у этого объекта добавим

подчиненный метод POST, в качестве HTTP-метода оставим предложенное по умолчанию значение POST. Далее создадим обработчик этого метода.

В модуле нашего HTTP-сервиса будет создан шаблон функции-обработчика `ПроиндексироватьЦеныPOST()`. Заполним эту функцию следующим образом (листинг 1.73).

Листинг 1.73. Функция «ПроиндексироватьЦеныPOST»

```
Функция ПроиндексироватьЦеныPOST(Запрос)

    Попытка

        // Получить из запроса параметр URL Процент.
        Процент = Запрос.ПараметрыURL.Получить("Процент");
        Если Процент = Неопределено Тогда
            Ответ = Новый HTTPСервисОтвет(400);
            Ответ.УстановитьТелоИзСтроки("Не задан параметр Процент");
            Ответ.Заголовки.Вставить("Content-type", "application/json");
            Возврат Ответ;
        КонецЕсли;

        Коэффициент = 1 + Число(Процент)/100;
        // Сформировать запрос к регистру сведений.
        Запрос = Новый Запрос;
        Запрос.Текст = "ВЫБРАТЬ
                       |    ЦеныТоваровСрезПоследних.Период КАК Период,
                       |    ЦеныТоваровСрезПоследних.Товар КАК Товар,
                       |    ЕСТЬNULL(ЦеныТоваровСрезПоследних.Цена, 0) КАК Цена
                       |ИЗ
                       |    РегистрСведений.ЦеныТоваров.СрезПоследних( , ) КАК ЦеныТоваровСрезПоследних";

        Выборка = Запрос.Выполнить().Выбрать();

        // В цикле обхода выборки записать в JSON и добавить
        // в регистр сведений последние цены товаров.
        // Создать объект записи и записать строковое значение в строку JSON.
        Запись = Новый ЗаписьJSON;
        Запись.УстановитьСтроку();

        // Записать начало корневого объекта.
        Запись.ЗаписатьНачалоОбъекта();
        Пока Выборка.Следующий() Цикл
            НоваяЦена = Выборка.Цена * Коэффициент;

            Запись.ЗаписатьИмяСвойства(Строка(Выборка.Товар));
            Запись.ЗаписатьНачалоОбъекта();
            Запись.ЗаписатьИмяСвойства("Период");
            Запись.ЗаписатьЗначение(Строка(ТекущаяДата()));
            Запись.ЗаписатьИмяСвойства("Цена");
            Запись.ЗаписатьЗначение(НоваяЦена);
            Запись.ЗаписатьКонецОбъекта();

            // Подготовить менеджер записи.
```

```
                ЗаписьРегистра = РегистрыСведений.ЦеныТоваров.СоздатьМенеджерЗаписи();

                // Установить ключевые поля менеджера записи.
                ЗаписьРегистра.Период = ТекущаяДата();
                ЗаписьРегистра.Товар = Выборка.Товар;
                ЗаписьРегистра.Цена = НоваяЦена;

                // Записать новую запись в базу данных.
                ЗаписьРегистра.Записать();
            КонецЦикла;
            // Записать конец корневого объекта.
            Запись.ЗаписатьКонецОбъекта();

            // Записать результат записи в строку JSON.
            Результат = Запись.Закрыть();

            // Сформировать ответ, возвращаемый HTTP-сервисом.
            Ответ = Новый HTTPСервисОтвет(200);

            // Установить тело ответа из строки Результат.
            Ответ.УстановитьТелоИзСтроки(Результат);

        Исключение

            // Вывести стуктурированную информацию об исключении.
            Ответ = Новый HTTPСервисОтвет(500);
            Информация = ИнформацияОбОшибке();
            Сообщение = Информация.Описание;
            Если Информация.Причина <> Неопределено Тогда
                Сообщение = Сообщение + ":" + Информация.Причина.Описание;
            КонецЕсли;
            Ответ.УстановитьТелоИзСтроки(Сообщение);

        КонецПопытки;

        Ответ.Заголовки.Вставить("Content-type", "application/json");
        Возврат Ответ;

КонецФункции
```

Прокомментируем код функции.

Родительский объект функции-обработчика – шаблон URL Проиндексировать-Цены, описанный с помощью шаблона `/percent/{Процент}`. Этот шаблон содержит непараметризованный сегмент `percent` и параметризованный сегмент `{Процент}`. Это означает, что любые символы, которые могут присутствовать в URL запроса к сервису после символов «/percent/», будут трактоваться как параметр URL `Процент`, который можно получить из переданного в функцию параметра `Запрос`.

Поэтому сначала в обработчике мы делаем попытку получить значение параметра `Процент` (`Запрос.ПараметрыURL.Получить("Процент")`) и анализируем его. Если этот параметр URL не определен, то мы прекращаем работу функции и возвращаем ответ сервиса с кодом состояния 400, сообщающим об ошибке запроса.

Затем мы преобразуем строковое значение параметра `Процент` в числовой коэффициент, на который нам нужно увеличить последние цены товаров.

После этого выполняем запрос, обращающийся к срезу последних регистра ЦеныТоваров. В цикле обхода выборки из результата запроса мы умножаем последние цены товаров на полученный нами коэффициент, заполняем строку JSON списком новых цен и записываем эту строку в переменную `Результат`.

Затем с помощью менеджера записи записываем новые цены товаров на текущую дату в регистр сведений.

После этого создаем объект `HTTPСервисОтвет` с кодом состояния 200, с помощью метода `УстановитьТелоИзСтроки()` устанавливаем тело ответа HTTP-сервиса из строки `Результат` и возвращаем ответ сервиса в переменной `Ответ`.

Если же при попытке получить значение параметра URL из запроса возникает ошибка, то генерируется исключение. С помощью метода глобального контекста `ИнформацияОбОшибке()` мы получаем структурированную информацию об исключении и возвращаем ответ сервиса с кодом состояния 500, сообщающим о причине ошибки.

Чтобы посмотреть результат работы HTTP-сервиса по индексации цен товаров в прикладном решении, надо обратиться к нему с помощью собственного HTTP-запроса и выполнить на сервере метод `POST`. Этот пример будет рассмотрен в следующем разделе «HTTP-запросы – Добавление данных».

HTTP-сервисы в расширениях

В расширениях также можно создавать собственные HTTP-сервисы. Таким образом, в процессе внедрения, не изменяя конфигурацию, находящуюся на поддержке, можно настроить взаимодействие прикладного решения с внешними системами. Например, для синхронизации данных между прикладным решением и веб-сайтом.

Для этого в расширении в ветку Общие > HTTP-сервисы нужно добавить HTTP-сервис, разработать его по аналогии с описанными выше примерами и опубликовать на веб-сервере, установив в диалоге публикации флажок Публиковать HTTP сервисы расширений по умолчанию.

В отличие от HTTP-сервисов, созданных в основной конфигурации, установка в диалоге публикации флажка Публиковать HTTP сервисы расширений по умолчанию не позволяет выбирать для публикации конкретные сервисы расширений. Поэтому если требуется опубликовать лишь некоторые из них, то можно вручную отредактировать файл публикации (default.vrd) и удалить там «ненужные» сервисы.

HTTP-запросы

В этом разделе рассказывается о том, как из системы «1С:Предприятие» можно обратиться к любому произвольному HTTP-сервису, опубликованному в Интернете, а также как обратиться к REST-интерфейсу любой внешней системы, доступному по протоколу OData.

Для простоты мы покажем примеры обращения к HTTP-сервисам и REST-интерфейсу (OData) прикладного решения «1С:Предприятия», реализованные нами в той же демонстрационной конфигурации, что и сами сервисы.

Ниже мы рассмотрим примеры получения, удаления и добавления данных прикладного решения «1С:Предприятия» с помощью HTTP-сервисов, разработанных нами в предыдущем разделе «Примеры реализации HTTP-сервисов». А затем изменение данных прикладного решения через REST-интерфейс (OData), который будет рассматриваться в следующем разделе «Автоматический REST-интерфейс (OData)».

ПОДРОБНЕЕ

> Подробнее познакомиться с реализацией и использованием HTTP-запросов можно в демонстрационной конфигурации «Примеры работы», которая прилагается к книге.

Для обращения к HTTP-сервисам и REST-интерфейсу (OData) используются такие объекты встроенного языка, как `HTTPСоединение`, `HTTPЗапрос` и `HTTPОтвет`.

- Объект `HTTPСоединение` предназначен для взаимодействия с внешними системами по протоколу HTTP. В том числе с помощью объекта `HTTPСоединение` можно:
  - создать HTTP-соединение;
  - записать, получить и удалить файл;
  - отправить ресурс на указанный адрес для обработки;

- определить параметры установленного соединения;
- вызвать произвольный HTTP-метод;
- передать на сервер заголовки запроса и получить заголовки с сервера;
- в качестве тела запроса отправить не только файл, но и строку (с автоматическим перекодированием в нужную кодировку), и двоичные данные.

- Объект `HTTPЗапрос` предназначен для описания HTTP-запросов, отправляемых через объект `HTTPСоединение`. Позволяет задать адрес запрашиваемого ресурса, заголовки и тело запроса.
- Объект `HTTPОтвет` – предоставляет доступ к содержимому ответа HTTP-сервера на запрос.

Для того чтобы изменить (добавить, удалить, прочитать и т. д.) данные информационной базы через REST-интерфейс, нужно создать HTTP-соединение с сервером (например, «localhost»), на котором опубликованы HTTP-сервисы (как созданные разработчиком, так и составляющие автоматический REST-интерфейс) для прикладного решения. Затем создать HTTP-запрос на основе адреса ресурса данных, которые требуется прочитать или изменить с помощью запроса. После этого нужно записать изменяемые данные в строку JSON/XML (или в файл, откуда будет считано тело запроса при отправке) и установить тело HTTP-запроса из этой строки (или установить имя файла тела запроса). И затем с помощью HTTP-соединения отправить запрос к серверу с соответствующим методом (`POST`, `PUT/PATCH`, `DELETE` и др.).

Чтобы выполнить HTTP-запрос нужного вида, у объекта `HTTPСоединение` существуют методы:

- `Получить()` – отправляет `GET`-запрос на сервер;
- `ОтправитьДляОбработки()` – отправляет `POST`-запрос на сервер;
- `Изменить()` – отправляет `PATCH`-запрос на сервер;
- `Записать()` – отправляет `PUT`-запрос на сервер;
- `Удалить()` – отправляет `DELETE`-запрос на сервер;
- с помощью метода `ВызватьHTTPМетод()` можно отправить на сервер произвольный запрос, выполняющий HTTP-метод, имя которого указано в первом параметре метода `ВызватьHTTPМетод()`.

Ниже для ознакомления приведен пример POST-запроса в формате atom/XML, с помощью которого создается новый элемент справочника Товары (листинг 1.74).

Листинг 1.74. POST-запрос в формате atom

```
POST http://host/base/odata/standard.odata/Catalog_Товары HTTP/1.1
User Agent: Fiddler
Host: host
Content Length: 981
<entry>
   <category term="StandardODATA.Catalog_Товары" scheme=
                                        "http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
   <title type="text"/>
   <updated>2014 02 14T12:05:55</updated>
   <author/>
   <summary/>
   <content type="application/xml">
      <m:properties xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m=
                                        "http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
         <d:DeletionMark>false</d:DeletionMark>
         <d:Parent_Key>bbb079ae 8c51 11db a9b0 00055d49b45e</d:Parent_Key>
         <d:IsFolder>false</d:IsFolder>
         <d:Code>000000800</d:Code>
         <d:Description>Шлепанцы</d:Description>
         <d:Артикул>SL56X</d:Артикул>
         <d:Поставщик_Key>086715b0 f348 11db a9c5 00055d49b45e</d:Поставщик_Key>
         <d:Вид>Товар</d:Вид>
         <d:Штрихкод/>
         <d:Описание>&lt;html&gt;Шлепанцы пляжные&lt;/html&gt;</d:Описание>
      </m:properties>
   </content>
</entry>
```

Тот же запрос в формате json выглядит гораздо компактнее (листинг 1.75).

Листинг 1.75. POST-запрос в формате json

```
POST http://host/base/odata/standard.odata/Catalog_Товары HTTP/1.1
Accept: application/json
Accept Charset: UTF 8
User Agent: Fiddler
Content Type: application/json
Content Length: 2426
{
"DeletionMark":false,
"Parent_Key":"bbb079ae 8c51 11db a9b0 00055d49b45e",
"IsFolder":false
"Code":"000000800",
"Description":"Шлепанцы",
"Артикул":"SL56X",
"Поставщик_Key":"086715b0 f348 11db a9c5 00055d49b45e",
"Вид":"Товар",
"Штрихкод":null,
"Описание:"Шлепанцы пляжные"
}
```

Обращение к HTTP-сервисам

Получение данных

Итак, сначала рассмотрим пример получения данных при помощи HTTP-запросов к HTTP-сервисам, разработанных нами в предыдущем разделе. Для большей универсальности URL, по которому мы обращаемся к HTTP-сервису, будем указывать непосредственно в поле ввода демонстрационной обработки.

Обработчик команды, получающей данные, заполним следующим образом (листинг 1.76).

Листинг 1.76. Обработчик команды «ПолучитьДанные»

```
&НаКлиенте
Процедура ПолучитьДанные(Команда)

	// Сформировать строку URL.
	АдресРесурса = СокрЛП(Адрес);

	// Отправить запрос на сервер.
	ВыполнитьЗапрос("GET", АдресРесурса, "");

КонецПроцедуры
```

В этом обработчике в переменной `АдресРесурса` мы запоминаем значение реквизита обработки Адрес, в котором хранится URL для обращения к HTTP-сервису.

Затем вызываем процедуру `ВыполнитьЗапрос()` и передаем в нее в качестве параметров адрес ресурса и имя метода («GET»), который надо выполнить на сервере (листинг 1.77).

Листинг 1.77. Процедура «ВыполнитьЗапрос»

```
&НаКлиенте
Процедура ВыполнитьЗапрос(ИмяМетода, АдресРесурса, СтрокаТелаЗапроса)

	Сообщение = Новый СообщениеПользователю;
	// Установить имя сервера.
	СерверИсточник = "localhost";

	Попытка
		// Создать HTTP-соединение с сервером localhost.
		HTTPСоединение = Новый HTTPСоединение(СерверИсточник);
	Исключение
		// Вывести сообщение об ошибке соединения с сервером.
		Сообщение.Текст = "Не удалось соединиться с сервером: " + СерверИсточник;
		Сообщение.Сообщить();
		Сообщение.Текст = ОписаниеОшибки();
```

```
            Сообщение.Сообщить();
            Возврат;
        КонецПопытки;

        // Создать HTTP-запрос на основе URL.
        HTTPЗапрос = Новый HTTPЗапрос(АдресРесурса);

        // Установить тело запроса из строки JSON.
        Если ИмяМетода <> "DELETE" И ИмяМетода <> "GET" Тогда
            HTTPЗапрос.УстановитьТелоИзСтроки(СтрокаТелаЗапроса);
        КонецЕсли;

        Попытка
            // Получить ответ сервера в виде объекта HTTPОтвет.
            Результат = HTTPСоединение.ВызватьHTTPМетод(ИмяМетода, HTTPЗапрос);
            // Получить содержимое ответа сервера в виде строки.
            Сообщение.Текст = Результат.ПолучитьТелоКакСтроку();
            Сообщение.Сообщить();
        Исключение
            // Вывести сообщение об ошибке при получении ответа сервера.
            Сообщение.Текст = ОписаниеОшибки();
            Сообщение.Сообщить();
            Возврат;
        КонецПопытки;

КонецПроцедуры
```

В этой процедуре в качестве имени сервера мы указываем «localhost», создаем на его основе объект `HTTPСоединение` и соединяемся с этим сервером.

Далее на основе URL, содержащегося в параметре `АдресРесурса`, создаем `HTTPЗапрос`. Затем, в случае если это не DELETE- и не GET-запрос, устанавливаем тело запроса из строки JSON, содержащейся в параметре `СтрокаТелаЗапроса`.

После этого при помощи метода `ВызватьHTTPМетод()` объекта `HTTPСоединение` отправляем на сервер запрос (в данном случае с методом GET) к информационной базе. Ответ сервера в виде объекта `HTTPОтвет` будет возвращен в переменную `Результат`. С помощью метода `ПолучитьТелоКакСтроку()` объекта `HTTPОтвет` мы получаем тело ответа в виде строки и выводим его в сообщение.

Проверим, как это работает.

Например, мы хотим получить список всех сотрудников из справочника Сотрудники. Это можно сделать с помощью HTTP-сервиса Сотрудники с корневым URL `employees`.

В поле ввода демонстрационной обработки укажем тот же URL, который мы вводили для этого в браузере, – `/REST/hs/employees` (только без имени сервера) и нажмем кнопку Получить данные. В результате в окне сообщений мы увидим ответ сервера, содержащий список всех сотрудников (рис. 1.16).

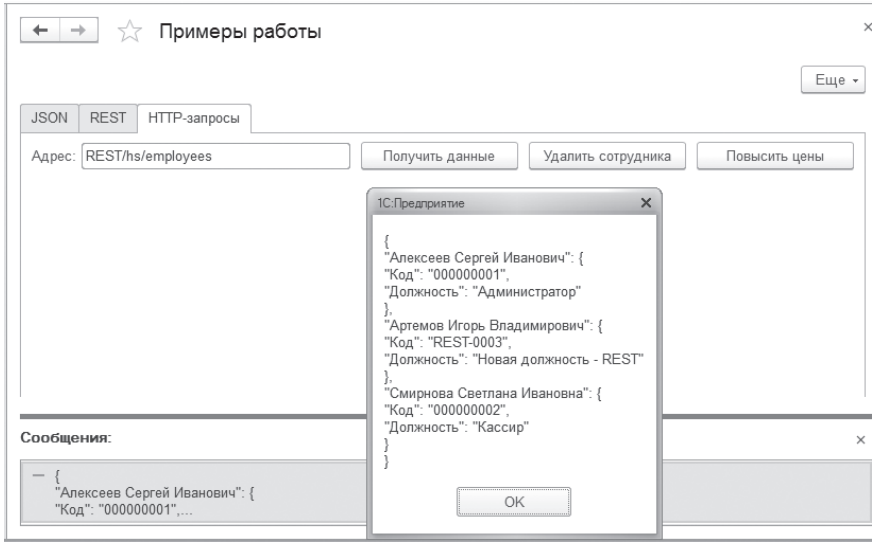


Рис. 1.16. Ответ сервера

Теперь получим список последних цен товаров из регистра сведений ЦеныТоваров, актуальных на конкретную дату. Это можно сделать с помощью HTTP-сервиса ЦеныТоваров с корневым URL `prices`.

В поле ввода обработки укажем тот же URL, который мы вводили для этого в браузере, – `/REST/hs/prices/date/20190701` (только без имени сервера) и нажмем кнопку Получить данные. В результате в окне сообщений мы увидим ответ сервера, содержащий список последних цен товаров (рис. 1.17).

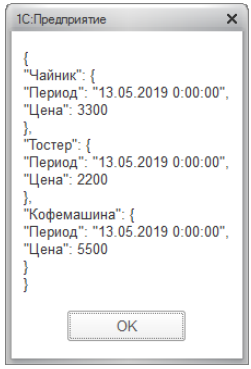


Рис. 1.17. Ответ сервера

Удаление данных

С помощью HTTP-сервиса Сотрудники, разработанного нами в предыдущем разделе «Удалить данные о конкретном сотруднике», а точнее шаблона сервиса Сотрудник и его HTTP-метода DELETE можно найти сотрудника в справочнике Сотрудники и поставить ему пометку удаления.

Для проверки введем в справочник нового сотрудника с кодом 000000004. Затем в поле ввода обработки укажем URL для доступа к сервису – `REST/hs/employees/code/000000004`.

Если мы, как и раньше, нажмем кнопку Получить данные, то сработает HTTP-метод GET шаблона Сотрудник HTTP-сервиса Сотрудники. В результате на сервер будет отправлен GET-запрос и мы увидим в ответе сервера данные о только что добавленном сотруднике.

Чтобы отправить на сервер DELETE-запрос, добавим команду УдалитьСотрудника. Обработчик команды заполним следующим образом (листинг 1.78).

Листинг 1.78. Обработчик команды «УдалитьСотрудника»

```
&НаКлиенте
Процедура УдалитьСотрудника(Команда)

	// Сформировать строку URL.
	АдресРесурса = СокрЛП(Адрес);

	// Отправить запрос на сервер.
	ВыполнитьЗапрос("DELETE", АдресРесурса, "");

КонецПроцедуры
```

В этом обработчике мы вызываем процедуру `ВыполнитьЗапрос()` и передаем в нее в качестве параметров адрес ресурса и имя метода («DELETE»), который надо выполнить на сервере. Поскольку эта процедура уже была рассмотрена в листинге 1.77, не будем еще раз на этом останавливаться.

Запустим нашу демонстрационную обработку, укажем URL для доступа к сервису – `REST/hs/employees/code/000000004` и нажмем кнопку Удалить сотрудника. В результате сработает HTTP-метод DELETE шаблона Сотрудник HTTP-сервиса Сотрудники. Будет вызван обработчик этого HTTP-метода, в котором сотрудник с кодом 000000004 будет найден в справочнике Сотрудники и помечен на удаление.

Добавление данных

С помощью HTTP-сервиса ЦеныТоваров, разработанного нами в предыдущем разделе «Проиндексировать цены на товары», а точнее – шаблона сервиса ПроиндексироватьЦены и его HTTP-метода POST, можно получить последние цены товаров из регистра сведений ЦеныТоваров, проиндексировать их на заданную величину и записать в регистр сведений новые цены товаров на текущую дату, а затем показать эти новые цены в теле ответа, возвращаемого сервисом.

Чтобы отправить на сервер POST-запрос, добавим команду ПроиндексироватьЦены. Обработчик команды заполним следующим образом (листинг 1.79).

Листинг 1.79. Обработчик команды «ПроиндексироватьЦены»

```
&НаКлиенте
Процедура ПроиндексироватьЦены(Команда)

    // Сформировать строку URL.
    АдресРесурса = СокрЛП(Адрес);

    // Отправить запрос на сервер.
    ВыполнитьЗапрос("POST", АдресРесурса, "");

КонецПроцедуры
```

В этом обработчике мы вызываем процедуру `ВыполнитьЗапрос()` и передаем в нее в качестве параметров адрес ресурса и имя метода («POST»), который надо выполнить на сервере. Поскольку эта процедура уже была рассмотрена в листинге 1.77, не будем еще раз на этом останавливаться.

Запустим нашу демонстрационную обработку, укажем URL для доступа к сервису – `/REST/hs/prices/percent/10` и нажмем кнопку Повысить цены.

В результате сработает HTTP-метод POST шаблона ПроиндексироватьЦены HTTP-сервиса ЦеныТоваров. Будет вызван обработчик этого HTTP-метода,

в котором выполняется индексация цен товаров, затем новые цены товаров будут повышены на 10 процентов и помещены в ответ сервера, а также в регистр сведений ЦеныТоваров (рис. 1.18).

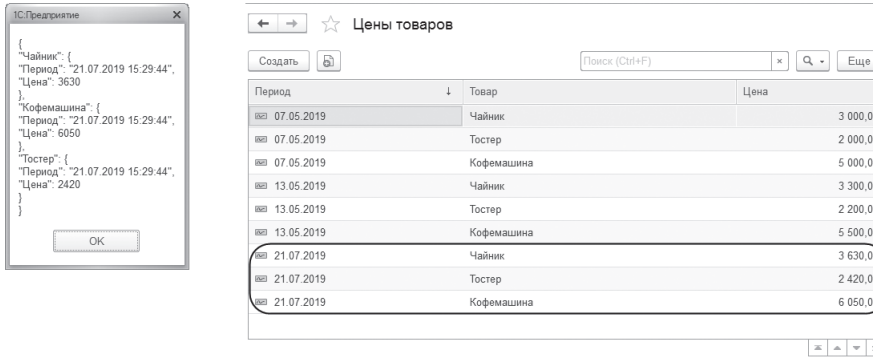


Рис. 1.18. Ответ сервера и данные регистра сведений «Цены товаров»

Обращение к REST-интерфейсу (OData)

Пример по изменению данных табличной части при помощи HTTP-запроса, обращающегося к REST-интерфейсу (OData) прикладного решения, рассмотрен ниже в разделе «Примеры использования – Модификация данных прикладного решения». Также там описаны другие подобные примеры, позволяющие модифицировать данные, доступные через REST-интерфейс.

Автоматический REST-интерфейс (OData)

Платформа «1С:Предприятие» может автоматически создать REST-интерфейс для любого прикладного решения. Благодаря своей универсальности и кроссплатформенности REST-интерфейс является очень удобным инструментом для интеграции прикладного решения со сторонними системами.

Общая информация

Для того чтобы сторонние системы могли обращаться к прикладному решению через REST-интерфейс, этот интерфейс достаточно опубликовать на веб-сервере и указать, какие прикладные объекты будут доступны в нем. После этого внешние приложения с помощью HTTP-запросов через REST-интерфейс могут читать данные «1С:Предприятия», изменять их, создавать новые объекты данных и удалять существующие.

Автоматический REST-интерфейс может использоваться для следующих задач:

- интеграция прикладного решения с интернет-сайтами и интернет-магазинами;
- реализация сторонними средствами дополнительной функциональности прикладного решения без изменения его конфигурации;
- загрузка данных в прикладное решение и выгрузка данных из него;
- интеграция прикладного решения с корпоративными системами, возможно, даже без дополнительного программирования.

Доступ к данным приложения через REST-интерфейс позволяет выполнять следующие типичные операции:

- получение списка документов, справочников, записей регистра сведений и т. п., возможно, с фильтром;
- получение данных элемента справочника, документа (по ссылке), данных записи независимого регистра сведений (по ключу), данных набора записей подчиненного регистра (по регистратору);
- редактирование данных одного элемента справочника, документа и другого ссылочного объекта;
- создание нового элемента справочника, документа, набора записей;
- проведение одного документа, старт бизнес-процесса.

В качестве протокола доступа платформа использует протокол OData версии 3.0. Это открытый веб-протокол для запроса и обновления данных. Он позволяет оперировать данными, используя в качестве запросов HTTP-команды. Получать ответы можно в формате Atom/XML или JSON.

Клиенты OData существуют практически для всех значимых платформ:

- мобильные: iOS, Windows Phone, Android;
- серверные/настольные: .NET, Java, PHP, Objective-C, Ruby, JavaScript;
- поддержка в системах управления содержимым (CMS): Drupal, Joomla.

В этой книге мы не будем рассматривать примеры работы в этих платформах и то, какими средствами в них выполняется обращение по протоколу OData. При необходимости вы можете обратиться к документации той платформы, которая вам нужна.

Поскольку эта книга – о системе «1С:Предприятие», то работу с автоматическим REST-интерфейсом мы рассмотрим для простоты на примере обращения к нему (клиентская часть) из этой же самой информационной базы. В реальной жизни такая задача вряд ли встретится, но взаимодействие одной базы «1С:Предприятия» с другой базой «1С:Предприятия» через REST-интерфейс – это довольно частая задача. И решается она точно такими же средствами.

Для обращения к REST-интерфейсу используются такие объекты встроенного языка, как `HTTPСоединение`, `HTTPЗапрос` и `HTTPОтвет` (дальше вы увидите это на примерах в разделе «Примеры использования»).

В REST-интерфейсе доступны практически все основные объекты конфигурации: справочники, документы, константы, перечисления, планы обмена, регистры накопления, расчета, бухгалтерии и регистры сведений, виртуальные таблицы периодического регистра сведений, регистров бухгалтерии и регистров накопления, планы счетов, видов характеристик и видов расчета, бизнес-процессы, задачи и журналы документов.

В REST-интерфейсе доступны также реквизиты объектов конфигурации, табличные части и реквизиты табличных частей, доступны операции создания, чтения, модификации и удаления данных, а также некоторые методы встроенного языка. Например:

- для документа – проведение и отмена проведения;
- для задачи – выполнение;
- для бизнес-процесса – старт;
- для регистра сведений – получение среза первых и среза последних;
- для регистра накопления и регистра бухгалтерии – получение остатков, оборотов, остатков и оборотов;
- для регистра расчета – получение данных графика, фактического периода действия, перерасчета и базы.

При чтении и записи данных с помощью REST-интерфейса платформа выполняет все обычные проверки прав и вызывает обработчики событий.

ПОДРОБНЕЕ

Более подробно про автоматический REST-интерфейс рассказано в документации «1С:Предприятия» в разделе «Глава 17. Механизмы интернет-сервисов – HTTP-сервисы – Стандартный интерфейс ODATA».

Правила формирования URL запроса

Обращение к стандартному интерфейсу OData выполняется с помощью HTTP-запроса по определенному URL. URL формируется специальным образом и состоит из следующих частей:

1. Адрес информационной базы.
2. Признак обращения к стандартному интерфейсу OData.
3. Имя ресурса, к которому выполняется обращение.
4. Параметры запроса обращения к ресурсу.

Адрес информационной базы – это обычный URL, по которому выполняется доступ, например, к информационной базе с помощью веб-клиента. Например, `http://localhost/base` или `http://host.server.zone/data-base`. Также необходимо помнить, что при использовании информационной базы, в которой настроено разделение данных, значения разделителей можно указывать только в URL информационной базы. Указание разделителей с помощью параметра Z не поддерживается.

Признак обращения к стандартному интерфейсу Odata – это последовательность символов в URL: `/odata/standard.odata`.

Имя ресурса, к которому выполняется обращение – это особым образом сформированный идентификатор ресурса (возможно, с параметром) или предопределенные ресурсы. Например, `$metadata` или `Catalog_Контрагент(guid'value')`.

Параметры обращения к ресурсу. В качестве параметров обращения выступают параметры в виде, принятом для HTTP-запросов: `?ключ=значение&ключ2=значение2`.

При обращении к ресурсу могут использоваться специальные ключевые слова, имеющие специальное назначение:

- `$format` – указывает, в каком формате необходимо получить данные. Если ключевое слово не указано, данные получаются в формате atom-xml:
  - `$format=atom` – возвращает данные в формате atom-xml;
  - `$format=json` – возвращает данные в формате json. При обращении к REST-интерфейсу из других внешних платформ для указания того, что данные должны возвращаться в формате json, можно указать `MIME`-тип application/json в заголовке `Accept` HTTP-запроса на получение данных.
- `$metadata` – указывает, что требуется получить описание стандартного интерфейса OData.
- `$filter` – описывает отбор, применяемый при получении данных (см. раздел «Правила формирования условий отбора»).
- `$select` – описывает перечень свойств *сущности* (объектных типов, наборов записей регистров и строк табличных частей объектных типов), которые нужно получить при обращении к стандартному интерфейсу OData.

После того как сформирован URL необходимого ресурса, следует выполнить HTTP-запрос нужного вида. В зависимости от того, какая операция выполняется, используется соответствующий HTTP-метод:

- Получение данных – метод `GET`. В платформе «1С:Предприятие» этому методу соответствует метод `Получить()` объекта `HTTP-соединение`.

- Создание объекта – метод `POST`. В платформе «1С:Предприятие» этому методу соответствует метод `ОтправитьДляОбработки()` объекта `HTTP-соединение`.
- Обновление данных:
  - метод `PATCH` – в этом случае можно указывать только те свойства сущности, которые необходимо обновить. В платформе «1С:Предприятие» этому методу соответствует метод `Изменить()` объекта `HTTP-соединение`;
  - метод `PUT` – в этом случае необходимо указывать все свойства сущности. В платформе «1С:Предприятие» этому методу соответствует метод `Записать()` объекта HTTP-соединение.
- Удаление данных – метод `DELETE`. В платформе «1С:Предприятие» этому методу соответствует метод `Удалить()` объекта `HTTP-соединение`.

В результате выполнения запроса клиентское приложение получает ответ сервера, который кроме кода состояния может содержать различные данные, предоставленные сервером, в виде XML- или JSON-документа.

Правила формирования имени ресурса

При обращении к какому-либо ресурсу его идентификатор формируется по следующему принципу: `ПрефиксИмени _ ИмяОбъектаКонфигурации _ СуффиксИмени`. С помощью стандартного интерфейса OData можно получить доступ к следующим объектам (табл. 1.1).

Таблица 1.1. Имена ресурсов для объектов конфигурации

Объект конфигурации	Префикс имени для указания в URL
Справочник	Catalog
Документ	Document
Журнал документов	DocumentJournal
Константа	Constant
План обмена	ExchangePlan
План счетов	ChartOfAccounts
План видов расчета	ChartOfCalculationTypes
План видов характеристик	ChartOfCharacteristicTypes
Регистр сведений	InformationRegister
Регистр накопления	AccumulationRegister
Регистр расчета	CalculationRegister
Регистр бухгалтерии	AccountingRegister
Бизнес-процесс	BusinessProcess
Задача	Task

ИмяОбъектаКонфигурации – имя объекта конфигурации, как оно задано при разработке прикладного решения в конфигураторе.

СуффиксИмени – предназначен для уточнения имени ресурса и является необязательной частью имени. В качестве суффикса имени могут выступать следующие выражения:

- имя табличной части объекта;
- имя реквизита табличной части или набора записей;
- имя виртуальной таблицы регистра;
- RowType;
- RecordType.

Если объект обладает табличной частью, то для получения доступа ко всем записям этой табличной части необходимо добавить имя табличной части после имени самого объекта. Например, для получения всех строк табличной части Товары всех документов РасходТовара будет необходимо выполнить GET-запрос по следующему адресу: http://localhost/base/odata/standard.odata/Document_РасходТовара_Товары. Соответствующий пример будет рассмотрен в разделе «Получение данных из табличной части документа».

Если объект обладает табличной частью, то имеется возможность указать, что требуется получение не всех реквизитов табличной части, а некоторого списка этих реквизитов. Для этого необходимо указать в параметре $select список требуемых реквизитов в следующем виде: <Имя табличной части>/<Имя поля>. Аналогичная возможность предоставляется для наборов записей регистров, где в качестве имени табличной части выступает RecordSet: RecordSet/<Имя поля>.

В роли виртуальной таблицы регистра выступает функция, связанная с ресурсом, возвращающая набор сущностей регистра. Имя функции совпадает с английским вариантом имени используемой виртуальной таблицы языка запросов. Параметры функции соответствуют параметрам виртуальной таблицы. Так, для получения среза последних регистра сведений КурсыВалют следует выполнить GET-запрос по следующему адресу: http://localhost/demo/odata/standard.odata/InformationRegister_КурсыВалют./SliceLast(). Соответствующий пример будет рассмотрен в разделе «Получение данных регистра сведений».

ПОДРОБНЕЕ

Правила формирования имен ресурсов для получения данных через REST-интерфейс более подробно описаны в документации «1С:Предприятия» в разделе «Глава 17. Механизмы интернет-сервисов – HTTP-сервисы – Стандартный интерфейс ODATA – Правила формирования имени ресурса».

Правила формирования условий отбора

При получении данных можно отфильтровать их, если это требуется. Для этого предназначен специальный язык, позволяющий описывать условия, которым должны соответствовать данные, которые возвращает стандартный интерфейс OData. Описание отбора начинается с ключевого слова `$filter`, после которого следует собственно условие. Поддерживаются следующие операции (табл. 1.2, 1.3).

Таблица 1.2. Логические операции, используемые для фильтрации данных

Описание	Имя	Пример
Равно	eq	/Catalog_Города?$filter=Description eq ‘Главный’
Не равно	ne	/Catalog_Города?$filter=Description ne ‘Пермь’
Больше	gt	/Catalog_Товары?$filter=Цена gt 10
Больше или равно	ge	/Catalog_Товары?$filter=Цена ge 10
Меньше	lt	/Catalog_Товары?$filter=Цена lt 10
Меньше или равно	le	/Catalog_Товары?$filter=Цена le 10
Логическое «Или»	or	/Catalog_Товары?$filter=Цена lt 10 or Цена gt 100
Логическое «И»	and	/Catalog_Товары?$filter=Цена gt 10 and Цена lt 100
Отрицание	not	/Catalog_Товары?$filter=not (Цена eq 10)

Таблица 1.3. Арифметические операции, используемые для фильтрации данных

Описание	Имя	Пример
Сложение	add	/Catalog_Товары?$filter=Цена add 5 gt 10
Вычитание	sub	/Catalog_Товары?$filter=Цена sub 5 gt 10
Умножение	mul	/Catalog_Товары?$filter=Цена mul 5 gt 1000
Деление	div	/Catalog_Товары?$filter=Цена div 4 gt 2

Например, используя следующее условие, можно получить товары из справочника Товары с именем Молоко и ценой менее 2500: `http://localhost/odata/standard.odata/Catalog _ Товары?$filter=Имя eq ‘Молоко’ and Цена lt 2500`.

Для группировки условий и указания приоритета операций их можно заключить в скобки. Например: `http://localhost/odata/standard.odata/Catalog _ Товары?$filter=(Цена add 5) gt 10`.

При формировании условий запроса или формировании реквизита, по которому выполняется упорядочивание, могут применяться функции для работы со строками (`substring ()`, `startswith()`, `endswith()` и др.), с датами (`year()`, `dateadd()`, `dayofyear()` и др.) и некоторые другие.

Примеры использования условий отбора получаемых данных будут рассмотрены в разделах «Отобрать записи справочника по условию».

ПОДРОБНЕЕ

> Правила формирования условий отбора для получения данных через REST-интерфейс более подробно описаны в документации «1С:Предприятия» в разделе «Глава 17. Механизмы интернет-сервисов – HTTP-сервисы – Стандартный интерфейс ODATA – Правила формирования условия отбора».

Примеры использования

Продемонстрируем все вышесказанное на небольших примерах.

ПОДРОБНЕЕ

> Подробнее познакомиться с обращением к данным информационной базы через REST-интерфейс можно в демонстрационной конфигурации «Примеры работы», которая прилагается к книге.

Прежде всего, чтобы прикладное решение стало доступно через REST-интерфейс, опубликуем интерфейс OData для этого решения на веб-сервере. Для этого в диалоге публикации информационной базы на веб-сервере должен быть установлен флажок Публиковать стандартный интерфейс OData (рис. 1.19).

В нашем примере публикация выполняется на локальном компьютере (на сервере «localhost») в каталог REST.

Затем, для того чтобы объекты конфигурации стали доступны через стандартный интерфейс OData, необходимо разрешить это с помощью метода глобального контекста `УстановитьСоставСтандартногоИнтерфейсаOData()`.

Рис. 1.19. Публикация конфигурации на веб-сервере

В реальных примерах для этого нужно написать и один раз выполнить обработку на встроенном языке, которая укажет, данные каких объектов конфигурации будут доступны через REST-интерфейс. Но в нашей тренировочной базе мы для простоты с помощью команды сделаем доступными справочники Поставщики, Должности, Сотрудники и Товары, документ ПоступлениеТоваров и регистр сведений ЦеныТоваров (листинг 1.80).

Листинг 1.80. Установка списка объектов, доступных через стандартный интерфейс OData

```
&НаСервереБезКонтекста
Процедура РазрешитьДоступODATA()

	МДанные = Новый Массив();
	МДанные.Добавить("Справочник.Сотрудники");
	МДанные.Добавить("Справочник.Должности");
	МДанные.Добавить("Справочник.Поставщики");
	МДанные.Добавить("Документ.ПоступлениеТоваров");
	МДанные.Добавить("РегистрСведений.ЦеныТоваров");
	МДанные.Добавить("Справочник.Товары");

	УстановитьСоставСтандартногоИнтерфейсаOData(МДанные);

КонецПроцедуры
```

Имена нужных объектов конфигурации мы добавляем в массив и передаем его в метод `УстановитьСоставСтандартногоИнтерфейсаOData()`.

Следует иметь в виду, что REST-интерфейс по умолчанию возвращает данные в формате atom-xml. Это «наследие» стандарта. По современным же представлениям приоритетным форматом является JSON. Поэтому во всех примерах в списке параметров запроса мы укажем, что данные должны быть возвращены именно в этом формате – `$format=json`.

Получение всех данных справочника

В нашей демонстрационной базе существует справочник Поставщики. Это справочник содержит следующие данные (рис. 1.20).

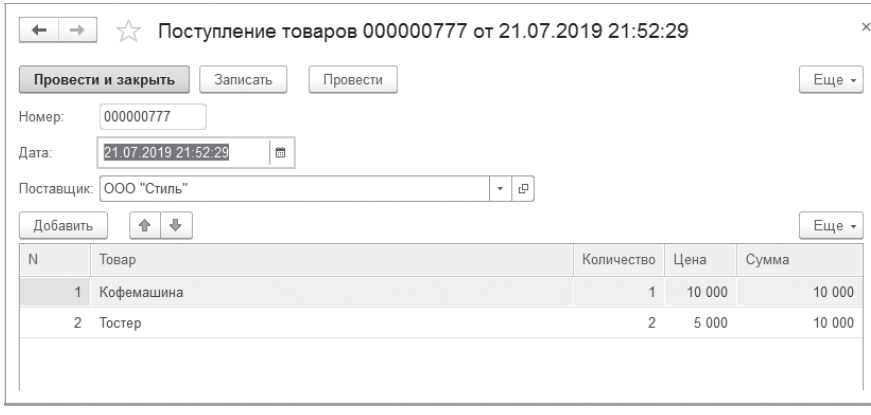


Рис. 1.20. Данные справочника «Поставщики»

Предположим, нам нужно получить все данные из этого справочника.

Процедура, которая выполняет запрос к данным справочника и получает ответ в формате JSON, может иметь следующий вид (листинг 1.81).

Листинг 1.81. Пример получения данных из справочника

```
&НаКлиенте
Процедура Команда1(Команда)

	СерверИсточник = "localhost";
	Сообщение = Новый СообщениеПользователю;

	Попытка
		// Создать HTTP-соединение с сервером localhost.
		HTTPСоединение = Новый HTTPСоединение(СерверИсточник);
	Исключение
```

```
            // Вывести сообщение об ошибке соединения с сервером.
            Сообщение.Текст = "Не удалось соединиться с сервером: " + СерверИсточник;
            Сообщение.Сообщить();
            Сообщение.Текст = ОписаниеОшибки();
            Сообщение.Сообщить();
            Возврат;
    КонецПопытки;

    // Сформировать строку URL.
    АдресРесурса = "/REST/odata/standard.odata/Catalog_Поставщики?$format=json";

    // Создать HTTP-запрос на основе URL.
    HTTPЗапрос = Новый HTTPЗапрос(АдресРесурса);

    Попытка
            // Получить ответ сервера в виде объекта HTTPОтвет.
            Результат = HTTPСоединение.Получить(HTTPЗапрос);
            // Получить содержимое ответа сервера в виде строки.
            Сообщение.Текст = Результат.ПолучитьТелоКакСтроку();
            Сообщение.Сообщить();
    Исключение
            // Вывести сообщение об ошибке при получении ответа сервера.
            Сообщение.Текст = ОписаниеОшибки();
            Сообщение.Сообщить();
            Возврат;
    КонецПопытки;

КонецПроцедуры
```

В качестве имени сервера мы просто указываем «localhost» и создаем на его основе объект `HTTPСоединение`.

Затем в переменной `АдресРесурса` мы формируем URL запроса для обращения к данным нашего справочника (.../`Catalog_Поставщики`?...). Поскольку нам нужно получить все данные справочника, мы просто указываем для этого префикс `Catalog_` и имя справочника.

Остальные составляющие и правила формирования URL были подробно описаны в разделе «Правила формирования URL запроса».

Далее на основе этого URL создаем `HTTPЗапрос` и выполняем GET-запрос к информационной базе при помощи метода `Получить()` объекта `HTTPСоединение`. Ответ сервера в виде объекта `HTTPОтвет` будет возвращен в переменную `Результат`. После этого с помощью метода `ПолучитьТелоКакСтроку()` объекта `HTTPОтвет` мы получаем тело ответа в виде строки и выводим его в сообщение.

Если при получении ответа сервера возникнут ошибки, будет вызвано исключение. Типичные ошибки клиента, возникающие при обращении к серверу, будут рассмотрены в разделе «Типичные ошибки при получении данных».

Ответ сервера в виде строки, содержащий все данные справочника Поставщики, будет выглядеть следующим образом (рис. 1.21).

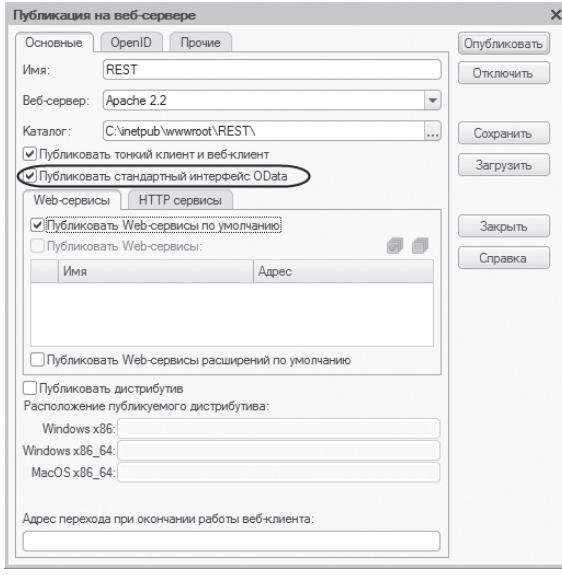


Рис. 1.21. Ответ сервера, содержащий все данные справочника «Поставщики»

Выше, в листинге 1.81, мы показали пример процедуры, с помощью которой устанавливается HTTP-соединение и через REST-интерфейс получаются данные информационной базы. Кроме того, аналогичным образом, как будет показано в следующих примерах, эти данные могут изменяться, удаляться или добавляться.

Но если нужно просто получить данные информационной базы методом `GET`, то можно набрать URL запроса (`localhost/REST/odata/standard.odata/Catalog_Поставщики?$format=json`) непосредственно в адресной строке браузера и сразу же в окне браузера увидеть содержимое ответа сервера (листинг 1.82).

Листинг 1.82. Содержимое ответа сервера

```
{
"odata.metadata": "http://localhost/REST/odata/standard.odata/$metadata#Catalog_Поставщики",
"value": [{
"Ref_Key": "9cfabfb0-4cb2-11e9-9334-642737df2048",
"DataVersion": "AAAAAQAAAAA=",
"DeletionMark": false,
"Code": "000000001",
```

```
"Description": "ОАО \"Фонтан\"",
"Predefined": false,
"PredefinedDataName": ""
},{
"Ref_Key": "9cfabfb1-4cb2-11e9-9334-642737df2048",
"DataVersion": "AAAAAgAAAAA=",
"DeletionMark": false,
"Code": "000000002",
"Description": "ООО \"Топаз\"",
"Predefined": false,
"PredefinedDataName": ""
},{
"Ref_Key": "9cfabfb2-4cb2-11e9-9334-642737df2048",
"DataVersion": "AAAAAwAAAAA=",
"DeletionMark": false,
"Code": "000000003",
"Description": "ООО \"Стиль\"",
"Predefined": false,
"PredefinedDataName": ""
}]
}
```


В дальнейших примерах при получении данных из информационной базы мы будем использовать именно этот метод, то есть смотреть результат запроса в браузере.

Получение количества записей в результате запроса

Предположим, мы хотим включить в результат запроса не только полученные записи, но и количество этих записей. В строке URL запроса это задается с помощью параметра `$inlinecount=allpages`. Наберем в адресной строке браузера следующий URL (листинг 1.83).

Листинг 1.83. URL запроса

```
localhost/REST/odata/standard.odata/Catalog_Поставщики?$format=json&$inlinecount=allpages
```

ПРИМЕЧАНИЕ

Подобные листинги в этой книге записаны в несколько строк только для того, чтобы их было удобнее читать. На самом деле URL – это одна строка.

Обратите внимание, что список параметров запроса начинается со знака вопрос (?), а различные параметры связываются между собой символом амперсанд (&).

В результате в окне браузера мы увидим содержимое ответа сервера. Количество полученных записей справочника – три (листинг 1.84).

Листинг 1.84. Содержимое ответа сервера

```
{
"odata.metadata": "http://localhost/REST/odata/standard.odata/$metadata#Catalog_Поставщики",
"odata.count": "3",
"value": [{
"Ref_Key": "9cfabfb0-4cb2-11e9-9334-642737df2048",
"DataVersion": "AAAAAQAAAAA=",
"DeletionMark": false,
"Code": "000000001",
"Description": "ОАО \"Фонтан\"",
"Predefined": false,
"PredefinedDataName": ""
},{
"Ref_Key": "9cfabfb1-4cb2-11e9-9334-642737df2048",
"DataVersion": "AAAAAgAAAAA=",
"DeletionMark": false,
"Code": "000000002",
"Description": "ООО \"Топаз\"",
"Predefined": false,
"PredefinedDataName": ""
},{
"Ref_Key": "9cfabfb2-4cb2-11e9-9334-642737df2048",
"DataVersion": "AAAAAwAAAAA=",
"DeletionMark": false,
"Code": "000000003",
"Description": "ООО \"Стиль\"",
"Predefined": false,
"PredefinedDataName": ""
}]
}
```

Получение данных из некоторых полей справочника и их сортировка

В нашей демонстрационной базе существует справочник Должности. Это справочник содержит следующие данные (рис. 1.22).

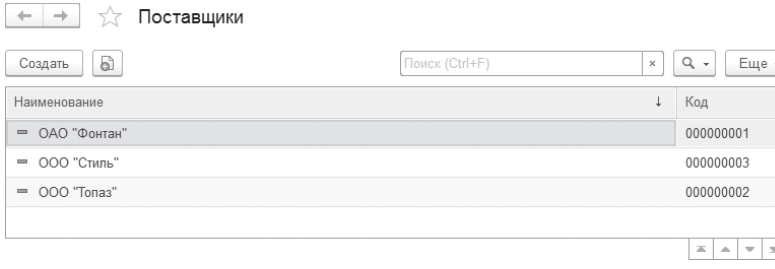


Рис. 1.22. Данные справочника «Должности»

Предположим, нам нужно получить не все данные из этого справочника, а только значения некоторых полей и при этом отсортировать их в определенном порядке.

Например, нам нужно получить только значения полей справочника `Ref _ Key` (ссылка) и `Description`. Перечень нужных полей в строке URL запроса задается через запятую в значении параметра `$select`. Кроме того, получаемые данные нужно отсортировать по возрастанию кодов должностей. Это задается с помощью параметра `$orderby=Code asc`.

Итак, наберем в адресной строке браузера следующий URL (листинг 1.85).

Листинг 1.85. URL запроса

```
localhost/REST/odata/standard.odata/Catalog_Должности?$format
                                        =json&$select=Ref_Key, Description&$orderby=Code asc
```

В результате в окне браузера мы увидим содержимое ответа сервера (листинг 1.86).

Листинг 1.86. Содержимое ответа сервера

```
{
"odata.metadata": "http://localhost/REST/odata/standard.odata/$metadata#Catalog_Должности",
"value": [{
"Ref_Key": "88d54405-36a1-11e9-8bb2-642737df2048",
"Description": "Менеджер"
},{
"Ref_Key": "88d54406-36a1-11e9-8bb2-642737df2048",
"Description": "Администратор"
},{
"Ref_Key": "88d54407-36a1-11e9-8bb2-642737df2048",
"Description": "Кассир"
}]
}
```

Получение данных по ссылке

Теперь получим все данные из справочника Должности для конкретной должности с известным идентификатором. Этот идентификатор мы можем скопировать из ответа сервера, полученного в предыдущем примере (см. листинг 1.86).

Поскольку справочник – это ссылочный объект конфигурации, каждая запись справочника обладает уникальным идентификатором – ссылкой. Эта ссылка содержится в поле `Ref _ Key`. И в предыдущем примере мы как раз получали значение этого поля для всех записей справочника Должности. Кроме того, когда возвращаются все данные справочника, для каждой записи также возвращается значение этого поля. Поэтому, получив предыдущий ответ, мы можем узнать из него `Ref _ Key` нужного элемента и в следующих запросах использовать его, указав значение идентификатора в скобках после имени ресурса – `(guid'...')`.

Итак, наберем в адресной строке браузера следующий URL (листинг 1.87).

Листинг 1.87. URL запроса

```
localhost/REST/odata/standard.odata/Catalog_Должности(
                                    guid'88d54406-36a1-11e9-8bb2-642737df2048')?$format=json&
```

В результате в окне браузера мы увидим содержимое ответа сервера (листинг 1.88).

Листинг 1.88. Содержимое ответа сервера

```
{
"odata.metadata": "http://localhost/REST/odata/standard.odata/$metadata#Catalog_Должности/@Element",
"Ref_Key": "88d54406-36a1-11e9-8bb2-642737df2048",
"Predefined": false,
"PredefinedDataName": "",
"DataVersion": "AAAAAgAAAAA=",
"Description": "Администратор",
"Code": "000000002",
"DeletionMark": false
}
```

Обратите внимание, что поскольку у справочника ключевое поле одно, то можно не указывать его имя, а сразу писать в скобках после имени справочника `guid'...'`. Если же ключевых полей несколько, то имена всех этих полей вместе с их значениями надо перечислять в скобках после имени ресурса: `(Ref _ Key=guid'...', LineNumber=...)`.

Отобрать записи справочника по условию

В строке URL запроса условия отбора получаемых данных задаются в параметре `$filter`. Более подробно об этом рассказано в разделе «Правила формирования условий отбора».

Отобрать записи справочника по началу наименования

Предположим, нам требуется отфильтровать записи справочника Поставщики так, чтобы из списка поставщиков отбирались те, чье наименование начинается на «ООО». Для описания условия отбора воспользуемся функцией `startswith()`.

Наберем в адресной строке браузера следующий URL (листинг 1.89).

Листинг 1.89. URL запроса

```
localhost/REST/odata/standard.odata/Catalog_Поставщики?$format=
                                    json&$filter=startswith(Description, 'ООО') eq true
```

В результате в ответ сервера будут включены только данные двух из трех поставщиков, удовлетворяющих условию отбора (листинг 1.90).

Листинг 1.90. Содержимое ответа сервера

```
{
"odata.metadata": "http://localhost/REST/odata/standard.odata/$metadata#Catalog_Поставщики",
"value": [{
"Ref_Key": "9cfabfb2-4cb2-11e9-9334-642737df2048",
"DataVersion": "AAAAAwAAAAA=",
"DeletionMark": false,
"Code": "000000003",
"Description": "ООО \"Стиль\"",
"Predefined": false,
"PredefinedDataName": ""
},{
"Ref_Key": "9cfabfb1-4cb2-11e9-9334-642737df2048",
"DataVersion": "AAAAAgAAAAA=",
"DeletionMark": false,
"Code": "000000002",
"Description": "ООО \"Топаз\"",
"Predefined": false,
"PredefinedDataName": ""
}]
}
```

Отобрать записи справочника по дате рождения

В нашей демонстрационной базе существует справочник Сотрудники. Это справочник содержит следующие данные (рис. 1.23).

Рис. 1.23. Данные справочника «Сотрудники»

Предположим, нам требуется отфильтровать записи справочника Сотрудники так, чтобы из списка сотрудников отбирались родившиеся в 1980 году. Для описания условия отбора воспользуемся функцией `year()`.

Наберем в адресной строке браузера следующий URL (листинг 1.91).

Листинг 1.91. URL запроса

```
localhost/REST/odata/standard.odata/Catalog_Сотрудники?$format=json&$filter=year(ДатаРождения) eq 1980
```

В результате в ответ сервера будут включены только данные сотрудников, удовлетворяющих условию отбора (листинг 1.92).

Листинг 1.92. Содержимое ответа сервера

```
{
"odata.metadata": "http://localhost/REST/odata/standard.odata/$metadata#Catalog_Сотрудники",
"value": [{
"Ref_Key": "9575f8ed-369f-11e9-8bb2-642737df2048",
"DataVersion": "AAAAAgAAAAA=",
"DeletionMark": false,
"Code": "000000001",
"Description": "Алексеев Сергей Иванович",
"ДатаРождения": "1980-12-10T00:00:00",
"КоличествоДетей": 1,
"Работает": true,
"Стаж": "10.05.01",
"Должность_Key": "88d54406-36a1-11e9-8bb2-642737df2048",
"Predefined": false,
"PredefinedDataName": "",
"Должность@navigationLinkUrl": "Catalog_Сотрудники(guid'9575f8ed-369f-11e9-8bb2-642737df2048')/Должность"
}]
}
```

Отобрать записи справочника по подстроке

Предположим, нам требуется отфильтровать записи справочника Сотрудники так, чтобы из списка отбирались сотрудники, имеющие в строке стажа значение 10 месяцев. Для описания условия отбора воспользуемся функцией `substring()`.

Наберем в адресной строке браузера следующий URL (листинг 1.93).

Листинг 1.93. URL запроса

```
localhost/REST/odata/standard.odata/Catalog_Сотрудники?$format=json&$filter=substring(Стаж, 4, 2) eq '10'
```

В результате в ответ сервера будут включены только данные сотрудников, удовлетворяющих условию отбора (листинг 1.94).

Листинг 1.94. Содержимое ответа сервера

```
{
"odata.metadata": "http://localhost/REST/odata/standard.odata/$metadata#Catalog_Сотрудники",
"value": [{
"Ref_Key": "0b6cd7db-36a0-11e9-8bb2-642737df2048",
"DataVersion": "AAAABAAAAAA=",
"DeletionMark": false,
"Code": "000000002",
"Description": "Смирнова Светлана Ивановна",
"ДатаРождения": "1990-02-22T00:00:00",
"КоличествоДетей": 0,
"Работает": false,
"Стаж": "03.10.27",
"Должность_Key": "88d54407-36a1-11e9-8bb2-642737df2048",
"Predefined": false,
"PredefinedDataName": "",
"Должность@navigationLinkUrl": "Catalog_Сотрудники(guid'0b6cd7db-36a0-11e9-8bb2-642737df2048')/Должность"
}]
}
```

Получение данных связанных сущностей (ссылок)

В листингах 1.92, 1.94 мы видим, что ссылочное поле Должность представлено в виде строкового значения ссылки. Но с помощью параметра `$expand` мы можем «развернуть» эти значения. Данный параметр позволяет вместе с результатами основного запроса получать значения связанных сущностей, что позволит не запрашивать каждую сущность отдельно.

Для этого наберем в адресной строке браузера следующий URL (листинг 1.95).

Листинг 1.95. URL запроса

```
localhost/REST/odata/standard.odata/Catalog_Сотрудники?$format=json&$select=Description
                                        , ДатаРождения, Должность&$expand=Должность
```

В результате в окне браузера мы увидим содержимое ответа сервера (листинг 1.96).

Листинг 1.96. Содержимое ответа сервера

```
{
"odata.metadata": "http://localhost/REST/odata/standard.odata/$metadata#Catalog_Сотрудники",
"value": [{
"Description": "Алексеев Сергей Иванович",
"ДатаРождения": "1980-12-10T00:00:00",
"Должность@navigationLinkUrl": "Catalog_Сотрудники(guid'9575f8ed-369f-11e9-8bb2-642737df2048')/Должность",
"Должность": {
"Ref_Key": "88d54406-36a1-11e9-8bb2-642737df2048",
"Predefined": false,
```

```
"PredefinedDataName": "",
"DataVersion": "AAAAAgAAAAA=",
"Description": "Администратор",
"Code": "000000002",
"DeletionMark": false
}
},{
"Description": "Смирнова Светлана Ивановна",
"ДатаРождения": "1990-02-22T00:00:00",
"Должность@navigationLinkUrl": "Catalog_Сотрудники(guid'0b6cd7db-36a0-11e9-8bb2-642737df2048')/Должность",
"Должность": {
"Ref_Key": "88d54407-36a1-11e9-8bb2-642737df2048",
"Predefined": false,
"PredefinedDataName": "",
"DataVersion": "AAAAAwAAAAA=",
"Description": "Кассир",
"Code": "000000003",
"DeletionMark": false
}
},{
"Description": "Артемов Игорь Владимирович",
"ДатаРождения": "1970-01-31T00:00:00",
"Должность@navigationLinkUrl": "Catalog_Сотрудники(guid'0b6cd7dc-36a0-11e9-8bb2-642737df2048')/Должность",
"Должность": {
"Ref_Key": "88d54405-36a1-11e9-8bb2-642737df2048",
"Predefined": false,
"PredefinedDataName": "",
"DataVersion": "AAAAAQAAAAA=",
"Description": "Менеджер",
"Code": "000000001",
"DeletionMark": false
}
}]
}
```


Получение данных из табличной части документа

Для получения данных из табличной части документа нужно к имени ресурса добавить имя табличной части. Например, в нашей демонстрационной базе существует документ ПоступлениеТоваров с табличной частью Товары, содержащей список поступивших товаров.

Для того чтобы увидеть только данные табличной части, наберем в адресной строке браузера следующий URL (листинг 1.97).

Листинг 1.97. URL запроса

```
localhost/REST/odata/standard.odata/Document_ПоступлениеТоваров_Товары?$format=json
```

В результате в окне браузера мы увидим содержимое ответа сервера (листинг 1.98).

Листинг 1.98. Содержимое ответа сервера


```
{
"odata.metadata": "http://localhost/REST/odata/standard.odata/$metadata#Document_ПоступлениеТоваров_Товары",
"value": [{
"Ref_Key": "34c5d79a-7636-11e9-88b7-642737df2048",
"LineNumber": "2",
"Товар_Key": "34c5d798-7636-11e9-88b7-642737df2048",
"Количество": 5,
"Цена": "2000",
"Сумма": "10000"
},{
"Ref_Key": "34c5d79a-7636-11e9-88b7-642737df2048",
"LineNumber": "1",
"Товар_Key": "34c5d799-7636-11e9-88b7-642737df2048",
"Количество": 3,
"Цена": "5000",
"Сумма": "15000"
},{
"Ref_Key": "34c5d79b-7636-11e9-88b7-642737df2048",
"LineNumber": "1",
"Товар_Key": "34c5d799-7636-11e9-88b7-642737df2048",
"Количество": 3,
"Цена": "5500",
"Сумма": "16500"
},{
"Ref_Key": "34c5d79b-7636-11e9-88b7-642737df2048",
"LineNumber": "2",
"Товар_Key": "34c5d797-7636-11e9-88b7-642737df2048",
"Количество": 3,
"Цена": "3300",
"Сумма": "9900"
},{
"Ref_Key": "34c5d79c-7636-11e9-88b7-642737df2048",
"LineNumber": "1",
"Товар_Key": "34c5d797-7636-11e9-88b7-642737df2048",
"Количество": 5,
"Цена": "3000",
"Сумма": "15000"
},{
"Ref_Key": "34c5d79c-7636-11e9-88b7-642737df2048",
"LineNumber": "2",
"Товар_Key": "34c5d798-7636-11e9-88b7-642737df2048",
"Количество": 5,
"Цена": "2000",
"Сумма": "10000"
}]
}
```


Как мы видим, в ответе сервера содержатся по две строки табличной части для каждого из трех документов ПоступлениеТоваров.

Чтобы получить данные конкретной строки табличной части документа, нужно иметь в виду, что у нее не одно, а два ключевых поля: ссылка на сам документ (`Ref _ Key`) и номер строки табличной части (`LineNumber`). И все эти поля, и их значения нужно перечислить через запятую в скобках после имени ресурса.

Итак, наберем в адресной строке браузера следующий URL (листинг 1.99).

Листинг 1.99. URL запроса

```
localhost/REST/odata/standard.odata/Document_ПоступлениеТоваров_Товары(
                    Ref_Key=guid'34c5d79b-7636-11e9-88b7-642737df2048', LineNumber=2)?$format=json
```

В результате в ответе сервера будут содержаться данные второй строки табличной части документа с указанной выше ссылкой (листинг 1.100).

Листинг 1.100. Содержимое ответа сервера

```
{
"odata.metadata": "http://localhost/REST/odata/standard.odata/$metadata#\
                                    Document_ПоступлениеТоваров_Товары/@Element",
"Ref_Key": "34c5d79b-7636-11e9-88b7-642737df2048",
"LineNumber": "2",
"Товар_Key": "34c5d797-7636-11e9-88b7-642737df2048",
"Количество": 3,
"Цена": "3300",
"Сумма": "9900"
}
```

Без указания номера строки табличной части никакие данные получены не будут, а в окне браузера мы увидим сообщение, что запрашиваемая страница не найдена.

Однако если указать значения ключевых полей с помощью параметра `$filter`, то в обоих случаях данные будут получены согласно условию отбора.

Составим условие отбора на равенство для обоих ключевых полей (листинг 1.101).

Листинг 1.101. URL запроса

```
localhost/REST/odata/standard.odata/Document_ПоступлениеТоваров_Товары?$format=json&$filter
                    =Ref_Key eq guid'34c5d79b-7636-11e9-88b7-642737df2048' and LineNumber eq 2
```

Мы получим тот же результат, что и раньше, то есть данные второй строки табличной части конкретного документа (см. листинг 1.100).

Опустим в условии отбора номер строки табличной части (листинг 1.102).

Листинг 1.102. URL запроса

```
localhost/REST/odata/standard.odata/Document_ПоступлениеТоваров_Товары?$format=json&$filter
                                        =Ref_Key eq guid'34c5d79b-7636-11e9-88b7-642737df2048'
```

Тогда в ответе сервера будут содержаться данные обеих строк табличной части документа с указанной в условии отбора ссылкой (листинг 1.103).

Листинг 1.103. Содержимое ответа сервера

```
{
"odata.metadata": "http://localhost/REST/odata/standard.odata/$metadata#Document_ПоступлениеТоваров_Товары",
"value": [{
"Ref_Key": "34c5d79b-7636-11e9-88b7-642737df2048",
"LineNumber": "1",
"Товар_Key": "34c5d799-7636-11e9-88b7-642737df2048",
"Количество": 3,
"Цена": "5500",
"Сумма": "16500"
},{
"Ref_Key": "34c5d79b-7636-11e9-88b7-642737df2048",
"LineNumber": "2",
"Товар_Key": "34c5d797-7636-11e9-88b7-642737df2048",
"Количество": 3,
"Цена": "3300",
"Сумма": "9900"
}]
}
```

Условие отбора по реквизитам табличных частей

При получении данных через REST-интерфейс можно не только выполнять отбор по реквизитам объектов конфигурации и их табличных частей, но и накладывать условия отбора на элементы коллекций. Для этого необходимо использовать лямбда-функции. Система поддерживает использование следующих лямбда-функций:

- `any` – применяет логическое выражение к каждому элементу коллекции и возвращает значение `true`, если хоть один элемент коллекции удовлетворяет этому условию. Лямбда-функция `any` без аргументов возвращает `true`, если коллекция не пуста;
- `all` – применяет логическое выражение к каждому элементу коллекции и возвращает значение `true`, если все элементы коллекции ему удовлетворяют.

Например, нам нужно получить список документов ПоступлениеТоваров, у которых цена хотя бы одного поступившего товара (значение реквизита Цена табличной части Товары) больше 3000.

Для этого наберем в адресной строке браузера следующий URL (листинг 1.104).

Листинг 1.104. URL запроса

```
localhost/REST/odata/standard.odata/Document_ПоступлениеТоваров?$filter
                                                    =Товары/any(d: d/Цена gt 3000)&$format=json
```

В результате в ответе сервера будут содержаться данные двух из трех документов (`Number: 000000001, 000000002`), в состав которых входит хотя бы одна строка табличной части, удовлетворяющая условию (листинг 1.105).

Листинг 1.105. Содержимое ответа сервера

```
{
"odata.metadata": "http://localhost/REST/odata/standard.odata/$metadata#Document_ПоступлениеТоваров",
"value": [{
"Ref_Key": "34c5d79a-7636-11e9-88b7-642737df2048",
"DataVersion": "AAAABQAAAAA=",
"DeletionMark": false,
"Number": "000000001",
"Date": "2019-05-10T12:00:00",
"Posted": true,
"Поставщик_Key": "9cfabfb1-4cb2-11e9-9334-642737df2048",
"Товары": [
{
"Ref_Key": "34c5d79a-7636-11e9-88b7-642737df2048",
"LineNumber": "1",
"Товар_Key": "34c5d799-7636-11e9-88b7-642737df2048",
"Количество": 3,
"Цена": "5000",
"Сумма": "15000"
},
{
"Ref_Key": "34c5d79a-7636-11e9-88b7-642737df2048",
"LineNumber": "2",
"Товар_Key": "34c5d798-7636-11e9-88b7-642737df2048",
"Количество": 5,
"Цена": "2000",
"Сумма": "10000"
}
],
"Поставщик@navigationLinkUrl": "Document_ПоступлениеТоваров(
                                guid'34c5d79a-7636-11e9-88b7-642737df2048')/Поставщик"
},{
"Ref_Key": "34c5d79b-7636-11e9-88b7-642737df2048",
"DataVersion": "AAAABgAAAAA=",
"DeletionMark": false,
"Number": "000000002",
"Date": "2019-05-13T12:00:00",
"Posted": true,
"Поставщик_Key": "9cfabfb0-4cb2-11e9-9334-642737df2048",
"Товары": [
```

```
{
"Ref_Key": "34c5d79b-7636-11e9-88b7-642737df2048",
"LineNumber": "1",
"Товар_Key": "34c5d799-7636-11e9-88b7-642737df2048",
"Количество": 3,
"Цена": "5500",
"Сумма": "16500"
},
{
"Ref_Key": "34c5d79b-7636-11e9-88b7-642737df2048",
"LineNumber": "2",
"Товар_Key": "34c5d797-7636-11e9-88b7-642737df2048",
"Количество": 3,
"Цена": "3300",
"Сумма": "9900"
}
],
"Поставщик@navigationLinkUrl": "Document_ПоступлениеТоваров(
                                guid'34c5d79b-7636-11e9-88b7-642737df2048')/Поставщик"
}]
}
```


А теперь предположим, что нам нужно получить список документов ПоступлениеТоваров, у которых цена каждого поступившего товара (значение реквизита Цена табличной части Товары) больше 3000.

Для этого наберем в адресной строке браузера следующий URL (листинг 1.106).

Листинг 1.106. URL запроса

```
localhost/REST/odata/standard.odata/Document_ПоступлениеТоваров?$filter
                                                    =Товары/all(d: d/Цена gt 3000)&$format=json
```

В результате в ответе сервера будут содержаться данные только одного документа (`Number: 000000002`), в составе которого все строки табличной части удовлетворяют заданному условию (листинг 1.107).

Листинг 1.107. Содержимое ответа сервера


```
{
"odata.metadata": "http://localhost/REST/odata/standard.odata/$metadata#Document_ПоступлениеТоваров",
"value": [{
"Ref_Key": "34c5d79b-7636-11e9-88b7-642737df2048",
"DataVersion": "AAAABgAAAAA=",
"DeletionMark": false,
"Number": "000000002",
"Date": "2019-05-13T12:00:00",
"Posted": true,
"Поставщик_Key": "9cfabfb0-4cb2-11e9-9334-642737df2048",
"Товары": [
{
```

```
"Ref_Key": "34c5d79b-7636-11e9-88b7-642737df2048",
"LineNumber": "1",
"Товар_Key": "34c5d799-7636-11e9-88b7-642737df2048",
"Количество": 3,
"Цена": "5500",
"Сумма": "16500"
},
{
"Ref_Key": "34c5d79b-7636-11e9-88b7-642737df2048",
"LineNumber": "2",
"Товар_Key": "34c5d797-7636-11e9-88b7-642737df2048",
"Количество": 3,
"Цена": "3300",
"Сумма": "9900"
}
],
"Поставщик@navigationLinkUrl": "Document_ПоступлениеТоваров(
                                guid'34c5d79b-7636-11e9-88b7-642737df2048')/Поставщик"
}]
}
```


Для сравнения: если мы наложим условие непосредственно на значение реквизита Цена табличной части Товары (`Товары/Цена`), то получим только те строки табличной части, которые удовлетворяют заданному условию (листинг 1.108, 1.109).

Листинг 1.108. URL запроса

```
localhost/REST/odata/standard.odata/Document_ПоступлениеТоваров?$filter
                                                    =Товары/Цена gt 3000&$format=json
```

Листинг 1.109. Содержимое ответа сервера


```
{
"odata.metadata": "http://localhost/REST/odata/standard.odata/$metadata#Document_ПоступлениеТоваров",
"value": [{
"Ref_Key": "34c5d79a-7636-11e9-88b7-642737df2048",
"DataVersion": "AAAABQAAAAA=",
"DeletionMark": false,
"Number": "000000001",
"Date": "2019-05-10T12:00:00",
"Posted": true,
"Поставщик_Key": "9cfabfb1-4cb2-11e9-9334-642737df2048",
"Товары": [
{
"Ref_Key": "34c5d79a-7636-11e9-88b7-642737df2048",
"LineNumber": "1",
"Товар_Key": "34c5d799-7636-11e9-88b7-642737df2048",
"Количество": 3,
"Цена": "5000",
"Сумма": "15000"
}
```

```
],
"Поставщик@navigationLinkUrl": "Document_ПоступлениеТоваров(
                                                guid'34c5d79a-7636-11e9-88b7-642737df2048')/Поставщик"
},{
"Ref_Key": "34c5d79b-7636-11e9-88b7-642737df2048",
"DataVersion": "AAAABgAAAAA=",
"DeletionMark": false,
"Number": "000000002",
"Date": "2019-05-13T12:00:00",
"Posted": true,
"Поставщик_Key": "9cfabfb0-4cb2-11e9-9334-642737df2048",
"Товары": [
{
"Ref_Key": "34c5d79b-7636-11e9-88b7-642737df2048",
"LineNumber": "1",
"Товар_Key": "34c5d799-7636-11e9-88b7-642737df2048",
"Количество": 3,
"Цена": "5500",
"Сумма": "16500"
},
{
"Ref_Key": "34c5d79b-7636-11e9-88b7-642737df2048",
"LineNumber": "2",
"Товар_Key": "34c5d797-7636-11e9-88b7-642737df2048",
"Количество": 3,
"Цена": "3300",
"Сумма": "9900"
}
],
"Поставщик@navigationLinkUrl": "Document_ПоступлениеТоваров(
                                                guid'34c5d79b-7636-11e9-88b7-642737df2048')/Поставщик"
}]
}
```


Получение данных регистра сведений

В нашей демонстрационной базе существует регистр сведений ЦеныТоваров. Этот регистр сведений содержит следующие данные (рис. 1.24).

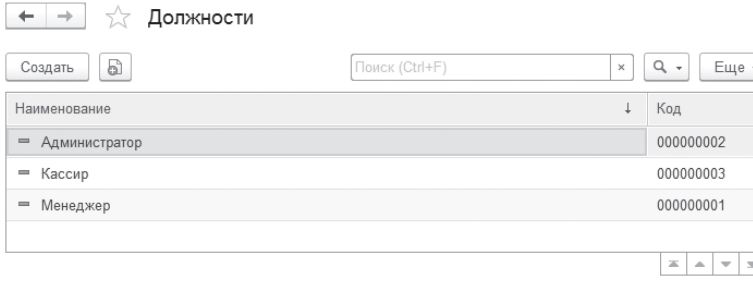


Рис. 1.24. Данные регистра сведений «ЦеныТоваров»

Предположим, нам требуется получить срез последних записей регистра сведений ЦеныТоваров, то есть получить последние актуальные цены товаров на текущую дату. Для этого воспользуемся функцией `SliceLast()`.

Наберем в адресной строке браузера следующий URL (листинг 1.110).

Листинг 1.110. URL запроса

```
localhost/REST/odata/standard.odata/InformationRegister_ЦеныТоваров/SliceLast()
                                    ?$format=json&$expand=Товар&$select=Period, Товар/Description, Цена
```

Для наглядности мы получили связанные данные из справочника товаров с помощью параметра `$expand` и вывели название товара (`Товар/Description`) в списке полей.

В результате в окне браузера мы увидим содержимое ответа сервера (листинг 1.111).

Листинг 1.111. Содержимое ответа сервера

```
{
"odata.metadata": "http://localhost/REST/odata/standard.odata/$metadata#InformationRegister_ЦеныТоваров",
"value": [{
"Period": "2019-07-21T00:00:00",
"Цена": 3630,
"Товар@navigationLinkUrl": "InformationRegister_ЦеныТоваров(
        Period=datetime'2019-07-21T00:00:00', Товар_Key=guid'34c5d797-7636-11e9-88b7-642737df2048')/Товар",
"Товар": {
"Description": "Чайник"
}
},{
"Period": "2019-07-21T00:00:00",
"Цена": 6050,
"Товар@navigationLinkUrl": "InformationRegister_ЦеныТоваров(
        Period=datetime'2019-07-21T00:00:00', Товар_Key=guid'34c5d799-7636-11e9-88b7-642737df2048')/Товар",
"Товар": {
"Description": "Кофемашина"
}
},{
"Period": "2019-07-21T00:00:00",
"Цена": 2420,
"Товар@navigationLinkUrl": "InformationRegister_ЦеныТоваров(
        Period=datetime'2019-07-21T00:00:00', Товар_Key=guid'34c5d798-7636-11e9-88b7-642737df2048')/Товар",
"Товар": {
"Description": "Тостер"
}
}]
}
```

Теперь для закрепления ранее полученных знаний выведем все записи регистра сведений для конкретного товара (кофемашина), при этом историю назначения цен на товар расположим в порядке убывания дат. Для этого отфильтруем записи регистра сведений по измерению Товар (ключевое поле `Товар _ Key`).

Наберем в адресной строке браузера следующий URL (листинг 1.112).

Листинг 1.112. URL запроса

```
localhost/REST/odata/standard.odata/InformationRegister_ЦеныТоваров?$filter
        =Товар_Key eq guid'34c5d799-7636-11e9-88b7-642737df2048'&$format=json&$expand
        =Товар&$select=Period, Товар/Description, Цена&$orderby=Period desc
```

В результате в окне браузера мы увидим содержимое ответа сервера (листинг 1.113).

Листинг 1.113. Содержимое ответа сервера

```
{
"odata.metadata": "http://localhost/REST/odata/standard.odata/$metadata#InformationRegister_ЦеныТоваров",
"value": [{
"Period": "2019-07-21T00:00:00",
"Цена": 6050,
"Товар@navigationLinkUrl": "InformationRegister_ЦеныТоваров(
        Period=datetime'2019-07-21T00:00:00', Товар_Key=guid'34c5d799-7636-11e9-88b7-642737df2048')/Товар",
"Товар": {
"Description": "Кофемашина"
}
},{
"Period": "2019-05-13T00:00:00",
"Цена": 5500,
"Товар@navigationLinkUrl": "InformationRegister_ЦеныТоваров(
        Period=datetime'2019-05-13T00:00:00', Товар_Key=guid'34c5d799-7636-11e9-88b7-642737df2048')/Товар",
"Товар": {
"Description": "Кофемашина"
}
},{
"Period": "2019-05-07T00:00:00",
"Цена": 5000,
"Товар@navigationLinkUrl": "InformationRegister_ЦеныТоваров(
        Period=datetime'2019-05-07T00:00:00', Товар_Key=guid'34c5d799-7636-11e9-88b7-642737df2048')/Товар",
"Товар": {
"Description": "Кофемашина"
}
}]
}
```

Модификация данных прикладного решения

До сих пор мы рассматривали примеры получения данных через REST-интерфейс, то есть чтения данных с сервера методом `GET`. Теперь мы покажем, как можно изменять, добавлять и удалять данные информационной базы.

Как уже говорилось, запросы на чтение данных можно выполнять непосредственно в браузере. Остальные действия, модифицирующие данные информационной базы, не могут быть выполнены подобным образом.

Для этого нужно записать изменяемые или добавляемые данные в строку JSON, затем установить HTTP-соединение с сервером, на котором опубликован интерфейс OData для прикладного решения. После этого создать HTTP-запрос на основе URL модифицируемых данных, установить тело HTTP-запроса из строки JSON и отправить запрос к серверу с соответствующим методом (`POST`, `PUT/PATCH`, `DELETE`).

Поясним вышесказанное на небольших примерах.

Добавление данных

Предположим, нам нужно добавить новую запись в справочник Должности.

Обработчик команды, добавляющей данные, заполним следующим образом (листинг 1.114).

Листинг 1.114. Обработчик команды «ДобавитьДанные»

```
&НаКлиенте
Процедура ДобавитьДанные(Команда)

    Сообщение = Новый СообщениеПользователю;

    // Сформировать строку URL.
    АдресРесурса = "/REST/odata/standard.odata/Catalog_Должности?$format=json";

    // Получить строку JSON с добавляемыми данными.
    СтрокаТелаЗапроса = СформироватьДобавляемыеДанные();
    Сообщение.Текст = СтрокаТелаЗапроса;
    Сообщение.Сообщить();

    // Отправить запрос на сервер.
    ВыполнитьЗапрос("POST", АдресРесурса, СтрокаТелаЗапроса);

КонецПроцедуры
```

В этом обработчике в переменной `АдресРесурса` мы формируем URL запроса для обращения к данным нашего справочника.

Затем с помощью функции `СформироватьДобавляемыеДанные()`, которая будет рассмотрена ниже, в листинге 1.116, мы записываем добавляемые данные в строку JSON и возвращаем ее в переменную `СтрокаТелаЗапроса`.

После этого вызываем процедуру `ВыполнитьЗапрос()` и передаем в нее в качестве параметров адрес ресурса, строку тела запроса и имя метода («POST»), который надо выполнить на сервере (листинг 1.115).

Листинг 1.115. Процедура «ВыполнитьЗапрос»

```
&НаКлиенте
Процедура ВыполнитьЗапрос(ИмяМетода, АдресРесурса, СтрокаТелаЗапроса)

	Сообщение = Новый СообщениеПользователю;
	// Установить имя сервера.
	СерверИсточник = "localhost";

	Попытка
		// Создать HTTP-соединение с сервером localhost.
		HTTPСоединение = Новый HTTPСоединение(СерверИсточник);
	Исключение
		// Вывести сообщение об ошибке соединения с сервером.
		Сообщение.Текст = "Не удалось соединиться с сервером: " + СерверИсточник;
		Сообщение.Сообщить();
		Сообщение.Текст = ОписаниеОшибки();
		Сообщение.Сообщить();
		Возврат;
	КонецПопытки;

	// Создать HTTP-запрос на основе URL.
	HTTPЗапрос = Новый HTTPЗапрос(АдресРесурса);

	// Установить тело запроса из строки JSON.
	Если ИмяМетода <> "DELETE" И ИмяМетода <> "GET" Тогда
		HTTPЗапрос.УстановитьТелоИзСтроки(СтрокаТелаЗапроса);
	КонецЕсли;

	Попытка
		// Получить ответ сервера в виде объекта HTTPОтвет.
		Результат = HTTPСоединение.ВызватьHTTPМетод(ИмяМетода, HTTPЗапрос);
		//получить содержимое ответа сервера в виде строки
		Сообщение.Текст = Результат.ПолучитьТелоКакСтроку();
		Сообщение.Сообщить();
	Исключение
		// Вывести сообщение об ошибке при получении ответа сервера.
		Сообщение.Текст = ОписаниеОшибки();
		Сообщение.Сообщить();
		Возврат;
	КонецПопытки;

КонецПроцедуры
```

В этой процедуре в качестве имени сервера мы указываем «localhost», создаем на его основе объект `HTTPСоединение` и соединяемся с этим сервером.

Далее на основе URL, содержащегося в параметре `АдресРесурса`, создаем `HTTPЗапрос`. Затем, в случае если это не DELETE- и не GET-запрос, устанавливаем тело запроса из строки JSON, содержащейся в параметре `СтрокаТелаЗапроса`.

После этого при помощи метода `ВызватьHTTPМетод()` объекта `HTTPСоединение` отправляем на сервер POST-запрос к информационной базе. Ответ сервера в виде объекта `HTTPОтвет` будет возвращен в переменную `Результат`. С помощью метода `ПолучитьТелоКакСтроку()` объекта `HTTPОтвет` мы получаем тело ответа в виде строки и выводим его в сообщение.

В результате в справочник будут добавлены новые данные, записанные в строку JSON в функции `СформироватьДобавляемыеДанные()`. Заполним эту функцию следующим образом (листинг 1.116).

Листинг 1.116. Функция «СформироватьДобавляемыеДанные»

```
&НаСервереБезКонтекста
Функция СформироватьДобавляемыеДанные()

    // Создать объект записи и записать строковое значение в строку JSON.
    Запись = Новый ЗаписьJSON;
    Запись.УстановитьСтроку();

    // Записать начало корневого объекта.
    Запись.ЗаписатьНачалоОбъекта();

    // Записать наименование новой должности.
    Запись.ЗаписатьИмяСвойства("Description");
    Запись.ЗаписатьЗначение("Новая должность - REST");

    // Записать конец корневого объекта.
    Запись.ЗаписатьКонецОбъекта();

    СтрокаJSON = Запись.Закрыть();

    Возврат СтрокаJSON;

КонецФункции
```

В этой функции методом потоковой записи с помощью объекта `ЗаписьJSON` мы записываем в строку JSON наименование (свойство `Description`) новой должности, которую мы хотим добавить в справочник Должности.

В результате при выполнении команды по добавлению данных на сервер будет отправлен POST-запрос к информационной базе. Ответ сервера в виде строки будет содержать данные нового элемента, добавленного в справочник (рис. 1.25).

Рис. 1.25. Ответ сервера

Таким образом, содержимое справочника теперь выглядит следующим образом (рис. 1.26).

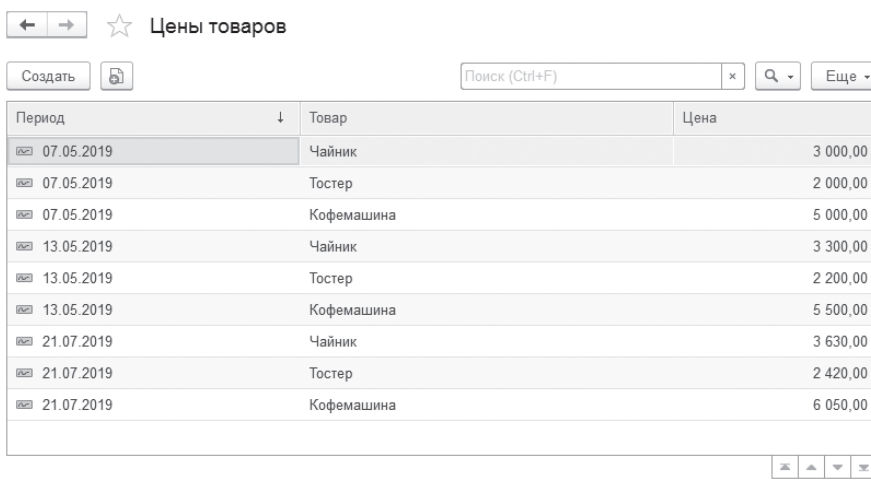


Рис. 1.26. Данные справочника «Должности»

Изменение данных

Теперь предположим, что нам нужно изменить некоторые поля (а именно: Код, ДатаРождения и Должность) у конкретного элемента справочника Сотрудники. Поскольку мы хотим изменить не все, а только часть полей записи справочника, для этого нужно обратиться к элементу справочника с конкретной ссылкой и отправить на сервер PATCH-запрос с изменяемыми данными.

Обработчик команды, изменяющей данные, заполним следующим образом (листинг 1.117).

Листинг 1.117. Обработчик команды «ИзменитьДанные»

```
&НаКлиенте
Процедура ИзменитьДанные(Команда)

    Сообщение = Новый СообщениеПользователю;

    // Сформировать строку URL.
    АдресРесурса = "/REST/odata/standard.odata/Catalog_Сотрудники(
                                guid'0b6cd7dc-36a0-11e9-8bb2-642737df2048')?$format=json";

    // Получить строку JSON с изменяемыми данными.
    СтрокаТелаЗапроса = СформироватьИзменяемыеДанные();
    Сообщение.Текст = СтрокаТелаЗапроса;
    Сообщение.Сообщить();

    // Отправить запрос на сервер.
    ВыполнитьЗапрос("PATCH", АдресРесурса, СтрокаТелаЗапроса);

КонецПроцедуры
```

В этом обработчике в переменной `АдресРесурса` мы формируем URL запроса для обращения к данным конкретного элемента нашего справочника. В скобках мы указываем ссылку на элемент, которую мы для простоты копируем из рассмотренных выше примеров чтения данных из этого справочника.

Затем с помощью функции `СформироватьИзменяемыеДанные()`, которая будет рассмотрена ниже, в листинге 1.118, мы записываем изменяемые данные в строку JSON и возвращаем ее в переменную `СтрокаТелаЗапроса`.

После этого вызываем процедуру `ВыполнитьЗапрос()` и передаем в нее в качестве параметров адрес ресурса, строку тела запроса и имя метода («PATCH»), который надо выполнить на сервере. Поскольку эта процедура уже была рассмотрена в листинге 1.115, не будем еще раз на этом останавливаться.

В результате у конкретного сотрудника (Артемов Игорь Владимирович) будут изменены те данные, которые мы запишем в строку JSON в функции `СформироватьИзменяемыеДанные()` следующим образом (листинг 1.118).

Листинг 1.118. Функция «СформироватьИзменяемыеДанные»

```
&НаСервереБезКонтекста
Функция СформироватьИзменяемыеДанные()

	// Создать объект записи и записать строковое значение в строку JSON.
	Запись = Новый ЗаписьJSON;
	Запись.УстановитьСтроку();

	// Записать начало корневого объекта.
	Запись.ЗаписатьНачалоОбъекта();

	// Изменить свойство Code.
	Запись.ЗаписатьИмяСвойства("Code");
	Запись.ЗаписатьЗначение("REST-0003");

	// Изменить свойство ДатаРождения.
	Запись.ЗаписатьИмяСвойства("ДатаРождения");
	Запись.ЗаписатьЗначение(ЗаписатьДатуJSON(ТекущаяДата()
	                                , ФорматДатыJSON.ISO, ВариантЗаписиДатыJSON.УниверсальнаяДата));

	// Изменить свойство ссылочное поле Должность.
	Запись.ЗаписатьИмяСвойства("Должность@odata.bind");
	Запись.ЗаписатьЗначение("Catalog_Должности(guid'524ffe8a-789f-11e9-babb-642737df2048')");

	// Записать конец корневого объекта.
	Запись.ЗаписатьКонецОбъекта();

	СтрокаJSON = Запись.Закрыть();

	Возврат СтрокаJSON;

КонецФункции
```

В этой функции методом потоковой записи с помощью объекта `ЗаписьJSON` мы записываем в строку JSON значения полей Код (свойство `Code`), ДатаРождения и Должность, которые мы хотим изменить у конкретного элемента справочника Сотрудники.

Код – это строковое поле, поэтому в свойство `Code` мы просто записываем строку «REST-0003». В поле ДатаРождения с помощью функции `ЗаписатьДатуJSON()` мы записываем текущую дату как универсальную дату в формате ISO. Поле Должность – это ссылочное поле, которое ссылается на справочник Должности, поэтому мы указываем для него имя свойства как `"Должность@odata.bind"`, а значение – `"Catalog_Должности(guid'524ffe8a-789f-11e9-babb-642737df2048')"`.

Значение уникального идентификатора этой должности мы для простоты копируем из предыдущего примера (см. рис. 1.25), когда мы добавляли новую запись в справочник Должности. Таким образом, наименование должности у сотрудника теперь изменится с Менеджер на Новая должность – REST (см. рис. 1.28).

В результате при выполнении команды по изменению данных на сервер будет отправлен PATCH-запрос к информационной базе. Ответ сервера в виде строки будет содержать данные измененного элемента справочника (рис. 1.27).

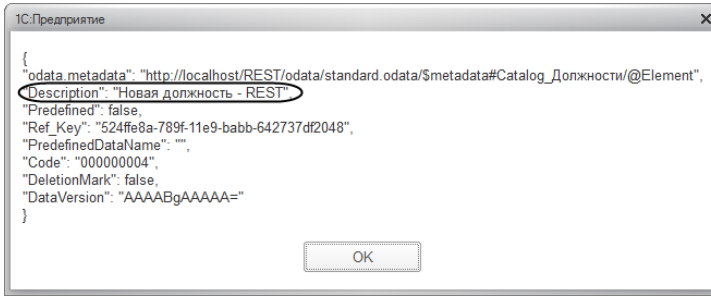


Рис. 1.27. Ответ сервера

Таким образом, содержимое справочника теперь выглядит следующим образом (рис. 1.28).

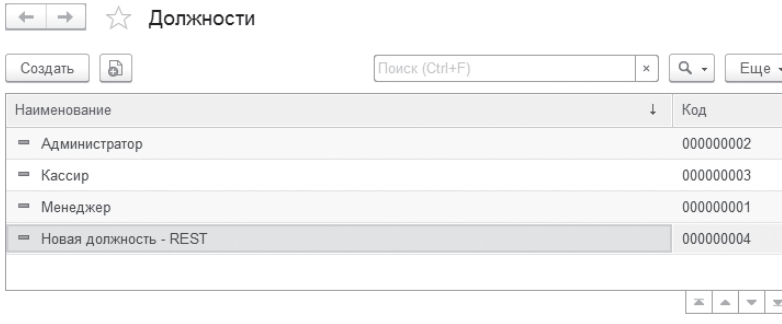


Рис. 1.28. Данные справочника «Сотрудники»

Удаление данных

Теперь предположим, что нам нужно удалить конкретную запись из справочника Должности. Для этого нужно обратиться к элементу справочника с конкретной ссылкой и отправить на сервер DELETE-запрос.

Обработчик команды, удаляющей данные, заполним следующим образом (листинг 1.119).

Листинг 1.119. Обработчик команды «УдалитьДанные»

```
&НаКлиенте
Процедура УдалитьДанные(Команда)

    // Сформировать строку URL.
    АдресРесурса = "/REST/odata/standard.odata/Catalog_Должности(
                                guid'4f158949-78ab-11e9-babb-642737df2048')?$format=json";

    // Отправить запрос на сервер.
    ВыполнитьЗапрос("DELETE", АдресРесурса, "");

КонецПроцедуры
```

В этом обработчике в переменной `АдресРесурса` мы формируем URL запроса для обращения к данным конкретного элемента нашего справочника. Чтобы получить ссылку на элемент справочника Должности, который необходимо удалить, выполним еще раз команду Добавить данные. Затем из тела ответа сервера скопируем ссылку на добавленную должность и вставим ее в адрес ресурса.

После этого вызываем процедуру `ВыполнитьЗапрос()` и передаем в нее в качестве параметров адрес ресурса и имя метода («DELETE»), который надо выполнить на сервере. Поскольку эта процедура уже была рассмотрена в листинге 1.115, не будем еще раз на этом останавливаться. Подчеркнем только, что для DELETE- и GET-запросов тело запроса устанавливать не нужно, поэтому третьим параметром мы передаем в процедуру пустую строку.

В результате запись с указанным идентификатором будет удалена из справочника Должности, а тело ответа сервера будет пустым.

Изменение данных табличной части

В заключение покажем, как изменять данные табличных частей.

Предположим, нам нужно изменить документ ПоступлениеТоваров, включая его табличную часть. В самом документе мы изменим номер и дату. Для этого нам нужно отправить на сервер PATCH-запрос к конкретному

документу. Однако данные табличной части нужно перезаписывать полностью, несмотря на то что мы хотели изменить только некоторые поля или строки этой табличной части.

Обработчик команды, изменяющей данные, заполним следующим образом (листинг 1.120).

Листинг 1.120. Обработчик команды «ЗаписатьДанные»

```
&НаКлиенте
Процедура ЗаписатьДанные(Команда)

    Сообщение = Новый СообщениеПользователю;

    // Сформировать строку URL.
    АдресРесурса = "/REST/odata/standard.odata/Document_ПоступлениеТоваров(
                                        guid'34c5d79c-7636-11e9-88b7-642737df2048')?$format=json";

    // Получить строку JSON с изменяемыми данными.
    СтрокаТелаЗапроса = СформироватьИзменяемыеДанныеТЧ();

    // Отправить запрос на сервер.
    ВыполнитьЗапрос("PATCH", АдресРесурса, СтрокаТелаЗапроса);

КонецПроцедуры
```

В этом обработчике в переменной `АдресРесурса` мы формируем URL, на основе которого будет создан HTTP-запрос для обращения к данным конкретной строки документа, которую мы хотим изменить. О правилах формирования URL ресурсов, доступ к которым осуществляется через REST-интерфейс, более подробно рассказывается в разделе «Автоматический REST-интерфейс – Правила формирования URL запроса».

Затем с помощью функции `СформироватьИзменяемыеДанныеТЧ()`, которая будет рассмотрена ниже, в листинге 1.121, мы записываем изменяемые данные в строку JSON и возвращаем ее в переменную `СтрокаТелаЗапроса`.

В этом обработчике мы вызываем процедуру `ВыполнитьЗапрос()` и передаем в нее в качестве параметров адрес ресурса, строку тела запроса и имя метода («PATCH»), который надо выполнить на сервере. Поскольку эта процедура уже была рассмотрена в листинге 1.115, не будем еще раз на этом останавливаться.

В результате у конкретного документа ПоступлениеТоваров будут изменены те данные, которые мы запишем в строку JSON в функции `СформироватьИзменяемыеДанныеТЧ()` следующим образом (листинг 1.121).

Листинг 1.121. Функция «СформироватьИзменяемыеДанныеТЧ»

```
&НаСервереБезКонтекста
Функция СформироватьИзменяемыеДанныеТЧ()

	// Создать объект записи и записать строковое значение в строку JSON.
	Запись = Новый ЗаписьJSON;
	Запись.УстановитьСтроку();

	// Записать начало корневого объекта.
	Запись.ЗаписатьНачалоОбъекта();

	// Изменить свойство Номер.
	Запись.ЗаписатьИмяСвойства("Number");
	Запись.ЗаписатьЗначение("000000777");

	// Изменить свойство Дата.
	Запись.ЗаписатьИмяСвойства("Date");
	Запись.ЗаписатьЗначение(ЗаписатьДатуJSON(ТекущаяДата()
			, ФорматДатыJSON.ISO, ВариантЗаписиДатыJSON.УниверсальнаяДата));

	// Изменить состав ТЧ Товары.
	Запись.ЗаписатьИмяСвойства("Товары@odata.type");
	Запись.ЗаписатьЗначение("Collection(
			StandardODATA.Document_ПоступлениеТоваров_Товары_RowType)");

	Запись.ЗаписатьИмяСвойства("Товары");
	Запись.ЗаписатьНачалоМассива();

	// записать полностью первую строку ТЧ.
	Запись.ЗаписатьНачалоОбъекта();
	Запись.ЗаписатьИмяСвойства("LineNumber");
	Запись.ЗаписатьЗначение("1");
	Запись.ЗаписатьИмяСвойства("Количество");
	Запись.ЗаписатьЗначение("1");
	Запись.ЗаписатьИмяСвойства("Цена");
	Запись.ЗаписатьЗначение("10000");
	Запись.ЗаписатьИмяСвойства("Сумма");
	Запись.ЗаписатьЗначение("10000");
	Запись.ЗаписатьИмяСвойства("Товар_Key");
	Запись.ЗаписатьЗначение("34c5d799-7636-11e9-88b7-642737df2048");
	Запись.ЗаписатьКонецОбъекта();

	// записать полностью вторую строку ТЧ.
	Запись.ЗаписатьНачалоОбъекта();
	Запись.ЗаписатьИмяСвойства("LineNumber");
	Запись.ЗаписатьЗначение("2");
	Запись.ЗаписатьИмяСвойства("Количество");
	Запись.ЗаписатьЗначение("2");
	Запись.ЗаписатьИмяСвойства("Цена");
	Запись.ЗаписатьЗначение("5000");
	Запись.ЗаписатьИмяСвойства("Сумма");
	Запись.ЗаписатьЗначение("10000");
	Запись.ЗаписатьИмяСвойства("Товар_Key");
	Запись.ЗаписатьЗначение("34c5d798-7636-11e9-88b7-642737df2048");
```

```
        Запись.ЗаписатьКонецОбъекта();

        Запись.ЗаписатьКонецМассива();

        // Записать конец корневого объекта.
        Запись.ЗаписатьКонецОбъекта();

        СтрокаJSON = Запись.Закрыть();

        Возврат СтрокаJSON;

КонецФункции
```

В этой функции методом потоковой записи с помощью объекта `ЗаписьJSON` мы записываем в строку JSON значения полей Номер (свойство `Number`), Дата (свойство `Date`) и все поля и строки табличной части Товары, которые мы хотим изменить у конкретного документа ПоступлениеТоваров.

Номер – это строковое поле, поэтому в свойство `Number` мы просто записываем строку «000000777». В поле Дата (свойство `Date`) с помощью функции `ЗаписатьДатуJSON()` мы записываем текущую дату как универсальную дату в формате ISO.

Поскольку мы собираемся изменять состав табличной части Товары, мы указываем имя свойства этой коллекции значений как `"Товары@odata.type"`, а значение – `"Collection(StandardODATA.Document_ПоступлениеТоваров_Товары_RowType)"`. Затем записываем в свойство Товары массив, каждый элемент которого содержит объект, полностью описывающий каждую строку табличной части.

В результате при выполнении команды по изменению данных на сервер будет отправлен PATCH-запрос к информационной базе. Ответ сервера в виде строки будет содержать данные измененного документа (рис. 1.29).

Таким образом, содержимое документа будет выглядеть следующим образом (рис. 1.30).

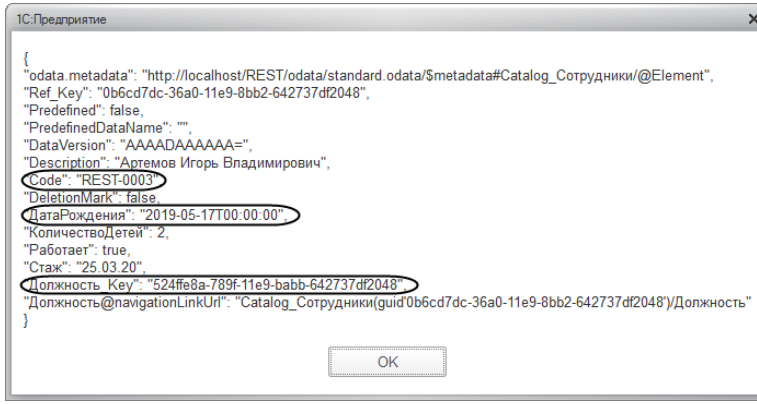


Рис. 1.29. Ответ сервера

Рис. 1.30. Данные измененного документа «Поступление товаров»

Типичные ошибки при получении данных

В этом разделе мы рассмотрим типичные ошибки, возникающие при обращении к REST-интерфейсу прикладного решения. Ошибки могут возникать как на стороне клиентского приложения, так и на стороне сервера. В случае возникновения ошибки на клиенте сервер пытается уточнить причину ошибки и может передать клиентскому приложению дополнительный внутренний код ошибки и информационное сообщение в теле ответа.

В первую очередь нужно понять, работает ли вообще REST-интерфейс с интересующей нас информационной базой. Для этого достаточно в браузере просто набрать URL: `<путь публикации базы>/odata/standard.odata`. Должен вернуться следующий ответ сервера (листинг 1.122).

Листинг 1.122. Ответ сервера

```
<service xmlns="http://www.w3.org/2007/app" xmlns:atom="http://www.w3.org/2005/Atom" xml:base
                                                    ="http://localhost/EXT/odata/standard.odata">
<workspace>
<atom:title>Default</atom:title>
</workspace>
</service>
```

Этот ответ говорит о том, что интерфейс работает, но для интерфейса OData недоступны никакие объекты конфигурации.

Ниже мы рассмотрим несколько типичных ошибок, которые возникают в случае неправильного написания URL.

Например, при указании адреса ресурса вы ошиблись в имени объекта конфигурации (листинг 1.123).

Листинг 1.123. URL запроса

```
localhost/REST/odata/standard.odata/CatalogДолжности?$format=json
```

В ответе сервера будет возвращен код ошибки – 8 и описание ошибки – «Тип сущности не найден» (листинг 1.124).

Листинг 1.124. Ответ сервера

```
{
"odata.error": {
"code": "8",
"message": {
"lang": "ru",
"value": "\"CatalogПоставщики\" entity is not found"
}
}
}
```

Если при указании адреса ресурса вы ошиблись в значении ссылки на конкретный объект конфигурации (листинг 1.125).

Листинг 1.125. URL запроса

```
localhost/REST/odata/standard.odata/Catalog_Должности(
                                guid'88d54406-36a1-11e9-8bb2-642737df2049')?$format=json&
```

В ответе сервера будет возвращен код ошибки – 9 и описание ошибки – «Экземпляр сущности не найден» (листинг 1.126).

Листинг 1.126. Ответ сервера

```
{
"odata.error": {
"code": "9",
"message": {
"lang": "ru",
"value": "Entity instance not found"
}
}
}
```

Если при указании выражения фильтра вы ошиблись, например, при написании функции отбора (листинг 1.127).

Листинг 1.127. URL запроса

```
localhost/REST/odata/standard.odata/Catalog_Поставщики?$format=json&$filter=startwith(
                                                    Description, 'ООО') eq true
```

В ответе сервера будет возвращен код ошибки – 14 и описание ошибки – «Ошибка разбора опций запроса» (листинг 1.128).

Листинг 1.128. Ответ сервера

```
{
"odata.error": {
"code": "14",
"message": {
"lang": "ru",
"value": "Error parsing $filter query option"
}
}
}
```

Если при указании адреса ресурса вы ошиблись, например, в имени параметра (листинг 1.129).

Листинг 1.129. URL запроса

```
localhost/REST/odata/standard.odata/Catalog_Сотрудники?$format=json&$select=Description
                                                , ДатаРождения, Должность&$expan=Должность
```

В ответе сервера будет возвращен код ошибки – 0 и описание ошибки – «Параметр не поддерживается» (листинг 1.130).

Листинг 1.130. Ответ сервера

```
{
"odata.error": {
"code": "0",
"message": {
"lang": "ru",
"value": "The $expan parameter is not supported"
}
}
}
```

Если при указании адреса ресурса вы ошиблись, например, в имени функции (листинг 1.131).

Листинг 1.131. URL запроса

```
localhost/REST/odata/standard.odata/InformationRegister_ЦеныТоваров/Slice_Last()
```

В ответе сервера будет возвращен код ошибки – 6 и описание ошибки – «Метод не найден» (листинг 1.132).

Листинг 1.132. Ответ сервера

```
{
"odata.error": {
"code": "6",
"message": {
"lang": "ru",
"value": "Метод не найден"
}
}
}
```

ПОДРОБНЕЕ

Типичные ошибки, возникающие при получении данных через REST-интерфейс, более подробно описаны в документации «1С:Предприятия» в разделе «Глава 17. Механизмы интернет-сервисов – HTTP-сервисы – Стандартный интерфейс ODATA – Ошибочные ситуации».

Web-сервисы

Механизм Web-сервисов позволяет использовать систему «1С:Предприятие» как набор сервисов в сложных распределенных и гетерогенных системах, а также позволяет интегрировать ее с другими информационными системами с использованием сервис-ориентированной архитектуры (SOA).

Сервис-ориентированная архитектура предлагает новый подход к созданию распределенных информационных систем, в которых программные ресурсы рассматриваются как сервисы, предоставляемые по сети.

Общая информация

Прикладное решение «1С:Предприятия» может являться как поставщиком веб-сервисов, так и потребителем веб-сервисов, опубликованных другими поставщиками (рис. 1.31).

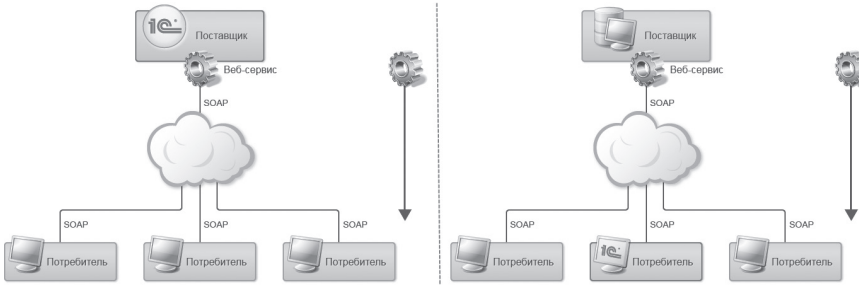


Рис. 1.31. Поставщики и потребители веб-сервисов

Если прикладное решение является поставщиком веб-сервиса, то через Web-сервисы оно может экспортировать свою функциональность вовне. Web-сервисы определяются в дереве объектов конфигурации и становятся доступны произвольным информационным системам после их публикации на веб-сервере.

В этом случае и в файловом, и в клиент-серверном варианте работы взаимодействие между прикладным решением и потребителями веб-сервиса осуществляется через веб-сервер с помощью модуля расширения веб-сервера (рис. 1.32).

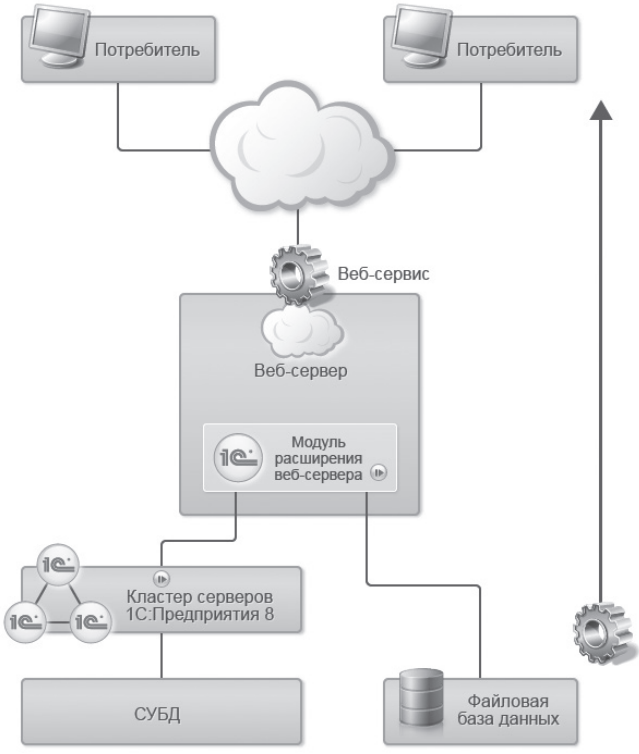


Рис. 1.32. «1С:Предприятие» – поставщик веб-сервиса

При этом, когда потребитель обращается к Web-сервису прикладного решения, выполняется модуль Web-сервиса. Этот модуль содержится в конфигурации, и в нем располагаются процедуры, выполняемые при вызове тех или иных операций Web-сервиса.

В случае клиент-серверного варианта работы этот модуль будет исполняться в кластере. В случае файлового варианта работы – в модуле расширения веб-сервера.

Если прикладное решение является потребителем веб-сервиса стороннего поставщика, то оно может обращаться к Web-сервисам сторонних производителей как через статические ссылки на Web-сервисы, определенные в дереве объектов конфигурации, так и с помощью динамических ссылок на Web-сервисы, создаваемых с помощью встроенного языка.

В этом случае взаимодействие между прикладным решением и поставщиком веб-сервиса осуществляет клиентское приложение. Оно вызывает те или иные операции веб-сервиса и обрабатывает полученные данные (рис. 1.33).

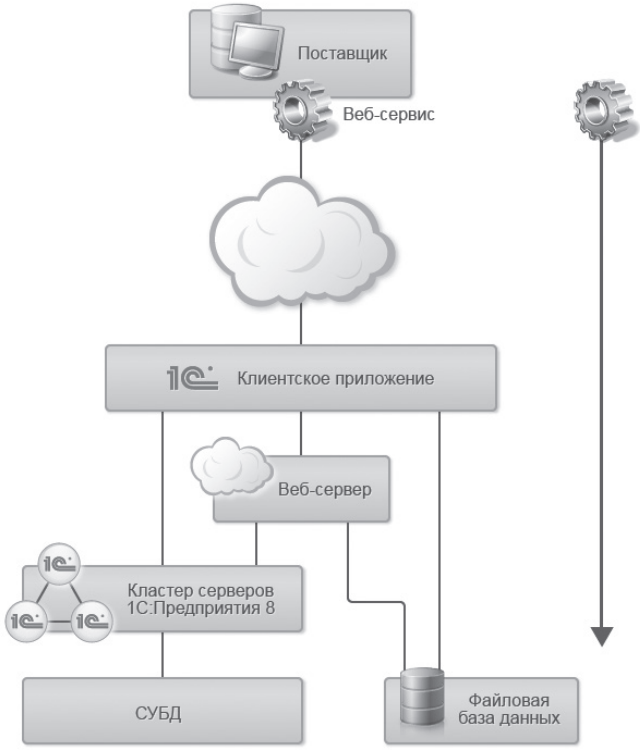


Рис. 1.33. «1С:Предприятие» – потребитель веб-сервиса

Для получения доступа к Web-сервису необходимо использовать адрес, который формируется следующим образом: `http://host/base/ws/<ИмяWebСервиса>` или `http://host/base/ws/<АдресWebСервиса>`. Рассмотрим составные части адреса:

- `http://host/base` – обычный URL, по которому выполняется доступ, например, к информационной базе с помощью веб-клиента. Необходимо помнить, что при использовании информационной базы, в которой настроено разделение данных, значения разделителей можно указывать только в URL информационной базы. Указание разделителей с помощью параметра Z не поддерживается;

- ws – признак того, что выполняется обращение к Web-сервису;
- <ИмяWebСервиса> – имя Web-сервиса. Задается в свойстве объекта Web-сервис;
- <АдресWebСервиса> – описывает альтернативное имя для доступа к Web-сервису. Задается в свойстве Имя файла публикации объекта Web-сервис. Может быть изменено при публикации Web-сервиса.

Обращения по имени и адресу Web-сервиса являются равносильными.

ПОДРОБНЕЕ

Более подробно Web-сервисы описаны в документации «1С:Предприятия» в разделе «Глава 17. Механизмы интернет-сервисов – Web-сервисы».

Предоставление функциональности через Web-сервисы

Чтобы функциональность системы «1С:Предприятие» стала доступной внешним потребителям Web-сервисов, нужно создать в конфигурации необходимые Web-сервисы и затем опубликовать эти Web-сервисы на веб-сервере с помощью специального инструмента конфигуратора (Администрирование > Публикация на веб-сервере…). Процесс публикации подробно описан в книге «1С:Предприятие 8.3. Руководство администратора», в главе 8 «Настройка веб-серверов для работы с "1С:Предприятием"».

Разработка Web-сервиса

Для создания Web-сервиса нужно:

- Добавить в дерево конфигурации объект WebСервис и описать основные свойства этого объекта:
  - URI пространства имен. Каждый Web-сервис может быть однозначно идентифицирован по своему имени и URI пространства имен, которому он принадлежит;
  - Пакеты XDTO. Содержит перечень пакетов XDTO, типы которых могут использоваться для описания типов параметров и возвращаемых значений Web-сервиса (см. раздел «Использование XDTO для описания типов параметров и возвращаемых значений Web-сервисов»);
  - Имя файла публикации. Содержит имя файла описания Web-сервиса, который опубликован на веб-сервере.

- Описать операции, которые может выполнять данный Web-сервис. Основные свойства операций:
  - `Тип возвращаемого значения`. Содержит тип значения, которое возвращает операция Web-сервиса. Может являться типом значения XDTO или типом объекта XDTO (см. раздел «Использование XDTO для описания типов параметров и возвращаемых значений Web-сервисов»);
  - `Возможно пустое значение`. Показывает, может ли значение, возвращаемое операцией, принимать неопределенное значение;
  - `Имя метода`. Содержит имя процедуры, расположенной в модуле Web-сервиса, которая будет выполняться при вызове данного свойства.
- Описать параметры выполняемых операций. Основные свойства параметров:
  - `Тип значения`. Содержит тип значения параметра операции Web-сервиса. Может являться типом значения XDTO или типом объекта XDTO (см. раздел «Использование XDTO для описания типов параметров и возвращаемых значений Web-сервисов»);
  - `Возможно пустое значение`. Показывает, может ли параметр операции принимать неопределенное значение;
  - `Направление передачи`. Определяет направление передачи данных с помощью параметра. `Входной` параметр используется для передачи данных Web-сервису, `Выходной` – для получения данных от Web-сервиса, `Входной-выходной` может использоваться как для передачи данных, так и для их получения от Web-сервиса.
- Создать модуль Web-сервиса и разработать в нем операции, определяющие его функциональность.

Объект конфигурации `WebСервис` содержит модуль, в котором создаются процедуры на встроенном языке, выполняемые при вызове тех или иных операций Web-сервиса. Типы параметров и возвращаемых значений этих операций описываются с помощью типов XDTO и могут представлять собой либо значения XDTO, либо объекты XDTO.

Механизм XDTO будет подробно рассмотрен в разделе «XDTO-сериализация», но сейчас мы расскажем, как типы данных XDTO используются в технологии Web-сервисов «1С:Предприятия».

Использование XDTO для описания типов параметров и возвращаемых значений Web-сервисов

Механизм XDTO позволяет определять объекты переноса данных, которые могут образовывать строгую иерархию и сериализоваться в/из XML. Эти свойства позволили использовать объекты XDTO в качестве параметров и возвращаемых значений операций Web-сервисов.

Основным понятием, на котором строится механизм XDTO, является фабрика XDTO. Фабрика XDTO содержит описание всех типов, с которыми оперирует система. В частности, при создании новой информационной базы «1С:Предприятия» автоматически создается глобальная фабрика XDTO, которая описывает все типы, используемые в конфигурации. Эта фабрика доступна через свойство глобального контекста `ФабрикаXDTO`.

Все типы данных XDTO подразделяются на типы-значения и типы-объекты. Типы-значения позволяют определять простые типы, например: строки, числа, даты, булевы значения и т. д. Типы-объекты позволяют определять сложные типы, такие как структуры и массивы.

Инфраструктура XDTO определяет набор предопределенных типов значений XDTO. Имена типов (`string`, `int`, `date` и т. п.) совпадают с именами типов XML Schema и принадлежат URI пространства имен – `http://www.w3.org/2001/XMLSchema`. Предопределенные типы являются автоматически зарегистрированными в любой фабрике XDTO.

Перед тем как использовать в Web-сервисе собственную структуру данных, необходимо создать пакет XDTO, описывающий тип-объект структуры. Для этого используются средства визуального конструирования, позволяющие добавлять пакеты XDTO в ветку дерева объектов конфигурации Общие > XDTO-пакеты.

Пакет XDTO содержит описание некоторого множества типов, принадлежащих одному пространству имен – пространству имен пакета.

Структуры моделируются типами-объектами. Тип-объект может содержать свойства, которые соответствуют элементам структуры. Каждое свойство характеризуется уникальным именем и типом. Тип свойства может быть как типом-значением, так и типом-объектом.

Новый объект XDTO может быть создан с помощью метода `Создать()` фабрики XDTO на основе типа объекта XDTO. После этого следует присвоить соответствующие значения свойствам объекта XDTO. Ниже приведен пример создания объекта XDTO с именем Номенклатура и заполнения его свойств: Наименование, ПолноеНаименование, ЗакупочнаяЦена и ШтрихКод (листинг 1.133).

Листинг 1.133. Пример создания объекта структурного типа

```
// Создать "пустой" объект XDTO.
СтруктурныйТип = ФабрикаXDTO.Тип("http://www.1c.ru/demos/products", "Номенклатура");
Номенклатура = ФабрикаXDTO.Создать(СтруктурныйТип);

// Заполнить значения свойств объекта XDTO.
ОбъектСправочника = ССылкаНаЭлементСправочника.ПолучитьОбъект();

Номенклатура.Наименование = ОбъектСправочника.Наименование;
Номенклатура.ПолноеНаименование = ОбъектСправочника.ПолноеНаименование;
Номенклатура.ЗакупочнаяЦена = ОбъектСправочника.ЗакупочнаяЦена;
Номенклатура.ШтрихКод = ОбъектСправочника.ШтрихКод;
```

Массивы моделируются свойствами типов-объектов. Тип массива нельзя создать напрямую, но для определенного свойства типа-объекта можно указать минимальное и максимальное количество элементов массива. Если оба значения равны 1, то это единичное свойство; если максимальное количество больше 1, то множественное свойство; если же максимальное количество равно -1, то количество элементов массива не ограничено.

Например, для создания массива элементов номенклатуры, определенного в свойстве Элементы структуры НоменклатураГруппа, нужно выполнить следующий фрагмент кода (листинг 1.134).

Листинг 1.134. Пример создания массива объектов структурного типа

```
СтруктурныйТип = ФабрикаXDTO.Тип("http://www.1c.ru/demos/products", "НоменклатураГруппа");
НоменклатураГруппа = ФабрикаXDTO.Создать(СтруктурныйТип);
НоменклатураГруппа.Элементы.Добавить(Номенклатура);
```

Работа с Web-сервисами сторонних поставщиков

Система «1С:Предприятие» может использовать Web-сервисы, предоставляемые другими поставщиками, следующими способами:

- с помощью статических ссылок, создаваемых в дереве объектов конфигурации;
- с помощью динамических ссылок, создаваемых средствами встроенного языка;
- комбинацией предыдущих способов.

При использовании статической ссылки система «1С:Предприятие» получает описание Web-сервиса поставщика только один раз, при создании ссылки. За счет этого достигается большая скорость работы.

При использовании динамической ссылки описание Web-сервиса получается каждый раз при вызове Web-сервиса. Скорость работы при этом уменьшается, но зато такой подход обеспечивает актуальность описания Web-сервиса поставщика. В случае же использования статических ссылок для получения актуального описания Web-сервиса требуется выполнить повторный импорт WSDL-описания средствами конфигуратора и затем сохранить измененную конфигурацию.

Например, для Web-сервиса с URI пространства имен `http://www.MyCompany.ru/shipment`, именем ДанныеРасходнойНакладной и именем файла публикации `Shipment.1cws` статическая ссылка создается путем импорта WSDL-описания опубликованного сервиса на основе URL – `http://www.MyCompany.ru/shipment/ws/Shipment.1cws?wsdl`. А создание WS-прокси на основании WS-ссылки будет выглядеть следующим образом (листинг 1.135).

Листинг 1.135. Использование статической WS-ссылки

```
// Создать WS-прокси на основании WS-ссылки.
Прокси = WSСсылки.ДанныеРасходнойНакладной.СоздатьWSПрокси(
        "http://www.MyCompany.ru/shipment", "ДанныеРасходнойНакладной", "ДанныеРасходнойНакладнойSoap");
```

При использовании динамической ссылки на этот Web-сервис WS-прокси создается на основании WS-определения (листинг 1.136).

Листинг 1.136. Использование динамической WS-ссылки

```
// Создать WS-прокси на основании WS-определения.
Определение = Новый WSОпределения("http://www.MyCompany.ru/shipment/ws/Shipment.1cws?wsdl");
Прокси = Новый WSПрокси(Определение, "http://www.MyCompany.ru/shipment"
                                    , "ДанныеРасходнойНакладной", "ДанныеРасходнойНакладнойSoap");
```

На практике бывают ситуации, когда один и тот же Web-сервис предоставляется по разным адресам (URL), однако имеет абсолютно одинаковое описание (WSDL). Например, когда используется тиражируемый сервис, выполняющий некоторую функцию. При этом адрес сервиса может быть различным. В этом случае можно использовать комбинированный способ. То есть сначала загрузить в конфигурацию описание Web-сервиса (добавить в дерево объектов конфигурации статическую ссылку на Web-сервис), а затем во время использования указать конкретный адрес, по которому расположен Web-сервис.

Например, если адрес реального расположения сервиса отличается от адреса, который использовался во время загрузки описания Web-сервиса в конфигурацию, то можно предусмотреть ввод адреса конкретного экземпляра сервиса в настройках прикладного решения, и затем этот новый адрес явно указать при создании объекта `WSПрокси` (листинг 1.137).

Листинг 1.137. Комбинированный способ на основе статической ссылки

```
// Создать WS прокси на основании ссылки.
Прокси = WSСсылки.ДанныеРасходнойНакладной.СоздатьWSПрокси("http://www.MyCompany.ru/shipment",
        "ДанныеРасходнойНакладной", "ДанныеРасходнойНакладнойSoap", , , , "http://www.realURL/realPath");
```

Или же можно использовать динамическую ссылку, но адрес расположения Web-сервиса получать не из файла описания (WSDL), а непосредственно указывать при создании объекта (листинг 1.138).

Листинг 1.138. Комбинированный способ на основе динамической ссылки

```
// Создать WS-прокси на основании WS-определения.
Определение = Новый WSОпределения("http://www.MyCompany.ru/shipment/ws/Shipment.1cws?wsdl");
Прокси = Новый WSПрокси(Определение, "http://www.MyCompany.ru/shipment", "ДанныеРасходнойНакладной",
                                    "ДанныеРасходнойНакладнойSoap", , , , "http://www.realURL/realPath");
```

При попытке загрузить описание Web-сервиса в конфигураторе (создание статической ссылки) или при использовании динамической ссылки (при помощи объекта `WSОпределения`) система выполняет проверку загружаемого описания Web-сервиса (WSDL). Если в описании Web-сервиса присутствует ошибка (с точки зрения системы «1С:Предприятие»), то описание не будет загружено и будет сгенерировано исключение. В тексте исключения будет находиться подробная диагностика причин отказа в загрузке.

ПОДРОБНЕЕ

Ошибки, которые могут возникнуть при проверке описания Web-сервиса, более подробно описаны в документации «1С:Предприятия» в разделе «Глава 17. Механизмы интернет-сервисов – Web-сервисы – Работа с веб-сервисами сторонних поставщиков – Общая информация».

Пример реализации Web-сервиса

В этом разделе мы разработаем Web-сервис, который будет получать незакрытые заказы товаров и возвращать пользователю сервиса всю информацию о таких заказах.

Во время разработки мы покажем процесс создания Web-сервиса, определения его основных свойств и работу с пакетами передачи данных XDTO.

Для реализации этой функциональности мы добавим для Web-сервиса соответствующую операцию и создадим функцию-обработчик этой операции в модуле Web-сервиса.

Затем мы покажем процесс использования функциональности, предоставляемой этим Web-сервисом. Для простоты реализуем это в демонстрационной обработке, в той же самой конфигурации, где и создавался Web-сервис, хотя, конечно же, в действительности это будет происходить извне прикладного решения.

Для доступа к Web-сервису через WS-прокси будет использоваться статическая ссылка. С помощью WS-прокси мы выполним операцию, реализованную для Web-сервиса. После этого мы покажем возможную обработку данных, полученных от Web-сервиса. А именно – создадим новые расходные накладные на основании полученных заказов.

ПОДРОБНЕЕ

Подробнее познакомиться с реализацией и использованием Web-сервиса можно в демонстрационной конфигурации «Примеры работы», которая прилагается к книге.

Закрытие заказов на товары

В данном примере мы рассмотрим ситуацию, когда в одном прикладном решении существуют заказы товаров, а в другом прикладном решении на основании этих заказов, не дожидаясь сеанса обмена данными, должны быть созданы расходные накладные. При этом нормативно-справочная информация у этих решений общая.

Для этого в прикладном решении поставщика Web-сервиса нам нужно разработать Web-сервис, который будет получать незакрытые заказы товаров и возвращать всю информацию о таких заказах. И затем опубликовать этот сервис на веб-сервере. А в прикладном решении потребителя Web-сервиса нам нужно реализовать функциональность (например, команду), при выполнении которой на основании полученных от Web-сервиса заказов будут сформированы соответствующие документы продаж.

Итак, приступим.

Сначала для описания типов параметров и возвращаемых значений Web-сервиса в ветке Общие > XDTO-пакеты добавим пакет XDTO ДанныеПередачи с пространством имен `http://localhost/REST`. В свойстве URI пространства имен содержится `http://localhost` – адрес веб-сервера, установленного на локальном компьютере, `/REST` – каталог, в который будет опубликован Web-сервис.

Пакет XDTO будет содержать следующие типы объектов XDTO (рис. 1.34):

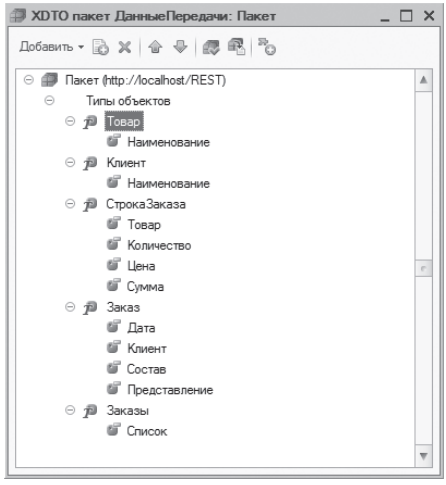


Рис. 1.34. Структура пакета XDTO «ДанныеПередачи»

- Товар – для передачи данных элемента справочника Товары. Этот тип объектов XDTO будет содержать свойство:
  - Наименование – тип `string` из пространства имен `http://www.w3.org/2001/XMLSchema`.
- Клиент – для передачи данных элемента справочника Клиенты. Этот тип объектов XDTO будет содержать свойство:
  - Наименование – тип `string` из пространства имен `http://www.w3.org/2001/XMLSchema`.
- СтрокаЗаказа – для передачи данных одной строки состава заказа товаров. Этот тип объектов XDTO будет содержать следующие свойства:
  - Товар – тип `Товар` из пространства имен `http://localhost/REST`. Представляет собой ссылку на объект XDTO, определенный нами ранее, в предыдущем примере;
  - Количество – тип `int` из пространства имен `http://www.w3.org/2001/XMLSchema`;
  - Цена – тип `int` из пространства имен `http://www.w3.org/2001/XMLSchema`;
  - Сумма – тип `int` из пространства имен `http://www.w3.org/2001/XMLSchema`.

- Заказ – для передачи всех данных одного заказа. Этот тип объектов XDTO будет содержать следующие свойства:
  - Дата – тип `date` из пространства имен `http://www.w3.org/2001/XMLSchema`;
  - Клиент – тип `Клиент` из пространства имен `http://localhost/REST`. Представляет собой ссылку на объект XDTO, определенный нами выше;
  - Состав – тип `СтрокаЗаказа` из пространства имен `http://localhost/REST`. Представляет собой ссылку на объект XDTO, определенный нами выше. Для того чтобы это свойство могло содержать неограниченное множество значений, установим его свойство Максимальное количество в значение -1 (рис. 1.35);

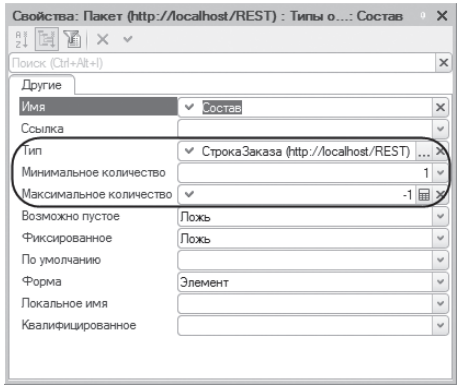


Рис. 1.35. Свойства свойства «Состав» типа объекта XDTO «Заказ»

  - Представление – тип `string` из пространства имен `http://www.w3.org/2001/XMLSchema`.
- Заказы – для передачи данных всех заказов. Этот тип объектов XDTO будет содержать единственное свойство:
  - Список – тип `Заказ` из пространства имен `http://localhost/REST`. Представляет собой ссылку на объект XDTO, определенный нами выше. Для того чтобы это свойство могло содержать неограниченное множество значений, установим его свойство Максимальное количество в значение -1.

Теперь, когда необходимые типы объектов XDTO созданы, в ветке Общие > Web-сервисы добавим Web-сервис с именем Заказы со следующими свойствами (рис. 1.36):

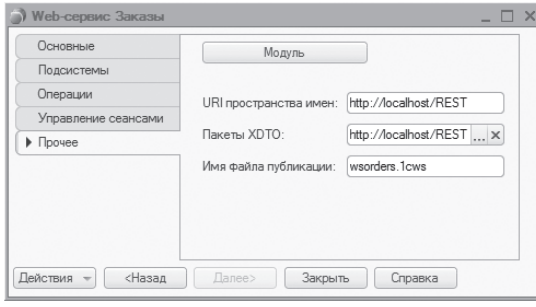


Рис. 1.36. Свойства Web-сервиса «Заказы»

- URI пространства имен – `http://localhost/REST`;
- Пакеты XDTO – `http://localhost/REST`;
- Имя файла публикации – `wsorders.1cws`.

На закладке Операции в окне свойств Web-сервиса определим операцию ПолучитьНезакрытыеЗаказы со следующими свойствами (рис. 1.37):

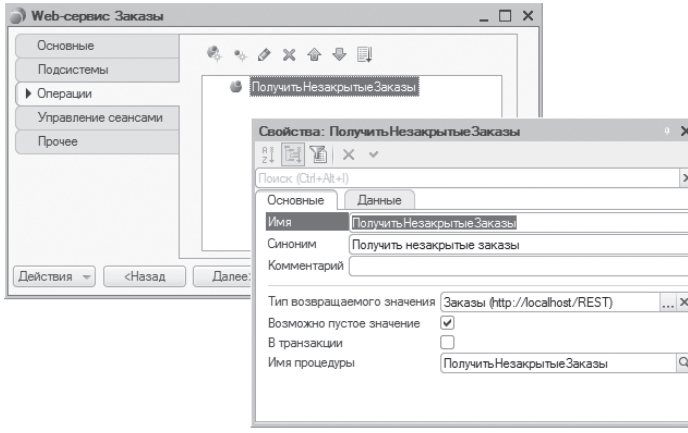


Рис. 1.37. Свойства операции «ПолучитьНезакрытыеЗаказы»

- Тип возвращаемого значения – `Заказы` из пространства имен `http://localhost/REST`;
- Возможно пустое значение – установлен;
- Имя процедуры – `ПолучитьНезакрытыеЗаказы`.

Эта операция будет возвращать список всех незакрытых заказов на товары, т. е. тех заказов, у которых признак закрытия еще не установлен и, следовательно, на их основании еще не создавались расходные накладные.

Нажмем на кнопку открытия со значком лупы у свойства Имя процедуры. В результате в модуле Web-сервиса будет создан шаблон функции-обработчика `ПолучитьНезакрытыеЗаказы()`, которая будет выполняться при его вызове. Заполним эту функцию следующим образом (листинг 1.139).

Листинг 1.139. Функция «ПолучитьНезакрытыеЗаказы()»

```
Функция ПолучитьНезакрытыеЗаказы()

	// Сформировать запрос к списку незакрытых заказов.
	Запрос = Новый Запрос;
	Запрос.Текст = "ВЫБРАТЬ
		|	Заказ.Ссылка КАК Ссылка,
		|	Заказ.Дата КАК Дата,
		|	Заказ.Клиент КАК Клиент,
		|	Заказ.Товары.(
		|		Товар КАК Товар,
		|		Количество КАК Количество,
		|		Цена КАК Цена,
		|		Сумма КАК Сумма
		|	) КАК Товары,
		|	Заказ.Представление КАК ПредставлениеЗаказа
		|ИЗ
		|	Документ.Заказ КАК Заказ
		|ГДЕ
		|	Заказ.Закрыт = ЛОЖЬ
		|	И Заказ.Проведен = ИСТИНА";

	РезультатЗапроса = Запрос.Выполнить();
	Если РезультатЗапроса.Пустой() Тогда
		Возврат Неопределено;
	КонецЕсли;

	Выборка = РезультатЗапроса.Выбрать();

	// Получить типы объектов XDTO.
	ТоварТип = ФабрикаXDTO.Тип("http://localhost/REST", "Товар");
	КлиентТип = ФабрикаXDTO.Тип("http://localhost/REST", "Клиент");
	СтрокаЗаказаТип = ФабрикаXDTO.Тип("http://localhost/REST", "СтрокаЗаказа");
	ЗаказТип = ФабрикаXDTO.Тип("http://localhost/REST", "Заказ");
	ЗаказыТип = ФабрикаXDTO.Тип("http://localhost/REST", "Заказы");

	// Создать объект XDTO список заказов.
	Заказы = ФабрикаXDTO.Создать(ЗаказыТип);

	// В цикле обхода выборки получить список и содержимое незакрытых заказов.
	Пока Выборка.Следующий() Цикл
		// Создать объект XDTO Заказ и Клиент.
		Заказ = ФабрикаXDTO.Создать(ЗаказТип);
```

```
            Клиент = ФабрикаXDTO.Создать(КлиентТип);
            // Заполнить свойства клиента и заказа.
            Клиент.Наименование = Выборка.Клиент.Наименование;
            Заказ.Клиент = Клиент;
            Заказ.Дата = Выборка.Дата;
            Заказ.Представление = Выборка.ПредставлениеЗаказа;

            ВыборкаСостав = Выборка.Товары.Выбрать();
            // В цикле обхода состава выборки получить состав заказа
            Пока ВыборкаСостав.Следующий() Цикл
            // Создать объекты XDTO строки заказа и товара.
                  СтрокаЗаказа = ФабрикаXDTO.Создать(СтрокаЗаказаТип);
                  Товар = ФабрикаXDTO.Создать(ТоварТип);
                  // Заполнить свойства товара и строки заказа.
                  Товар.Наименование = ВыборкаСостав.Товар.Наименование;
                  СтрокаЗаказа.Товар = Товар;
                  СтрокаЗаказа.Цена = ВыборкаСостав.Цена;
                  СтрокаЗаказа.Количество  = ВыборкаСостав.Количество;
                  СтрокаЗаказа.Сумма = ВыборкаСостав.Сумма;
                  // Добавить строку заказа.
                  Заказ.Состав.Добавить(СтрокаЗаказа);
            КонецЦикла;

            // Добавить заказ в список заказов.
            Заказы.Список.Добавить(Заказ);

            // Закрыть заказ.
            Документ = Выборка.Ссылка.ПолучитьОбъект();
            Документ.Закрыт = Истина;
            Документ.Записать();

      КонецЦикла;

      Возврат Заказы;

КонецФункции
```

Прокомментируем код функции.

Сначала мы выполняем запрос, получающий информацию обо всех незакрытых заказах.

После этого с помощью глобальной фабрики XDTO получаем типы объектов XDTO, ранее описанные нами в пакете XDTO ДанныеПередачи: `ТоварТип`, `КлиентТип`, `СтрокаЗаказаТип`, `ЗаказТип` и `ЗаказыТип`. На основе последнего типа мы создаем объект XDTO `Заказы`.

В цикле обхода выборки из результата запроса мы создаем объект XDTO `Клиент`, заполняем его наименование и объект XDTO `Заказ` и заполняем его свойства: `Клиент`, `Представление` и `Дата`.

Затем мы обходим табличную часть заказа Товары. В цикле обхода выборки состава заказа мы создаем объект XDTO `Товар`, заполняем

его наименование и объект XDTO `СтрокаЗаказа` и заполняем его свойства: `Товар`, `Количество`, `Цена` и `Сумма`. После этого объект `СтрокаЗаказа` добавляем в список значений XDTO `Заказ` – `Заказ.Состав.Добавить(СтрокаЗаказа)`.

А при завершении цикла обхода родительской выборки объект `Заказ` добавляем в список значений XDTO `Заказы` – `Заказы.Список.Добавить(Заказ)`.

После этого от ссылки на заказ получается объект, признак заказа Закрыт устанавливается в `Истина`, заказ записывается. И, таким образом, заказ закрывается – следовательно, при следующем вызове Web-сервиса на основании этого заказа уже не будет создаваться расходная накладная.

В заключение функция возвращает объект XDTO `Заказы`. Тип этого объекта был описан в свойстве Тип возвращаемого значения операции ПолучитьНезакрытыеЗаказы созданного нами Web-сервиса Заказы.

Теперь осталось только опубликовать созданный Web-сервис на веб-сервере – например, расположенном на локальном компьютере `http://localhost` в каталоге `/REST`. Для этого в диалоге публикации информационной базы на веб-сервере на закладке Web-сервисы установим флажок Публиковать Web-сервисы по умолчанию и флажок Публиковать Web-сервисы и отметим имеющиеся Web-сервисы, к которым мы хотим предоставить доступ из внешних систем (рис. 1.38).

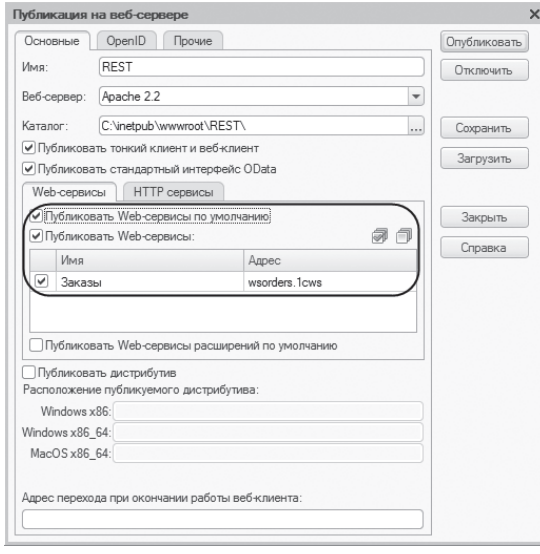


Рис. 1.38. Диалог публикации на веб-сервере

Теперь посмотрим, как используется функциональность, предоставляемая Web-сервисом Заказы, в прикладном решении «1С:Предприятия».

Например, при нажатии кнопки Закрыть заказы в форме демонстрационной обработки вызывается серверная процедура `ЗакрытьЗаказыНаСервере()` (листинг 1.140).

Листинг 1.140. Функция «ЗакрытьЗаказыНаСервере()»

```
&НаСервереБезКонтекста
Процедура ЗакрытьЗаказыНаСервере()

	// Создать WS-прокси на основании WS-ссылки.
	Прокси = WSCсылки.ДанныеЗаказов.СоздатьWSПрокси("http://localhost/REST", "Заказы", "ЗаказыSoap");

	// Выполнить операцию Веб-сервиса Заказы.
	СписокЗаказов = Прокси.ПолучитьНезакрытыеЗаказы();
	Если СписокЗаказов = Неопределено Тогда
		Возврат;
	КонецЕсли;

	Для Каждого Заказ Из СписокЗаказов.Список Цикл
		// Создать расходную накладную на основании заказа.
		ДокументОбъект = Документы.РасходнаяНакладная.СоздатьДокумент();
		ДокументОбъект.Дата = Заказ.Дата;
		// Найти клиента по наименованию.
		ДокументОбъект.Клиент = Справочники.Клиенты.НайтиПоНаименованию(
					Заказ.Клиент.Наименование);
		ДокументОбъект.Основание = Заказ.Представление;

		// Заполнить ТЧ расходной накладной данными заказа.
		Для Каждого СтрокаСостава Из Заказ.Состав Цикл
			НоваяСтрока = ДокументОбъект.Товары.Добавить();
			НоваяСтрока.Количество = СтрокаСостава.Количество;
			НоваяСтрока.Цена = СтрокаСостава.Цена;
			НоваяСтрока.Сумма = СтрокаСостава.Сумма;

			// Найти товар по наименованию.
			НоваяСтрока.Товар = Справочники.Товары.НайтиПоНаименованию(
						СтрокаСостава.Товар.Наименование);
		КонецЦикла;

		ДокументОбъект.Записать(, РежимПроведенияДокумента.Неоперативный);
	КонецЦикла;

КонецПроцедуры
```

В данной процедуре используется статическая ссылка на Web-сервис. Для этого в дерево объектов конфигурации в ветку Общие > WS-ссылки мы добавили объект `WSCсылка` с именем ДанныеЗаказов, ссылающийся на опубликованный Web-сервис. При создании ссылки мы выполнили импорт WSDL-описания Web-сервиса, указав в качестве URL `http://localhost/REST/ws/wsorders.1cws?wsdl` (рис. 1.39).

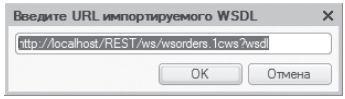


Рис. 1.39. Создание статической WS-ссылки

На основе статической ссылки на Web-сервис создается объект `WSПрокси`, с помощью которого выполняется операция Web-сервиса `ПолучитьНезакрытыеЗаказы()`.

В результате список незакрытых заказов возвращается Web-сервисом в виде списка XDTO в переменную `СписокЗаказов`.

Далее в цикле обхода этого списка (`СписокЗаказов.Список`) на основании каждого заказа создается расходная накладная, и ее реквизиты: Дата, Клиент, Основание, а также табличная часть Товары заполняются данными заказа (рис. 1.40, 1.41).

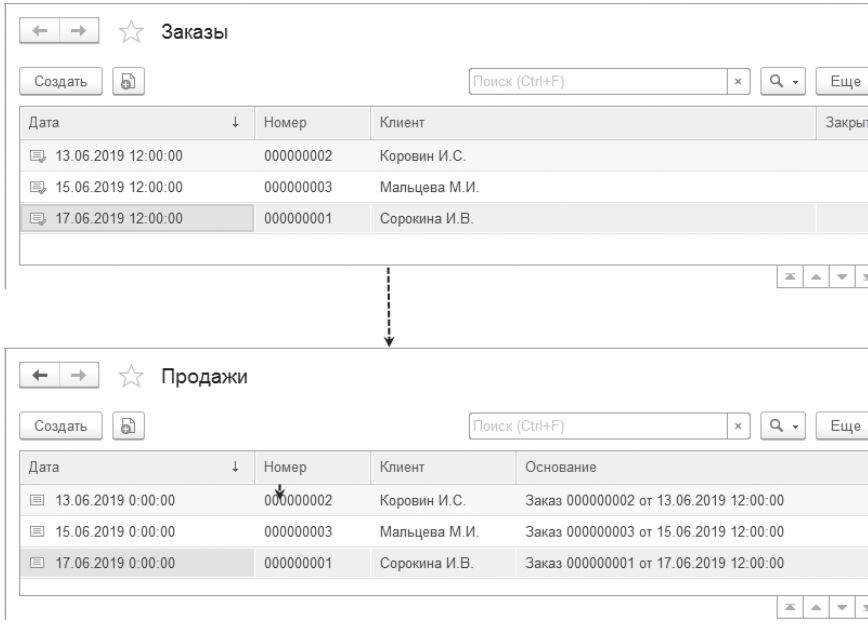


Рис. 1.40. Заказы товаров и соответствующие им расходные накладные

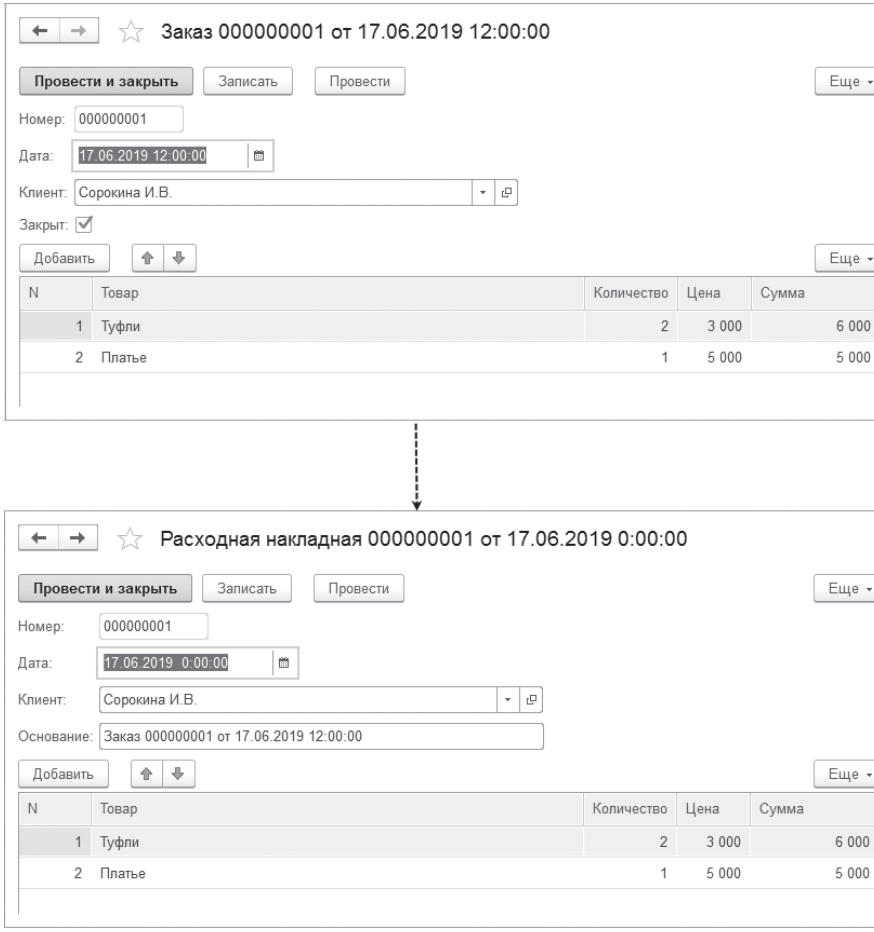


Рис. 1.41. Заказы товаров и соответствующие им расходные накладные

Таким образом, в прикладном решении потребителя Web-сервиса появятся расходные накладные, соответствующие заказам товаров, которые существуют у поставщика этого Web-сервиса.

Web-сервисы в расширениях

В расширениях также можно создавать собственные Web-сервисы. Таким образом в процессе внедрения, не изменяя конфигурацию, находящуюся на поддержке, можно настроить взаимодействие прикладного решения с внешними системами. Например, для синхронизации данных между прикладным решением и веб-сайтом.

Для этого в расширении в ветку Общие > Web-сервисы нужно добавить Web-сервис, разработать его по аналогии с описанными выше примерами и опубликовать на веб-сервере, установив в диалоге публикации флажок Публиковать Web-сервисы расширений по умолчанию.

Как правило, Web-сервисы не существуют сами по себе, а ссылаются на XDTO-пакеты. Поэтому в расширении можно создать не только собственные Web-сервисы, но и собственные XDTO-пакеты. А также XDTO-пакеты (или их отдельные элементы) можно заимствовать из основной конфигурации.

Кроме того, в расширении можно создавать и собственные WS-ссылки, с помощью которых статически описывать сторонние веб-сервисы, к которым обращается конфигурация.

В отличие от Web-сервисов, созданных в основной конфигурации, установка в диалоге публикации флажка Публиковать Web-сервисы расширений по умолчанию не позволяет выбирать для публикации конкретные сервисы расширений. Поэтому, если требуется опубликовать лишь некоторые из них, можно вручную отредактировать файл публикации (default.vrd) и удалить там «ненужные» сервисы.

Повторное использование сеансов интернет-сервисов

В данном разделе мы рассмотрим, как можно повысить производительность веб-сервисов при помощи повторного использования сеансов этих сервисов. В данном случае под веб-сервисами понимаются любые интернет-сервисы: Web-сервис, HTTP-сервис, а также стандартный интерфейс Odata.

Дело в том, что если не переиспользовать сеансы веб-сервисов, то при каждом вызове сервиса система будет создавать новый сеанс работы с информационной базой. При этом выполняется довольно «тяжелый» обработчик `УстановкаПараметровСеанса()` и др. При завершении вызова интернет-сервиса сеанс завершается, а при повторном вызове система опять тратит время и ресурсы на создание сеанса.

Кроме этого существует и функциональный недостаток. Веб-сервисы не обладают состоянием. Однако иногда требуется реализовывать логику, использующую сохранение состояния между вызовами веб-сервиса.

Для устранения этих недостатков в настоящее время в платформе реализованы две различные стратегии, обеспечивающие переиспользование сеансов:

- автоматическое переиспользование сеансов из пула;
- управление сеансами с помощью HTTP-заголовков.

Необходимость использования той или иной стратегии можно определить в свойствах объектов конфигурации Web-сервис и HTTP-сервис, а также, при необходимости, переопределить в файле публикации default.vrd.

Свойство объектов конфигурации Web-сервис и HTTP-сервис `Повторное использование сеансов` может принимать значения:

- Использовать. В этом случае временем жизни и характером повторного использования сеанса управляет непосредственно клиент интернет-сервиса. Для управления созданием нового сеанса используются специальные заголовки HTTP-запроса к интернет-сервису.
- Использовать автоматичесЧки (значение по умолчанию). В этом случае системой организуется пул сеансов. Каждый сеанс в пуле характеризуется некоторым набором параметров. Если сеанс в пуле свободен и характеризуется тем же набором параметров, что и запрос интернет-сервиса, то запросу сразу будет предоставлен существующий сеанс.
- Не использовать. При каждом обращении к интернет-сервису будет создаваться новый сеанс.

Свойство `Время жизни сеанса` позволяет указать:

- через какой промежуток времени «свободный» сеанс будет удален из пула сеансов в случае автоматического повторного использования сеансов;
- через какое время бездействия сеанс будет принудительно завершен системой при ручном повторном использовании сеансов.

Если данное свойство установлено в значение 0, то это означает, что сеансы повторно не используются. Такая установка эквивалентна установке свойства `Повторное использование сеансов` в значение Не использовать.

В файле публикации на веб-сервере default.vrd для элементов, описывающих SOAP-сервисы, HTTP-сервисы и сервисы OData, также существуют соответствующие атрибуты:

- `reuseSessions` – аналог свойства `ПовторноеИспользованиеСеансов`, может принимать значения autouse, use и dontuse;
- `sessionMaxAge` – аналог свойства `ВремяЖизниСеанса`;

- `poolTimeout` – используется при автоматическом управлении сеансами, содержит время ожидания доступности сеанса;
- `poolSize` – используется при автоматическом управлении сеансами, содержит максимальное количество сеансов, которое может быть создано в пуле.

Например, файл default.vrd может выглядеть следующим образом (листинг 1.141).

Листинг 1.141. Файл публикации на веб-сервере

```
<?xml version="1.0" encoding="UTF-8"?>
<point xmlns="http://v8.1c.ru/8.2/virtual-resource-system"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        base="/REST"
        ib="File=&amp;quot;C:\Users\dell\Documents\InfoBase14&amp;quot;;">
    <standardOdata enable="true"
                reuseSessions="autouse"
                sessionMaxAge="20"
                poolSize="10"
                poolTimeout="5"/>
    <ws>
        <point name="ЦеныТоваров"
                alias="wsprices.1cws"
                enable="true"
                reuseSessions="autouse"
                sessionMaxAge="20"
                poolSize="10"
                poolTimeout="5"/>
        <point name="Заказы"
                alias="wsorders.1cws"
                enable="true"
                reuseSessions="autouse"
                sessionMaxAge="20"
                poolSize="10"
                poolTimeout="5"/>
    </ws>
    <httpServices>
        <service name="Сотрудники"
                rootUrl="employees"
                enable="true"
                reuseSessions="autouse"
                sessionMaxAge="20"
                poolSize="10"
                poolTimeout="5"/>
        <service name="ЦеныТоваров"
                rootUrl="prices"
                enable="true"
                reuseSessions="autouse"
                sessionMaxAge="20"
                poolSize="10"
                poolTimeout="5"/>
    </httpServices>
</point>
```

Файл default.vrd имеет больший приоритет, чем конфигурация. При публикации конфигурации атрибуты `reuseSessions` и `sessionMaxAge` заполняются из соответствующих свойств объектов конфигурации Web-сервис и HTTP-сервис. Но при необходимости можно изменить эти значения прямо в файле, и тогда платформа будет использовать их, а не значения из конфигурации.

Автоматическое переиспользование сеансов

Стратегия, реализующая автоматическое переиспользование сеансов из пула, подходит для высоконагруженных публичных сервисов, к которым обращаются клиенты, выполняющие шаблонные операции и обладающие унифицированными привилегиями.

Сеансы в пуле хранятся в разрезе типа сервиса, наименования сервиса, пользователя/пароля, значений разделителей и безопасного режима. Причем в пуле может быть несколько сеансов с одинаковыми значениями перечисленных реквизитов.

При обнаружении входящего запроса к интернет-сервису система анализирует текущий пул сеансов и ищет сеанс с параметрами, в точности соответствующими входящему запросу. Если такой сеанс обнаруживается и он не используется запросом от другого клиента интернет-сервиса, входящий запрос будет обслуживаться найденным сеансом. Если свободного сеанса не найдено – выполняется попытка создать новый сеанс. Если при попытке создания нового сеанса будет превышен размер пула сеансов (`poolSize`), входящий запрос будет ожидать некоторое время (`poolTimeout`). Если по истечении таймаута пула свободный сеанс не обнаруживается – интернет-запрос будет завершен с ошибкой 406 «Not Acceptable». Свободный сеанс будет уничтожен системой автоматически после истечения времени жизни свободного сеанса (`ВремяЖизниСеанса`).

Настройки автоматического пула сеансов действуют в рамках публикации. Таким образом, если для некоторой информационной базы созданы несколько публикаций, то при вызове интернет-сервиса используется настройка пула той публикации, через которую выполняется вызов.

При выборе размера пула следует учитывать все разрезы, в которых хранятся сеансы в пуле. Рекомендуется подбирать размер пула сеансов с небольшим запасом, чтобы при максимальной загрузке в пуле оставалось несколько свободных сеансов. Иначе при пиковой нагрузке некоторые вызовы будут завершаться с ошибками.

Однако эта стратегия не подходит для сценариев, в которых нужно использовать сохраненное состояние сеанса на сервере. Потому что нет никакой гарантии, что при следующем вызове клиент будет подключен к тому же самому сеансу, так как в пуле может быть несколько сеансов с одинаковыми значениями ключевых реквизитов.

Ручное управление сеансами

Стратегия ручного управления сеансами подразумевает, что клиент интернет-сервиса самостоятельно управляет количеством сеансов и временем их жизни. Эта стратегия лучше подходит для высокоинтегрированных систем в рамках одной организации. Разработчик может реализовать собственный алгоритм, который будет управлять временем жизни сеансов и их количеством.

В качестве управляющего элемента выступает заголовок HTTP-запроса `IBSession`. Этот заголовок может принимать два значения:

- `start` – в этом случае система «1С:Предприятие» создает новый сеанс, выполняет аутентификацию, устанавливает разделители, выполняются все необходимые обработчики событий. Если система не может создать новый сеанс, то клиент получит ошибку 406 «Not Acceptable». Если создание сеанса выполнено успешно, в HTTP-ответ помещается директива установки cookie `IBsession` с идентификатором созданного сеанса: `Set-Cookie: IBsession=<ID сеанса>`.

  При необходимости использовать ранее созданный сеанс необходимо в HTTP-запросе к системе «1С:Предприятие» указать идентификатор ранее созданного сеанса в заголовке: `Cookie: IBsession=<ID сеанса>`. Если в запросе указывается идентификатор сеанса, который ранее не создавался или был завершен, клиент получает ошибку 400 «Bad Request».

  Если HTTP-запрос не содержит заголовка `IBSession`, то сеанс создается и завершается при каждом вызове интернет-сервиса.

- `finish` – в этом случае система «1С:Предприятие» завершает сеанс, который указан в запросе (`Cookie: IBsession=<ID сеанса>`), одновременно с командой завершения сеанса. Завершение сеанса произойдет автоматически, если в этом сеансе не выполнялось никаких действий за время жизни сеанса.

Пул сеансов в этом случае не используется.

Эта стратегия позволяет реализовать сценарии, в которых используется состояние сеанса, сохраненное на сервере. Потому что в ответе сервера содержится уникальный идентификатор созданного сеанса. Однако нужно

помнить, что завершение сеанса может происходить автоматически, без участия разработчика, когда превышен период бездействия сеанса, поэтому сеансовые данные могут быть сброшены.

Коды состояния в ответах HTTP-сервера

В данном разделе будут рассмотрены различные коды ответов, возвращаемые HTTP-сервером, а точнее – коды состояния, содержащиеся в свойстве `КодСостояния` объектов `HTTPСервисОтвет` и `HTTPОтвет`.

ПОДРОБНЕЕ

> Более подробно коды состояний, возвращаемые сервером, описаны в протоколе по адресу http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

Если при выполнении HTTP-запроса на веб-сервере возникает фатальная ошибка, то в большинстве случаев нужно остановить работу и показать пользователю сообщение об ошибке, включив в него код состояния, возвращенный HTTP-сервером.

Прежде всего, если при попытке выполнить HTTP-запрос на веб-сервере было вызвано исключение, то это говорит о том, что запрос не дошел до HTTP-сервера.

Коды состояний 4XX сигнализируют о неправильном запросе. Причины ошибок могут быть следующими:

- сервер не может понять запрос из-за неправильного синтаксиса;
- сервер не нашел ничего, соответствующего Request-URI;
- метод, указанный в строке запроса, не разрешен для ресурса, идентифицируемого Request-URI;
- запрос требует аутентификации пользователя и др.

Коды состояний 5XX сигнализируют о проблемах на сервере (возможно, на прокси-сервере). Причины ошибок могут быть следующими:

- сервер не поддерживает функциональность, необходимую для выполнения запроса;
- сервер не может обработать запрос из-за временной перегрузки или обслуживания сервера;
- на сервере произошла внутренняя ошибка, которая не позволила ему выполнить запрос;
- сервер не поддерживает или отказывается поддерживать версию протокола HTTP, которая использовалась в сообщении запроса и др.

Коды состояний 3XX означают, что клиент должен предпринять дальнейшие действия для выполнения запроса. При этом запрос должен быть перенаправлен на новый адрес. Причины ошибок могут быть следующими:

- запрошенному ресурсу был назначен новый постоянный URI;
- запрашиваемый ресурс временно находится под другим URI и др.

Коды состояний 1XX и 2XX означают, что запрос клиента был успешно получен, понят и принят.

Ниже приводится пример процедуры `ВыполнитьHTTPЗапрос()`, которая получает в качестве параметра адрес ресурса. В процедуре устанавливается HTTP-соединение с локальным веб-сервером «localhost», затем на основе адреса ресурса создается HTTP-запрос и выполняется на сервере методом `Получить()` объекта `HTTPСоединение`. Ответ сервера в виде объекта `HTTPОтвет` возвращается в переменную `Результат` (листинг 1.142).

Листинг 1.142. Процедура «ВыполнитьHTTPЗапрос»

```
&НаСервереБезКонтекста
Процедура ВыполнитьHTTPЗапрос(АдресРесурса)

	СерверИсточник = "localhost";
	Сообщение = Новый СообщениеПользователю;

	// Создать HTTP-соединение с сервером localhost.
	HTTPСоединение = Новый HTTPСоединение(СерверИсточник);

	// Создать HTTP-запрос на основе URL.
	HTTPЗапрос = Новый HTTPЗапрос(АдресРесурса);

	Попытка
		Результат = HTTPСоединение.Получить(HTTPЗапрос);
	Исключение
		Сообщение.Текст = "Произошла сетевая ошибка!";
		Сообщение.Сообщить();
		ВызватьИсключение;
	КонецПопытки;

	Если Результат.КодСостояния >= 400 и Результат.КодСостояния < 500  Тогда
		Сообщение.Текст = "Код статуса больше 4XX, ошибка запроса. Код статуса:
				" + Результат.КодСостояния;
	КонецЕсли;

	Если Результат.КодСостояния >= 500 и Результат.КодСостояния < 600 Тогда
		Сообщение.Текст = "Код статуса больше 5XX, ошибка сервера. Код статуса:
				" + Результат.КодСостояния;
	КонецЕсли;

	// Обработать перенаправление.
	Если Результат.КодСостояния >= 300 и Результат.КодСостояния < 400 Тогда
		Сообщение.Текст = "Код статуса больше 3XX, Перенаправление. Код статуса:
				" + Результат.КодСостояния;
```

```
            Если Результат.КодСостояния = 302 Тогда
                Сообщение.Текст = Сообщение.Текст + Символы.ПС
                        + "Код статуса 302, Постоянное перенаправление.";
                АдресРесурса = Результат.Заголовки.Получить("Location");

                Если АдресРесурса <> Неопределено Тогда
                    Сообщение.Текст = Сообщение.Текст + Символы.ПС
                        + "Выполняю запрос по новому адресу " +
                        Символы.ПС + АдресРесурса;
                    ВыполнитьHTTPЗапрос(АдресРесурса);
                Иначе
                    Сообщение.Текст = Сообщение.Текст + Символы.ПС
                        + "Сервер не сообщил адрес ресурса!";
                КонецЕсли;
            КонецЕсли;
        КонецЕсли;

        Если Результат.КодСостояния < 300 Тогда
            Сообщение.Текст = "Скорее всего все хорошо!";
            Сообщение.Текст = Сообщение.Текст + Символы.ПС + "Код статуса: " + Результат.КодСостояния;
        КонецЕсли;

        Сообщение.Сообщить();

КонецПроцедуры
```

В этой процедуре обрабатываются и выводятся пользователю различные коды состояния HTTP-сервера. Отдельно анализируются коды ответа, которые сигнализируют о перенаправлении запросов. Если код состояния – 302, то процедура выполняется заново, с новым адресом ресурса.

FTP-соединение

В этом разделе мы рассмотрим, как организовать взаимодействие с FTP-сервером по FTP-протоколу обмена файлами (FTPS, FTPES). Для указания параметров доступа к FTP-серверу используется объект `FTPСоединение`.

При этом разработчику доступны следующие возможности:

- определить активный или пассивный режим работы FTP-соединения;
- установить пользователя, от имени которого установлено соединение, порт сервера, с которым произведено соединение, сервер (прокси), через который установлено соединение;
- записать (найти) файлы объектов типа `FTPФайл`;
- переименовать файлы и каталоги на сервере;

- получить ресурс из указанного адреса;
- создать (удалить) каталог на сервере;
- установить текущий каталог на сервере;
- определить размер файла (в байтах);
- определить параметры файлов и каталогов;
- использовать протокол TLS 1.0 – TLS1.2 для установления защищенного соединения;
- проверить сертификат сервера с использованием стандартных механизмов ОС или файла сертификата.

Ниже мы рассмотрим небольшие примеры, как с помощью методов `Получить()` и `Записать()` объекта `FTPСоединение` получать и записывать файлы с/на FTP-сервер. А также – как получать файлы с сервера с помощью прямого копирования методом глобального контекста `НачатьКопированиеФайла()`.

ПОДРОБНЕЕ

> Подробнее познакомиться с примерами взаимодействия с FTP-сервером можно в демонстрационной конфигурации «Примеры работы», которая прилагается к книге.

В форме нашей демонстрационной обработки для обращения к FTP-ресурсу используются следующие параметры, которые хранятся в соответствующих реквизитах формы:

- Сервер – сервер, с которым осуществляется соединение, например `ftp.example.com`;
- ИмяПользователя – имя пользователя на FTP-сервере;
- ПарольПользователя – пароль пользователя на FTP-сервере;
- КаталогНаСервере – путь на FTP-сервере до нужных файлов. Путь нужно указывать только с прямыми слешами вместо обратных, например: `/TEST/`;
- ЛокальныйКаталог – путь на локальном компьютере, определяющий, куда копировать файлы, например `E:\TEST\`.

Получить файлы с сервера

Предположим, нам нужно получить файлы определенного типа с FTP-сервера через FTP-соединение. Для этого нужно создать FTP-соединение с сервером, используя имя FTP-сервера, а также имя и пароль пользователя на этом сервере. Затем найти нужные файлы по заданной маске (например, «*.tif») методом `НайтиФайлы()` в каталоге на сервере и скопировать найденные файлы (объекты `FTPФайл`) в локальный каталог на компьютере пользователя методом `Получить()` объекта `FTPСоединение`.

Обработчик команды для получения файлов с FTP-сервера может выглядеть следующим образом (листинг 1.143).

Листинг 1.143. Процедура «ПолучитьФайлыССервера»

```
&НаКлиенте
Процедура ПолучитьФайлыССервера(Команда)

	Сообщение = Новый СообщениеПользователю;

	Попытка
		// Установить соединение с FTP-сервером.
		FTPСервер = Новый FTPСоединение(Сервер, , ИмяПользователя, ПарольПользователя);
	Исключение
		// Вывести сообщение об ошибке соединения с сервером.
		Сообщение.Текст = "Не удалось соединиться с сервером: " + Сервер;
		Сообщение.Сообщить();
		Сообщение.Текст = ОписаниеОшибки();
		Сообщение.Сообщить();
		Возврат;
	КонецПопытки;

	// Найти файлы в нужном каталоге по указанной маске.
	МаскаФайлов = "*.tif";
	МассивФайлов = FTPСервер.НайтиФайлы(КаталогНаСервере, МаскаФайлов);

	Для Каждого Файл Из МассивФайлов Цикл
		// Проверить, что это не каталог.
		Если Файл.ЭтоФайл() Тогда

			// Копировать файл в локальный каталог.
			Сообщение.Текст = "Считывается файл - " + Файл.Имя;
			Сообщение.Сообщить();

			FTPСервер.Получить(Файл.ПолноеИмя, ЛокальныйКаталог + Файл.Имя);
		КонецЕсли;
	КонецЦикла;

КонецПроцедуры
```

В первом параметре метода `Получить()` содержится относительный адрес ресурса на FTP-сервере (полное имя FTP-файла), из которого будут прочитаны данные. Во втором – полное имя файла, в который будут помещены данные полученного ресурса.

Записать файлы на сервер

Теперь реализуем обратную операцию. Найдем файлы по заданной маске на локальном компьютере и запишем их в каталог на FTP-сервере. Для этого нужно создать FTP-соединение с сервером, используя имя FTP-сервера, а также имя и пароль пользователя на этом сервере. И затем записать найденные локальные файлы (объекты `Файл`) в каталог на сервере методом `Записать()` объекта `FTPСоединение`.

Обработчик команды для записи файлов на FTP-сервер может выглядеть следующим образом (листинг 1.144).

Листинг 1.144. Процедура «ЗаписатьФайлыНаСервер»

```
&НаКлиенте
Процедура ЗаписатьФайлыНаСервер(Команда)

    // Найти файлы в локальном каталоге по указанной маске.
    МаскаФайлов = "*.tif";
    НачатьПоискФайлов(Новый ОписаниеОповещения("ЗаписатьФайлыНаСерверЗавершение", ЭтотОбъект),
        ЛокальныйКаталог, МаскаФайлов);

КонецПроцедуры
```

Обратите внимание, что поиск файлов на компьютере пользователя мы будем выполнять с помощью немодального метода `НачатьПоискФайлов()` вместо модального `НайтиФайлы()`. Первым параметром в метод `НачатьПоискФайлов()` мы передаем описание оповещения, указывающее на экспортную процедуру `ЗаписатьФайлыНаСерверЗавершение()`, которая будет вызвана по окончании поиска файлов (листинг 1.145). А также мы передаем в эту процедуру локальный каталог, в котором будет производиться поиск, и маску для поиска файлов (например, «*.tif»).

Листинг 1.145. Процедура «ЗаписатьФайлыНаСерверЗавершение»

```
&НаКлиенте
Процедура ЗаписатьФайлыНаСерверЗавершение(МассивФайлов, Дополнительно ) Экспорт

    Сообщение = Новый СообщениеПользователю;

    Попытка
        // Установить соединение с FTP-сервером.
        FTPСервер = Новый FTPСоединение(Сервер, , ИмяПользователя, ПарольПользователя);
    Исключение
        // Вывести сообщение об ошибке соединения с сервером.
        Сообщение.Текст = "Не удалось соединиться с сервером: " + Сервер;
        Сообщение.Сообщить();
        Сообщение.Текст = ОписаниеОшибки();
        Сообщение.Сообщить();
```

```
        Возврат;
    КонецПопытки;

    Для Каждого Файл Из МассивФайлов Цикл
        // Записать файл в каталог на сервере.
        Сообщение.Текст = "Записывается файл - " + Файл.Имя;
        Сообщение.Сообщить();

        FTPСервер.Записать(ЛокальныйКаталог + Файл.Имя, КаталогНаСервере + Файл.Имя);
    КонецЦикла;

КонецПроцедуры
```

В этой процедуре мы создаем FTP-соединение, обходим в цикле массив найденных файлов, содержащихся в параметре `МассивФайлов` (объекты `Файл`) и записываем каждый файл в каталог на сервере методом `Записать()` объекта `FTPСоединение`.

Копировать файлы с сервера

В этом примере мы рассмотрим возможность получения файлов с сервера с помощью прямого копирования методом глобального контекста `НачатьКопированиеФайла()`.

Поскольку доступ к FTP-ресурсам будет осуществляться напрямую, а не через FTP-соединение, то полный путь на сервере должен включать помимо каталога на сервере также и имя самого сервера, и имя и пароль пользователя на сервере вида: `ftp://<ИмяПользователя>:<ПарольПользователя>@<Сервер>/<КаталогНаСервере>`.

Обработчик команды для копирования файлов с FTP-сервера может выглядеть следующим образом (листинг 1.146).

Листинг 1.146. Процедура «КопироватьФайлыССервера»

```
&НаКлиенте
Процедура КопироватьФайлыССервера(Команда)

    ПутьНаСервере = "ftp://" + ИмяПользователя + ":" + ПарольПользователя
                                                + "@" + Сервер + КаталогНаСервере;
    // Найти файлы в нужном каталоге по указанной маске.
    МаскаФайлов = "*.png";
    НачатьПоискФайлов(Новый ОписаниеОповещения(
                                                "КопироватьФайлыССервераЗавершение", ЭтотОбъект),
        ПутьНаСервере, МаскаФайлов);

КонецПроцедуры
```

Как и в предыдущем примере, поиск файлов на FTP-сервере мы будем выполнять с помощью немодального метода `НачатьПоискФайлов()`. В этот метод первым параметром мы передаем описание оповещения, указывающее на экспортную процедуру `КопироватьФайлыССервераЗавершение()`, которая будет вызвана по окончании поиска файлов (листинг 1.147). А также мы передаем в эту процедуру полный путь на FTP-сервере для доступа к файлам и маску для поиска файлов.

Листинг 1.147. Процедура «КопироватьФайлыССервераЗавершение»

```
&НаКлиенте
Процедура КопироватьФайлыССервераЗавершение(МассивФайлов, Дополнительно ) Экспорт

    Сообщение = Новый СообщениеПользователю;

    Для Каждого Файл Из МассивФайлов Цикл
        Сообщение.Текст = "Считывается файл - " + Файл.Имя;
        Сообщение.Сообщить();

        КопироватьФайлыВЛокальныйКаталог(Новый ОписаниеОповещения(
                                    "КопироватьФайлыЛокальноЗавершение", ЭтотОбъект), Файл.Имя);
    КонецЦикла;

КонецПроцедуры
```

В этой процедуре мы обходим в цикле массив найденных файлов, содержащихся в параметре `МассивФайлов`, и для каждого файла вызываем процедуру `КопироватьФайлыВЛокальныйКаталог()`, в которую передаем описание оповещения, указывающее на экспортную процедуру `КопироватьФайлыЛокальноЗавершение()`, которая вернет процесс копирования файлов к следующему шагу цикла, а также имя копируемого файла.

Процедуру `КопироватьФайлыВЛокальныйКаталог()` заполним следующим образом (листинг 1.148).

Листинг 1.148. Процедура «КопироватьФайлыВЛокальныйКаталог»

```
&НаКлиенте
Процедура КопироватьФайлыВЛокальныйКаталог(Оповещение, ИмяФайла)

    Сообщение = Новый СообщениеПользователю;

    ПутьНаСервере = "ftp://" + ИмяПользователя + ":" + ПарольПользователя
                                                + "@" + Сервер + КаталогНаСервере;

    ДопПараметры = Новый Структура("ИмяФайла, Оповещение", ИмяФайла, Оповещение);

    Попытка
        // Копировать файл в локальный каталог.
```

```
            НачатьКопированиеФайла(Новый ОписаниеОповещения(
                                "КопироватьФайлыЗавершение", ЭтотОбъект, ДопПараметры.Оповещение),
                        ПутьНаСервере + ДопПараметры.ИмяФайла, ЛокальныйКаталог
                                + ДопПараметры.ИмяФайла);
        Исключение
            Сообщение.Текст = ОписаниеОшибки();
            Сообщение.Сообщить();
            Возврат;
        КонецПопытки;

КонецПроцедуры
```

В этой процедуре мы сначала упаковываем в структуру имя файла и имя процедуры оповещения, содержащиеся в параметрах `ИмяФайла` и `Оповещение`. Затем вызываем процедуру `НачатьКопированиеФайла()`, в которую передаем описание оповещения, указывающее на экспортную процедуру `КопироватьФайлыЗавершение()`, которая будет вызвана по окончании копирования файла (листинг 1.149). А также мы передаем в эту процедуру полный путь на FTP-сервере для доступа к файлу и полный путь на локальном компьютере, по которому будет скопирован FTP-ресурс.

Листинг 1.149. Процедура «КопироватьФайлыЗавершение»

```
&НаКлиенте
Процедура КопироватьФайлыЗавершение(СкопированныйФайл, ДополнительныеПараметры) Экспорт

    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = "Файл скопирован - " + СкопированныйФайл;
    Сообщение.Сообщить();

    ВыполнитьОбработкуОповещения(ДополнительныеПараметры);

КонецПроцедуры
```

В этой процедуре после окончания копирования файла мы вызываем процедуру оповещения `КопироватьФайлыЛокальноЗавершение()`, имя которой содержится в параметре `ДополнительныеПараметры` (листинг 1.150).

Листинг 1.150. Процедура «КопироватьФайлыЛокальноЗавершение»

```
&НаКлиенте
Процедура КопироватьФайлыЛокальноЗавершение(Результат, Дополнительно) Экспорт
    Возврат;
КонецПроцедуры
```

В результате мы вернемся в цикл копирования найденных файлов с FTP-сервера (см. листинг 1.147) в процедуре `КопироватьФайлыССервераЗавершение()`.

Электронная почта

Отправку и получение электронной почты из «1С:Предприятия» можно организовать с помощью объекта `ИнтернетПочта`. Объект `ИнтернетПочта` позволяет работать с почтовыми серверами напрямую. Для отправки писем используется протокол SMTP, а получать почту можно по протоколам POP3 и IMAP.

Для подключения к почтовому серверу необходимо знать его адрес, логин и пароль пользователя, адрес отправителя и т. д. Настройки обращения к почтовым серверам указываются в специализированном объекте `ИнтернетПочтовыйПрофиль`.

Ниже мы рассмотрим примеры отправки почты в виде форматированного текста и получения почты в виде текста HTML, а также отправки и получения сообщений обмена с вложенными файлами.

ПОДРОБНЕЕ

> Подробнее познакомиться с примерами работы по отправке и получению почты можно в демонстрационной конфигурации «Примеры работы», которая прилагается к книге.

В форме нашей демонстрационной обработки для работы с электронной почтой используются следующие параметры, которые хранятся в соответствующих реквизитах формы:

- СерверSMTP – сервер для отправки электронной почты, например `smtp.mail.ru`;
- СерверPOP3 – сервер для получения электронной почты, например `pop3.mail.ru`;
- СерверIMAP – сервер для получения электронной почты, например `imap.mail.ru`;
- ИспользоватьIMAP – признак получения писем по протоколу IMAP;
- Пользователь – имя пользователя (логин) на почтовом сервере. Для IMAP-сервера нужно указывать полное имя пользователя, например `<логин>@mail.ru`;
- Пароль – пароль пользователя на почтовом сервере;
- АдресОтправителя – почтовый адрес, с которого будет отправляться почта;
- АдресПолучателя – почтовый адрес, на который будет отправляться почта;
- ФайлВложения – полное имя отправляемого файла;

- Содержимое – содержимое отправляемого письма в виде форматированного документа;
- ТекстHTML – содержимое полученного письма в виде HTML-документа.

Отправить и получить почту

Прежде всего реализуем общую для всех примеров функцию `ПолучитьПрофиль()`, которая будет возвращать почтовый профиль (объект `ИнтернетПочтовыйПрофиль`), заполненный введенными в форме обработки настройками (листинг 1.151).

Листинг 1.151. Функция «ПолучитьПрофиль»

```
&НаКлиенте
Функция ПолучитьПрофиль()

    // Создать почтовый профиль.
    Профиль = Новый ИнтернетПочтовыйПрофиль;

    Профиль.АдресСервераSMTP = СерверSMTP;
    Профиль.ПользовательSMTP = Пользователь;
    Профиль.ПарольSMTP = Пароль;
    Профиль.ТолькоЗащищеннаяАутентификацияSMTP = Истина;

    Профиль.АдресСервераPOP3 = СерверPOP3;
    Профиль.Пользователь = Пользователь;
    Профиль.Пароль = Пароль;

    Профиль.АдресСервераIMAP = СерверIMAP;
    Профиль.ПользовательIMAP = Пользователь;
    Профиль.ПарольIMAP = Пароль;

    Возврат Профиль;

КонецФункции
```

Обратите внимание, что свойства профиля ПортSMTP, ПортIMAP и ПортPOP3 мы не указывали. По умолчанию значения этих свойств равны нулю. Это значит, что для защищенных или незащищенных соединений будут использоваться стандартные порты. Поэтому, если нет какой-то специфической задачи использовать нестандартные порты, значения этих свойств можно не задавать.

С помощью полученного почтового профиля выполняется подключение к нужному почтовому серверу.

Теперь рассмотрим пример, когда тело сообщения формируется с помощью форматированного документа, в него помещаются форматированный текст, картинки, ссылки, специальные символы. Затем это сообщение отправляется,

далее принимается с помощью почтового сервера и отображается в форме обработки в HTML-документе.

Для этого в демонстрационную обработку нужно добавить реквизит типа `ФорматированныйДокумент` и перетащить его в дерево элементов формы. Для отображения командной панели форматированного документа в форме нужно добавить над ним группу вида Командная панель и заполнить ее свойство Источник команд реквизитом формы, который будет содержать этот форматированный документ. А также в демонстрационную обработку нужно добавить строковый реквизит, перетащить его в дерево элементов формы и установить вид связанного с ним поля как Поле HTML документа.

Для отправки сообщений мы должны получить почтовый профиль с помощью функции `ПолучитьПрофиль()`, описанной выше (см. листинг 1.151), и сформировать отправляемое сообщение. Затем подключиться к SMTP-серверу методом `Подключиться(<Профиль>)` объекта `ИнтернетПочта`, отправить сообщение методом `Послать()` и отключиться от почтового сервера.

Для отправки сообщения, содержащего форматированный документ, можно использовать следующий фрагмент кода (листинг 1.152).

Листинг 1.152. Процедура «ОтправитьПочту»

```
&НаКлиенте
Процедура ОтправитьПочту(Команда)

    Перем HTML, Картинки;

    // Получить почтовый профиль.
    Профиль = ПолучитьПрофиль();

    // Сформировать сообщение, содержащее форматированный документ.
    Сообщение = Новый ИнтернетПочтовоеСообщение;
    Сообщение.Тема = "Форматированный документ ";
    Сообщение.Отправитель = АдресОтправителя;

    Сообщение.Получатели.Добавить(АдресПолучателя);

    // Создать вложения с картинками.
    Содержимое.ПолучитьHTML(HTML, Картинки);
    Для Каждого Картинка Из Картинки цикл
        Вложение = Сообщение.Вложения.Добавить(Картинка.Значение.ПолучитьДвоичныеДанные());
        Вложение.Идентификатор = Картинка.Ключ;
        Вложение.Имя = Картинка.Ключ;
        HTML = СтрЗаменить(HTML, Картинка.Ключ, "cid:" + Вложение.Идентификатор);
    КонецЦикла;

    ТекстСообщения = Сообщение.Тексты.Добавить(HTML, ТипТекстаПочтовогоСообщения.HTML);

    Почта = Новый ИнтернетПочта;
```

```
    Сообщ = Новый СообщениеПользователю();

    Попытка
        Почта.Подключиться(Профиль);
        // Отправить сообщение с форматированным текстом и картинками.
        Почта.Послать(Сообщение);
    Исключение
        // Вывести сообщение об ошибке при подключении к серверу или при отправке сообщения обмена.
        Сообщ.Текст = "Ошибка при отправке файла: " + ФайлВложения;
        Сообщ.Сообщить();
        Сообщ.Текст = ОписаниеОшибки();
        Сообщ.Сообщить();
        Возврат;
    КонецПопытки;

    Сообщ.Текст = "Сообщение отправлено. ";
    Сообщ.Сообщить();

    Почта.Отключиться();

КонецПроцедуры
```

Сначала мы получаем почтовый профиль, затем формируем сообщение обмена для отправки. При этом мы указываем отправителя в свойстве `Отправитель` объекта `ИнтернетПочтовоеСообщение` и добавляем получателя в список получателей сообщения – `Сообщение.Получатели.Добавить(АдресПолучателя)`.

Для отправки содержимого форматированного документа (реквизит Содержимое) используется его метод `ПолучитьHTML()`, который получает HTML-составляющие форматированного документа – массив картинок и HTML-текст. Затем полученные картинки добавляются как вложения в почтовое сообщение и сопоставляются с текстом письма путем установки свойства `Идентификатор` объекта `ИнтернетПочтовоеВложение`. HTML-текст добавляется в массив текстов почтового сообщения с указанием типа текста `ТипТекстаПочтовогоСообщения.HTML`. Остальная работа по отправке сообщения выполняется с помощью методов `Подключиться()` и `Послать()` объекта `ИнтернетПочта`.

Для получения сообщений мы должны получить почтовый профиль с помощью функции `ПолучитьПрофиль()`, описанной ранее (см. листинг 1.151). Затем, используя этот профиль, подключиться к почтовому серверу для получения почты по протоколу IMAP или POP3. Тип протокола интернет-почты можно передать вторым параметром в метод `Подключиться()` объекта `ИнтернетПочта` или использовать протокол по умолчанию (SMTP – для отправки почты, POP3 – для получения почты).

При получении писем надо учитывать, что при большом потоке писем полная их загрузка может занимать много времени. Чтобы решить эту проблему, существует возможность не загружать полностью все почтовые сообщения, а первоначально загружать только их заголовки.

С помощью метода `ПолучитьЗаголовки()` объекта `ИнтернетПочта` можно получать с сервера только заголовки сообщений. Благодаря этому, например, можно получить информацию о теме, отправителе и т. д. и решить, принимать это письмо, оставить его или сразу удалить.

Данный метод возвращает массив объектов типа `ИнтернетПочтовоеСообщение`. Каждый объект содержит информацию из заголовка сообщения. Вложения и тексты с сервера не принимаются.

Для получения сообщения, содержащего форматированный документ, и отображения его в поле HTML-документа можно использовать следующий фрагмент кода (листинг 1.153).

Листинг 1.153. Процедура «ПолучитьПочту»

```
&НаКлиенте
Процедура ПолучитьПочту(Команда)

	// Получить почтовый профиль.
	Профиль = ПолучитьПрофиль();

	Сообщ = Новый СообщениеПользователю();

	Почта = Новый ИнтернетПочта;

	Попытка
		Если ИспользоватьIMAP Тогда
			Почта.Подключиться(Профиль, ПротоколИнтернетПочты.IMAP);
		Иначе
			Почта.Подключиться(Профиль, ПротоколИнтернетПочты.POP3);
		КонецЕсли;

	Исключение
		// Вывести сообщение об ошибке при подключении к серверу.
		Сообщ.Текст = "Ошибка при подключении или приеме" + ОписаниеОшибки();
		Сообщ.Сообщить();
		Возврат;
	КонецПопытки;

	// Получить заголовки сообщений с отбором из почтового ящика.
	ПараметрыОтбораIMAP = Новый Структура;
	ПараметрыОтбораIMAP.Вставить("Прочитанные", Ложь);
	ЗаголовкиСообщений = Почта.ПолучитьЗаголовки(ПараметрыОтбораIMAP);

	КоличествоСообщений = ЗаголовкиСообщений.Количество();
	Если КоличествоСообщений = 0 Тогда
		Сообщ.Текст = "Сообщений в почтовом ящике нет.";
		Сообщ.Сообщить();
		Почта.Отключиться();
		Возврат;
	КонецЕсли;
```

```
    // Создать соответствие для установки флагов сообщений.
    ФлагиСообщенийIMAP = Новый Соответствие;

    // Получить полностью сообщения.
    МассивСообщений = Почта.Выбрать(Ложь, ЗаголовкиСообщений);

    Для Индекс = 0 По КоличествоСообщений - 1 Цикл

        ФлагСообщения = Новый ФлагиИнтернетПочтовогоСообщения();
        ФлагСообщения.Удаленное = Истина;
        ФлагиСообщенийIMAP.Вставить(МассивСообщений[Индекс].Идентификатор[0], ФлагСообщения);

        Сообщ.Текст = "Принято сообщение: " + МассивСообщений[Индекс].Тема;
        Сообщ.Сообщить();

        // Вывести содержимое письма в HTML документ для последнего сообщения.
        Если Индекс = КоличествоСообщений - 1 Тогда
            Для каждого Элемент Из МассивСообщений[Индекс].Тексты Цикл
                Если Элемент.ТипТекста = ТипТекстаПочтовогоСообщения.HTML Тогда
                    // Отобразить тело сообщения в HTML документе.
                    ТекстHTML = Элемент.Текст;
                    Если Найти(ТекстHTML, "<HTML>") = 0 Тогда
                        ТекстHTML = "<HTML><BODY>" + ТекстHTML + "</BODY></HTML>";
                    КонецЕсли;

                    Вложения = Новый Массив;
                    // Обработать вложения, чтобы правильно сформировать HTML.
                    Для каждого Вложение Из МассивСообщений[Индекс].Вложения Цикл
                        Ид = "cid:" + Вложение.Идентификатор;
                        Если Найти(ТекстHTML, Ид) <> 0 Тогда
                            Вложения.Добавить(Вложение);
                        КонецЕсли;
                    КонецЦикла;
                КонецЕсли;
            КонецЦикла;

            Индекс = 0;
            Для каждого Вложение Из Вложения Цикл
                // Записать файл картинки во временный файл.
                ФайлОбмена = Вложение.Данные;
                ИмяФайла = "c:\temp\pict_" + Строка(Индекс) + ".png";
                ФайлОбмена.Записать(ИмяФайла);

                // Отобразить картинку в HTML.
                Ид = """cid:" + Вложение.Идентификатор + """";
                ТекстHTML = СтрЗаменить(ТекстHTML, Ид, """ + ИмяФайла + """");
                Индекс = Индекс + 1;
            КонецЦикла;

        КонецЕсли;

    КонецЦикла;

    Почта.УстановитьФлагиСообщений(ФлагиСообщенийIMAP);

    Почта.Отключиться();

КонецПроцедуры
```

В данном примере для получения почты мы будем использовать протокол IMAP, поэтому с помощью метода `ПолучитьЗаголовки()` мы можем получить с сервера массив заголовков только тех сообщений, которые удовлетворяют параметрам отбора, указанным нами в структуре `ПараметрыОтбораIMAP`. Эту структуру мы передаем в метод `ПолучитьЗаголовки()`. В нашем примере мы используем отбор по непрочитанным письмам – `ПараметрыОтбораIMAP.Вставить("Прочитанные", Ложь)`.

Если же для получения почты используется протокол POP3, то параметры отбора игнорируются и с сервера получаются заголовки всех сообщений.

После этого массив отобранных заголовков сообщений мы передаем вторым параметром в метод `Выбрать()` объекта `ИнтернетПочта` и получаем нужные нам сообщения целиком (вместе с вложениями) в массиве `МассивСообщений`.

Затем мы создаем соответствие `ФлагиСообщенийIMAP` для установки флагов сообщений, которое мы будем заполнять в цикле обхода сообщений и передавать в метод `УстановитьФлагиСообщений()`. Этот метод доступен также только при подключении по протоколу IMAP.

В цикле обхода массива сообщений мы добавляем в соответствие элемент, ключом которого является идентификатор почтового сообщения (`МассивСообщений[Индекс].Идентификатор[0]`), а значением – объект `ФлагиИнтернетПочтовогоСообщения`. Свойство `Удаленное` этого объекта мы устанавливаем в значение `Истина`.

Для отображения содержимого полученного сообщения используется элемент формы ТекстHTML вида Поле HTML документа, связанный с соответствующим строковым реквизитом формы. В этот реквизит помещается HTML-текст прочитанного сообщения.

Для отображения картинок в HTML-документе мы заполняем массив `Вложения`. В цикле обхода этого массива мы сохраняем каждую картинку во временный файл и сопоставляем этот файл с текстом HTML.

В заключение устанавливаем флаги прочитанных сообщений методом `УстановитьФлагиСообщений()` объекта `ИнтернетПочта`, в результате чего эти сообщения удаляются с почтового сервера.

Итак, после отправки и получения почты форма обработки будет иметь следующий вид (рис. 1.42).

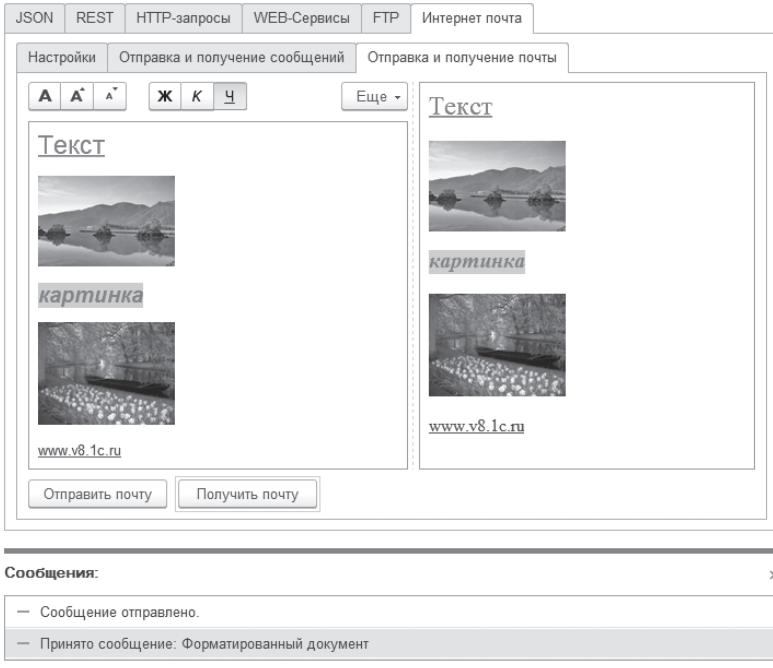


Рис. 1.42. Работа с почтой

Отправить и получить сообщение обмена

Теперь рассмотрим пример, позволяющий отправлять почтовые сообщения вместе с вложенными в них файлами обмена.

Процедуру для отправки сообщения заполним следующим образом (листинг 1.154).

Листинг 1.154. Процедура «ОтправитьСообщение»

```
&НаКлиенте
Процедура ОтправитьСообщение(Команда)

	// Получить почтовый профиль.
	Профиль = ПолучитьПрофиль();

	// Сформировать сообщение обмена.
	Сообщение = Новый ИнтернетПочтовоеСообщение;
	Сообщение.Тема = "СООБЩЕНИЕ_ОБМЕНА: " + ФайлВложения;
	Сообщение.Отправитель = АдресОтправителя;

	Сообщение.Получатели.Добавить(АдресПолучателя);
```

```
    Если ФайлВложения <> "" Тогда
        ФайлСообщения = Новый Файл(ФайлВложения);
    // Создать вложение с файлом обмена.
        Сообщение.Вложения.Добавить(ФайлВложения, ФайлСообщения.Имя);
    КонецЕсли;

    Почта = Новый ИнтернетПочта;

    Сообщ = Новый СообщениеПользователю();

    Попытка
        Почта.Подключиться(Профиль);
    // Отправить сообщение с файлом обмена.
        Почта.Послать(Сообщение);
    Исключение
        // Вывести сообщение об ошибке при подключении к серверу или при отправке сообщения обмена.
        Сообщ.Текст = "Ошибка при отправке файла: " + ФайлВложения;
        Сообщ.Сообщить();
        Сообщ.Текст = ОписаниеОшибки();
        Сообщ.Сообщить();
        Возврат;
    КонецПопытки;

    Сообщ.Текст = "Отправлен файл: " + ФайлВложения + " на адрес: " + АдресПолучателя;
    Сообщ.Сообщить();

    Почта.Отключиться();

КонецПроцедуры
```

Сначала мы получаем почтовый профиль, затем формируем сообщение обмена для отправки. При этом мы указываем отправителя в свойстве `Отправитель` объекта `ИнтернетПочтовоеСообщение` и добавляем получателя в список получателей сообщения – `Сообщение.Получатели.Добавить(АдресПолучателя)`. А также мы помечаем тему сообщения как «СООБЩЕНИЕ_ОБМЕНА: », чтобы при приеме почты, при получении заголовков сообщений, отобрать нужные нам сообщения по теме.

И если файл вложения был выбран, то добавляем его в коллекцию вложений сообщения–`Сообщение.Вложения.Добавить(ФайлВложения, ФайлСообщения.Имя)`.

Выбор файла вложения будет осуществляться по нажатию специальной кнопки. В обработчике соответствующей команды мы будем показывать диалог выбора файлов с помощью немодального метода `Показать()` объекта `ДиалогВыбораФайла`. В этот метод первым параметром мы передаем описание оповещения, указывающее на экспортную процедуру `ПослеВыбораФайлов()`, которая будет вызвана по закрытии диалога выбора файлов.

Обработчик команды для выбора файла вложения заполним следующим образом (листинг 1.155).

Листинг 1.155. Процедура «ВыбратьФайлВложения»

```
&НаКлиенте
Процедура ВыбратьФайлВложения(Команда)

    ВыборФайла = Новый ДиалогВыбораФайла(РежимДиалогаВыбораФайла.Открытие);
    ВыборФайла.Заголовок = "Выбор файла";
    ВыборФайла.Показать(Новый ОписаниеОповещения("ПослеВыбораФайлов", ЭтотОбъект));

КонецПроцедуры
```

В процедуру-обработчик оповещения `ПослеВыбораФайлов()` в параметре `ВыбранныеФайлы` передается массив выбранных имен файлов (листинг 1.156).

Листинг 1.156. Процедура «ПослеВыбораФайлов»

```
&НаКлиенте
Процедура ПослеВыбораФайлов(ВыбранныеФайлы, ДопПараметры) Экспорт

    Если ВыбранныеФайлы = Неопределено Тогда
        Возврат;
    КонецЕсли;

    ФайлВложения = ВыбранныеФайлы[0];

КонецПроцедуры
```

Теперь реализуем получение отправленного сообщения с почтового сервера. Процедуру для получения сообщения заполним следующим образом (листинг 1.157).

Листинг 1.157. Процедура «ПолучитьСообщение»

```
&НаКлиенте
Процедура ПолучитьСообщение(Команда)

    // Получить почтовый профиль.
    Профиль = ПолучитьПрофиль();

    Сообщ = Новый СообщениеПользователю();

    Почта = Новый ИнтернетПочта;

    Попытка
        Если ИспользоватьIMAP Тогда
            Почта.Подключиться(Профиль, ПротоколИнтернетПочты.IMAP);
        Иначе
            Почта.Подключиться(Профиль, ПротоколИнтернетПочты.POP3);
        КонецЕсли;

    Исключение
```

```
            // Вывести сообщение об ошибке при подключении к серверу или при приеме сообщения обмена.
            Сообщ.Текст = "Ошибка при подключении к серверу или приеме" + ОписаниеОшибки();
            Сообщ.Сообщить();
            Возврат;
    КонецПопытки;

    // Получить заголовки сообщений из почтового ящика.
    МассивЗаголовков = Новый Массив;
    ЗаголовкиСообщений = Почта.ПолучитьЗаголовки();

    Если ЗаголовкиСообщений.Количество() = 0 Тогда
            Сообщ.Текст = "Сообщений в почтовом ящике нет.";
            Сообщ.Сообщить();
            Почта.Отключиться();
            Возврат;
    КонецЕсли;

    // Отфильтровать по теме массив заголовков сообщений обмена.
    Для Индекс = 0 По ЗаголовкиСообщений.Количество() - 1 Цикл
            ЧастьТемы = "СООБЩЕНИЕ_ОБМЕНА: ";
            Если Лев(ЗаголовкиСообщений[Индекс].Тема, СтрДлина(ЧастьТемы)) = ЧастьТемы Тогда
                    МассивЗаголовков.Добавить(ЗаголовкиСообщений[Индекс]);
            КонецЕсли;
    КонецЦикла;


    Если МассивЗаголовков.Количество() = 0 Тогда
            Сообщ.Текст = "Сообщений обмена в почтовом ящике нет.";
            Сообщ.Сообщить();
            Почта.Отключиться();
            Возврат;
    КонецЕсли;

    // Получить полностью сообщения обмена, которые впоследствии будут удалены с сервера.
    МассивСообщенийОбмена = Почта.Выбрать(Истина, МассивЗаголовков);

    ФайлВложения = "";
    Для Индекс = 0 По МассивСообщенийОбмена.Количество() - 1 Цикл
            Если МассивСообщенийОбмена[Индекс].Вложения[0].Имя > ФайлВложения Тогда
                    // Выбрать сообщение с последним номером сообщения обмена.
                    ФайлВложения = МассивСообщенийОбмена[Индекс].Вложения[0].Имя;
                    ИндексСообщения = Индекс;
            КонецЕсли;
    КонецЦикла;

    Если ИндексСообщения <> Неопределено Тогда
            // Записать файл обмена во временный файл.
            ФайлОбмена = МассивСообщенийОбмена[ИндексСообщения].Вложения[0];
            ФайлОбмена.Данные.Записать("c:\temp\" + ФайлОбмена.ИмяФайла);
            Сообщ.Текст = "Принят файл обмена данными: "
                                    + ФайлВложения + " с адреса: " + АдресОтправителя;
            Сообщ.Сообщить();
    КонецЕсли;

    Почта.Отключиться();

КонецПроцедуры
```

В этой процедуре мы подключаемся к серверу для получения почты, затем с помощью метода `ПолучитьЗаголовки()` получаем с сервера массив заголовков всех сообщений (объектов типа `ИнтернетПочтовоеСообщение`). Затем из массива `ЗаголовкиСообщений` в массив `МассивЗаголовков` отфильтровываем только те сообщения, тема которых начинается с префикса «СООБЩЕНИЕ_ОБМЕНА: ».

Если `МассивЗаголовков` пуст, то, значит, сообщений обмена на сервере нет, поэтому мы отключаемся от почтового сервера и заканчиваем работу процедуры. Иначе мы передаем массив заголовком вторым параметром в метод `Выбрать()`объекта `ИнтернетПочта` и получаем нужные нам сообщения обмена целиком (вместе с вложениями) в массиве `МассивСообщенийОбмена`. Первым параметром в этот метод мы передаем `Истина` (это значение по умолчанию). Это означает, что полученные сообщения обмена будут удалены с сервера.

В заключение среди всех сообщений обмена мы находим сообщение с последним номером файла вложения, сохраняем его на компьютер и отключаемся от почтового сервера.

Кроме того, можно заключить процедуру получения сообщений в операторы `Попытка ... Исключение`, что позволит продолжить получение, если при приеме определенного сообщения произошла ошибка. Также можно поместить в форме индикатор для графического отображения процесса получения сообщений.

Глава 2. Внешние источники данных

В процессе работы прикладных решений «1С:Предприятия» может возникнуть потребность получить и изменить данные, хранящиеся во внешних базах данных, созданных с помощью различных сторонних СУБД (Microsoft SQL Server, PostgreSQL, MySQL и т. п.). Эти базы данных могут быть как реляционными, так и аналитическими.

В этой главе будет подробно, с примерами, рассмотрена работа с реляционными источниками данных, в которых информация представлена в виде взаимосвязанных двумерных таблиц, каждая из которых содержит информацию об объектах определенного типа.

Данные в аналитических базах данных (OLAP-системах) содержат агрегированную информацию в виде многомерных пространств, так называемых кубов. Работа с аналитическими источниками данных в целом аналогична получению данных из реляционных баз данных. Этот вопрос более подробно рассмотрен в документации «1С:Предприятия» в разделе «Глава 23. Внешние источники данных – Работа с внешним источником данных OLAP».

Для доступа к внешним данным в платформе «1С:Предприятие» существует объект конфигурации `ВнешниеИсточникиДанных`. В состав внешнего источника данных могут входить таблицы и функции реляционного источника данных и кубы аналитического источника данных.

Работа с реляционными внешними источниками данных

Для того чтобы использовать информацию из внешних источников данных в прикладном решении «1С:Предприятия», нужно:

- Изучить структуру внешнего источника данных и понять, какая информация (таблицы и поля или функции) необходима для работы алгоритмов или отчетов в «1С:Предприятии».
- Создать объект `ВнешнийИсточникДанных`, содержащий необходимые подчиненные таблицы и поля. Это можно либо сделать вручную, либо заполнить структуру данных этого объекта из внешнего источника данных.
- Реализовать использование созданных объектов в прикладном решении.
- Выполнить настройку параметров подключения к внешнему источнику данных в той сети, где будет использоваться прикладное решение. Эти параметры могут отличаться от тех, которые использовались при загрузке структуры внешнего источника данных.

ВНИМАНИЕ!

Параметры доступа к внешнему источнику данных, которые были заданы в конфигураторе, не будут использованы системой в режиме 1С:Предприятие. В общем случае при подключении к внешнему источнику данных в конфигураторе могут использоваться одни параметры, а при подключении в режиме 1С:Предприятие – другие.

Общая информация

Объект, описывающий внешний источник данных, подключенный к реляционной базе данных, может быть использован в «1С:Предприятии» следующим образом:

- в качестве источника данных для запросов;
- в качестве источника данных в системе компоновки данных;
- в качестве источника для динамических списков;
- входить в состав общих реквизитов;
- записи таблиц могут отображаться в управляемых формах «1С:Предприятия» (для объектов внешних источников данных не поддерживается использование обычных форм);
- таблицы внешнего источника данных могут выступать в качестве типов реквизитов информационной базы;

- к таблицам (и полям) внешних источников данных можно применять права доступа и накладывать ограничения доступа к данным;
- доступ к таблицам и полям возможен из встроенного языка;
- таблица внешнего источника данных может входить в состав подсистем;
- таблица внешнего источника данных может входить в состав функциональных опций.

ВНИМАНИЕ!

Механизм внешних источников данных не должен использоваться для доступа к базам данных «1С:Предприятия», так как модель данных «1С:Предприятия» не рассчитана на работу с данными на уровне физических структур хранения в системах управления базами данных (СУБД). Возможность записи во внешние источники данных не должна использоваться для замены штатного механизма хранения данных прикладного решения.

Возможности системы «1С:Предприятие» максимально учитываются при использовании в качестве источника внешних данных следующих СУБД (свойство Тип СУБД параметров соединения с источником данных):

- Microsoft SQL Server,
- IBM DB2,
- PostgreSQL,
- Oracle Database,
- MySQL.

При использовании других СУБД возможности работы с внешними источниками данных зависят от самой СУБД.

ПРИМЕЧАНИЕ

При работе с внешним источником данных могут возникать ошибки в том случае, если свойство Тип СУБД параметров соединения с внешним источником данных содержит значение, которое не соответствует реально используемой системе.

ПОДРОБНЕЕ

Более подробно получение информации из реляционного источника данных описано в документации «1С:Предприятия» в разделе «Глава 24. Внешние источники данных – Работа с реляционными внешними источниками данных».

Для получения доступа к внешним источникам данных используется механизм ODBC. Данные внешних источников данных доступны как для чтения, так и для записи.

Для подключения к внешнему источнику данных следует сформировать строку подключения, которая может содержать в себе либо все параметры подключения, необходимые для выбранного драйвера ODBC, либо указание на сформированное описание источника данных DSN (Data Source Name). Подробнее об этом рассказывается в следующем разделе.

Строка соединения

Для подключения к внешнему источнику данных с помощью DSN можно воспользоваться специальными утилитами администрирования ODBC, которые могут сформировать описание источника данных, и затем указать в параметрах соединения специальную конструкцию вида `DSN=<ИмяDSN>`. Описания источника данных могут быть пользовательскими и системными. Пользовательские описания источников данных доступны на данном компьютере и только тому пользователю, который создавал этого описание. Системное описание создается системным администратором и доступно всем пользователем данного компьютера.

Чтобы в ОС Windows запустить эту утилиту, нужно выполнить команду Панель управления > Администрирование > Источники данных (ODBC). При этом открывается утилита администрирования, соответствующая версии операционной системы (рис. 2.1, 2.2).

ПОДРОБНЕЕ

> Более подробно процесс создания DSN будет показан в разделе «Примеры использования».

Следует помнить, что в 64-разрядной версии ОС Windows существуют разные утилиты администрирования для 32-разрядного и 64-разрядного ODBC. Поэтому для создания описания источника данных следует использовать версию утилиты администрирования, соответствующую версии системы «1С:Предприятия», которая будет исполнять запросы к внешнему источнику данных. Так, например, если обращение к ODBC выполняется из 64-разрядной версии «1С:Предприятия», то надо использовать 64-разрядную версию утилиты администрирования.

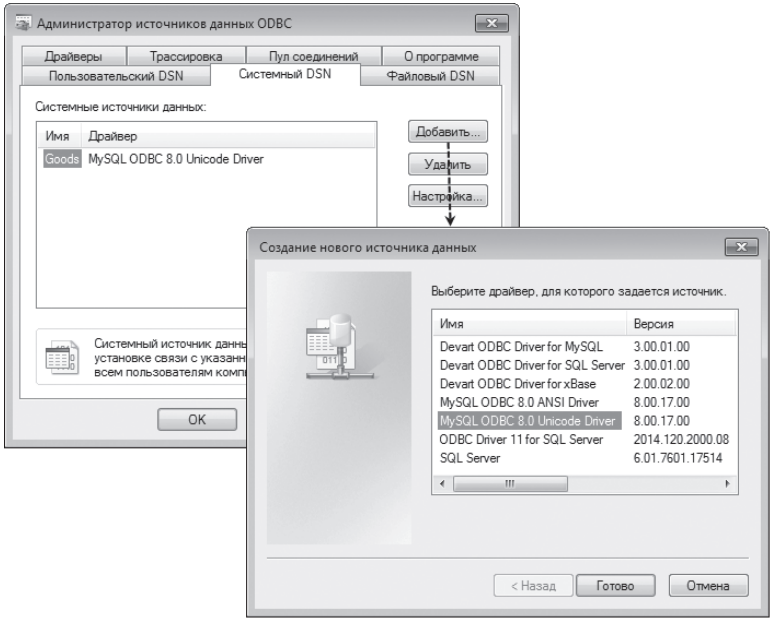


Рис. 2.1. Создание источника данных ODBC

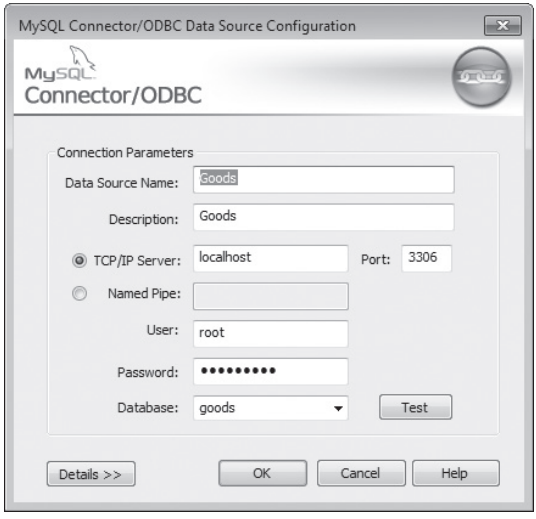


Рис. 2.2. Создание источника данных ODBC

На 64-разрядной версии ОС Windows утилиты администрирования расположены следующим образом:

- 64-разрядная версия: %SYSTEMROOT%\System32\odbcad32.exe;
- 32-разрядная версия: %SYSTEMROOT%\SysWOW64\odbcad32.exe.

В ОС Linux эта утилита называется ODBCConfig и доступна при установке соответствующего пакета (зависит от версии ОС Linux).

При описании полной строки соединения параметры строки соединения должны записываться парами `КлючевоеСлово=Значение` и разделяться символом «;». Для упрощения формирования строки соединения можно воспользоваться сайтом `http://www.connectionstrings.com/`.

Таким образом, при подключении к внешнему источнику данных строка соединения может иметь вид:

```
DRIVER=MySQL ODBC 8.0 Unicode Driver;User ID=root;
Password=password;Data Source=localhost;Database=spa
```

Или же при подключении с помощью предварительно настроенного источника данных (DSN):

```
DSN=MyDB
```

Также следует помнить, что соединение с внешним источником данных следует указывать не только в конфигураторе (если используется механизм импорта структуры таблиц из внешнего источника данных), но и в режиме «1С:Предприятие» для получения собственно данных.

Редактирование структуры внешнего источника данных

Для работы с внешними источниками данных предназначена ветвь дерева конфигурации Внешние источники данных. В этой группе объектов можно добавить внешний источник данных, определить, какие таблицы будет содержать этот источник, и затем описать состав полей (и их свойства), которые будут храниться в каждой таблице.

При этом в конфигурации может быть описано меньшее количество таблиц и полей в них, чем есть в реальной базе данных, но нельзя задать таблицу или поле, которое отсутствует в реальной базе данных.

Создание структуры внешнего источника данных возможно как в ручном режиме, так и при загрузке этой структуры с помощью специального конструктора. Использование конструктора наиболее просто и надежно. Платформа сама безошибочно настроит все основное, что нужно для работы с внешними источниками данных. Но бывают такие ситуации, когда требу-

ется добавить или изменить состав таблиц и полей источника данных вручную. Такой вариант тоже возможен, хотя он более трудоемкий и чреват ошибками.

Создание таблиц внешнего источника данных с помощью конструктора

Чтобы загрузить список таблиц и полей из внешней базы данных, нужно в окне редактирования свойств внешнего источника на закладке Данные нажать кнопку Добавить и выбрать пункт Выбрать из списка таблиц внешнего источника данных в конструкторе таблиц внешнего источника данных (рис. 2.3).

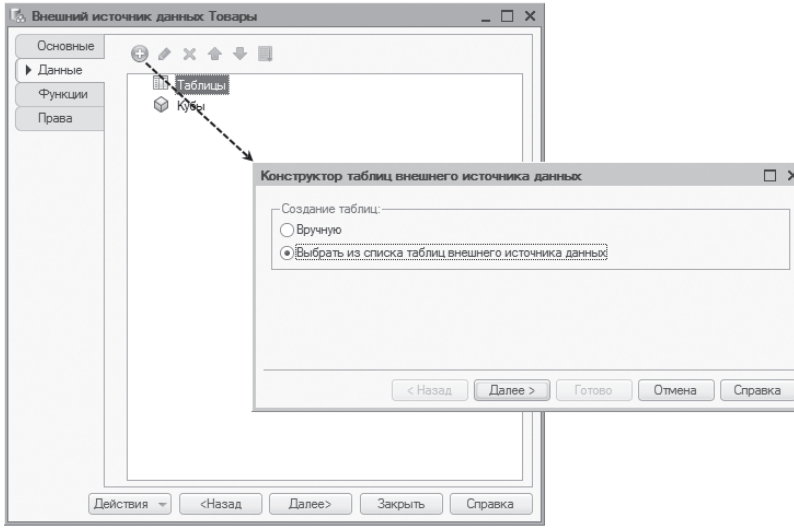


Рис. 2.3. Загрузка структуры таблиц из внешнего источника данных

Затем нужно указать строку соединения с внешней базой данных, воспользовавшись окном Подключение к источнику данных. Например, можно подключиться с помощью ранее созданного DSN, указав в строке соединения имя источника данных ODBC (рис. 2.4).

После того как подключение к внешней базе успешно выполнено, будет открыт список с перечнем таблиц и полей подключенного источника данных. Из этого списка можно выбрать необходимые таблицы и поля, на основе которых платформа создаст соответствующую структуру объектов, описывающих текущий источник данных (рис. 2.5).

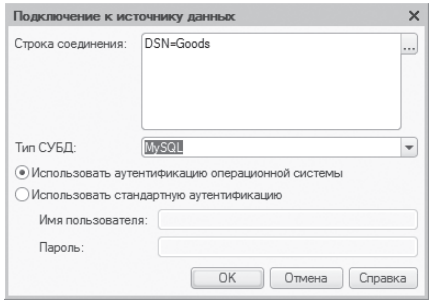


Рис. 2.4. Подключение к источнику данных ODBC

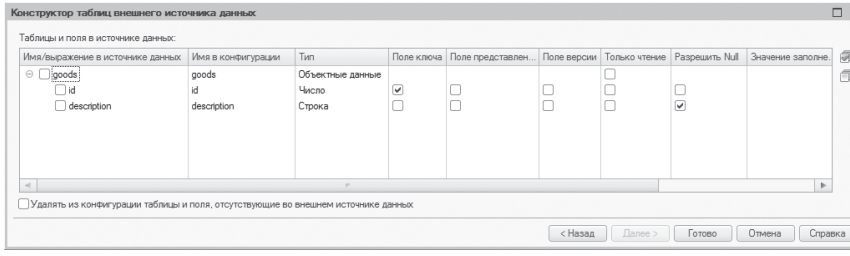


Рис. 2.5. Конструктор таблиц внешнего источника данных

При получении структуры внешнего источника данных платформа автоматически пытается определить, какие данные расположены в таблице: объектные или необъектные. Таблица будет считаться объектной, если для нее указано только одно ключевое поле, и необъектной в противном случае. Если платформа ошиблась с указанием ключевых полей, можно вручную изменить состав полей, которые образуют ключ таблицы. Если таблица определена как объектная, у нее можно указать поле, которое формирует представление данных такого типа. Поле представления необходимо указать вручную.

Затем платформа преобразует типы полей таблиц внешнего источника данных к типам «1С:Предприятия»: число, строка, дата, булево, уникальный идентификатор, двоичные данные и типы, связанные с объектными таблицами источников данных. Если платформа неправильно выбрала тип колонки таблицы (например, ссылочное поле), его можно изменить вручную.

Кроме того, в конструкторе загрузки структуры внешнего источника данных можно настроить и другие параметры:

- имя объекта в конфигурации – колонка Имя в конфигурации;
- тип загружаемого реквизита – колонка Тип;

- указать поля, входящие в состав ключа (для объектных данных), – колонка Поле ключа;
- указать поле, хранящее представление объекта, – колонка Поле представления;
- указать поле, хранящее версию объекта (для осуществления оптимистической блокировки данных), – колонка Поле версии;
- указать, что таблица или поле доступны только на чтение, – колонка Только чтение;
- указать возможность записывать значение `NULL` в поле – колонка Разрешить NULL;
- задать значение заполнения для реквизита – колонка Значение заполнения.

После настройки указанных параметров, отметки нужных таблиц и полей и нажатия кнопки Готово произойдет загрузка структуры внешнего источника данных в конфигурацию.

ПОДРОБНЕЕ

Более подробно загрузка структуры внешнего источника данных будет показана в разделе «Примеры использования».

Создание таблиц внешнего источника данных вручную

Помимо этого список таблиц внешнего источника данных можно полностью заполнить вручную в конфигураторе. То есть таблицы и их поля можно создать обычным образом, как подчиненные внешнему источнику объекты конфигурации, и установить их свойства.

При этом нужно иметь в виду, что в одной таблице прикладного решения могут быть размещены данные только из одной физической таблицы сторонней СУБД.

Обратим внимание на наиболее важные свойства таблиц внешнего источника данных. Они собраны на закладке Данные окна редактирования свойств таблицы (рис. 2.6).

При ручном заполнении списка таблиц и полей нужно следить, чтобы их свойство Имя в источнике данных в точности соответствовало имени таблиц и полей внешнего источника данных. В отличие от этого при заполнении с помощью конструктора основные свойства таблиц и полей определяются автоматически.

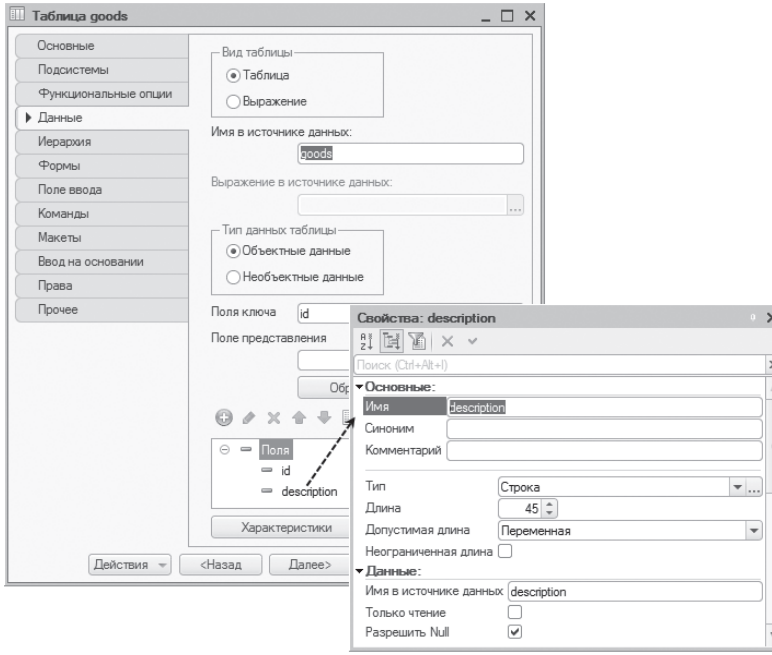


Рис. 2.6. Свойства таблицы внешнего источника данных

Также очень важно правильно установить свойство Тип данных таблицы. Нужно отметить, какие данные хранятся в этой таблице: объектные или необъектные. Если в таблице можно выделить одно поле, которое однозначно определяет запись в таблице, значит, таблица хранит объектные данные. Ближайшим аналогом таких таблиц служит справочник. Если запись в таблице идентифицируется несколькими ключевыми полями, то такая таблица содержит необъектные данные. Ближайшим аналогом таких таблиц служит регистр сведений. Ключевые поля таблицы необходимо указать в свойстве Поля ключа.

Для таблиц, содержащих объектные данные, можно указать поле, которое будет выступать в качестве представления объекта. Это можно сделать с помощью свойства Поле представления.

В зависимости от того, какой объект СУБД описывает создаваемая таблица, следует корректно установить свойство Вид таблицы. Если свойство установлено в значение Таблица, то объект конфигурации будет выступать аналогом реальной таблицы базы данных или представления (view).

Для таблицы вида Выражение следует указать свойство Выражение в источнике данных. В простейшем случае это будет вызов функции, возвращающей табличные данные, например `dbo.ufnGetContactInformation(&1)`. Если в это поле записывается выражение, то рекомендуется указать его в скобках для минимизации различных конфликтов при реальном выполнении запроса к СУБД.

Свойство Только чтение отвечает за возможность записи в таблицу внешнего источника данных. Это свойство автоматически устанавливается в значение `Истина` для таблиц, основанных на представлениях (view) и функциях. Для таблицы, в которую допустима запись информации, флажок Только чтение должен быть сброшен. При записи в таблицу важны следующие свойства:

- Уровень изоляции транзакций – определяет уровень изоляции транзакций, который будет устанавливаться при неявных транзакциях записи в эту таблицу.
- Поле версии данных – в данном свойстве указывается поле внешнего источника данных, которое увеличивает свое значение автоматически при каждой записи в данную таблицу. Если поле указано, то система при интерактивном изменении объекта опирается на значение этого поля. При записи выполняется сравнение значений этого поля в базе данных и в форме: если значения различаются, значит, объект изменен. Если поле не указано, то при интерактивном открытии формы происходит считывание объекта в память, а при записи происходит повторное чтение объекта и сравнение с копией в памяти. Если две копии различаются, значит, объект изменен.
- Поля блокировки данных – указывает поля, по которым можно выполнять блокировку данных.
- Вводится на основании – указывает объекты конфигурации, на основании которых могут создаваться записи в таблице.
- Режим управления блокировкой данных – указывает, какой режим управления блокировками будет применяться при записи в данную таблицу.

Свойства полей

Затем у таблиц внешнего источника данных вида Таблица нужно указать список полей, которые описывают, какие данные физической таблицы будут доступны из прикладного решения. Свойство поля Имя в источнике данных служит для указания соответствия между полем прикладного решения и полем физической таблицы. Поэтому значение в этом свойстве должно в точности соответствовать имени колонки этой таблицы.

Необходимо помнить, что если значение, находящееся в свойстве Имя в источнике данных, заключено в одинарные кавычки, то в SQL-запрос к базе данных это значение попадает без преобразований вне зависимости от состава символов. Если значение в поле не заключено в одинарные кавычки, то в SQL-запрос к базе данных это значение попадет, заключенное в двойные кавычки, если в имени содержатся спецсимволы.

В свойстве Тип нужно указать тип поля. Выбор ограничен следующими типами:

- Число;
- Строка;
- Дата;
- Булево;
- УникальныйИдентификатор;
- ДвоичныеДанные;
- типы, которые определяются таблицами внешних источников данных.

Если поле имеет составной тип, то в таком составном типе могут участвовать только типы Число, Строка, Дата, Булево.

Свойство Только чтение указывает, может ли выполняться запись в данное поле таблицы. Это свойство следует устанавливать в значение Истина для полей, рассчитываемых автоматически (AUTOINCREMENT), вычисляемых полей и т. п.

Однако при работе с внешним источником данных иногда возникает необходимость выполнять запись полей, которые в обычной работе предназначены только для чтения, или наоборот. Для этого можно временно изменить (как сузить, так и расширить) список записываемых полей с помощью методов ПолучитьИзменяемыеПоля() и УстановитьИзменяемыеПоля(). Соответствующий пример будет показан в разделе «Примеры использования».

Таким образом, состояние свойства Только чтение описывает поведение поля по умолчанию, а с помощью указанных методов можно в редких случаях изменять поведение поля.

Свойство Разрешить NULL указывает, можно ли в поле таблицы записать значение NULL. Это свойство следует устанавливать в значение Истина для всех полей, кроме ключевых и тех полей, которые во внешнем источнике описаны как NOT NULL.

Работа с функциями внешнего источника данных

Иногда в «1С:Предприятии» требуется использовать специфические возможности различных СУБД, такие как функции, последовательности, хранимые процедуры СУБД, к которым подключен внешний источник данных прикладного решения. В качестве примера можно привести объектную таблицу СУБД, ключ которой получается из последовательности. Функции и процедуры могут ничего не возвращать (процедура), а также возвращать одиночное значение или таблицу.

Общая информация

Функции внешнего источника данных в «1С:Предприятии» могут быть следующих видов:

- функция, которая возвращает какое-либо значение;
- функция, которая не возвращает значения (процедура);
- таблица вида Выражение в реляционном источнике данных.

Функции могут использоваться во встроенном языке, языке запросов и системе компоновки данных. Для функций можно указывать права доступа, регламентирующие использование этих функций.

Если во внешнем источнике данных существуют функции, возвращающие таблицы, то необходимо также определить таблицы внешнего источника данных вида Выражение.

В конфигурации может быть описано меньшее количество функций и процедур, чем есть в реальной базе данных.

Также можно написать некоторую функцию (на языке используемой СУБД), которая отсутствует в реальной базе данных. При этом надо учитывать следующие особенности:

- Если функция возвращает значение, то при использовании во встроенном языке ее выражение, указанное в свойстве Выражение в источнике данных, подставляется в конструкцию `select <...>`. А при использовании в языке запросов (и системе компоновки данных) происходит замена вызова функции на собственно выражение.
- Если функция не возвращает значение, то не рекомендуется использование произвольного выражения для такой функции с целью максимальной совместимости с различными СУБД. Использование таких функций не поддерживается в языке запросов и системе компоновки данных.

ПОДРОБНЕЕ

Более подробно работа с функциями внешнего источника данных описана в документации «1С:Предприятия» в разделе «Глава 24. Внешние источники данных – Работа с процедурами и функциями внешних источников данных».

Редактирование функций

Создание функций внешнего источника данных возможно как в ручном режиме, так и при загрузке списка функций с помощью специального конструктора.

Чтобы загрузить список функций из внешней базы данных, нужно в окне редактирования свойств внешнего источника на закладке Функции нажать кнопку Добавить и выбрать пункт Выбрать из списка функций внешнего источника данных в конструкторе функций внешнего источника данных (рис. 2.7).

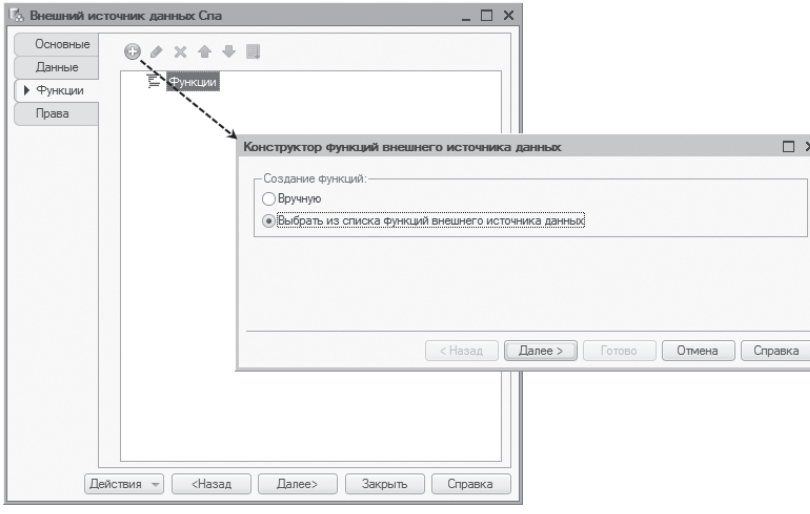


Рис. 2.7. Загрузка списка функций из внешнего источника данных

Затем нужно указать строку соединения с внешней базой данных, воспользовавшись окном Подключение к источнику данных (см. рис. 2.4).

После того как подключение к внешней базе успешно выполнено, будет открыт список с перечнем процедур и функций подключенного источника данных. Из этого списка можно выбрать необходимые функции, на основе которых платформа создаст соответствующую структуру объектов, описывающих текущий источник данных (рис. 2.8).

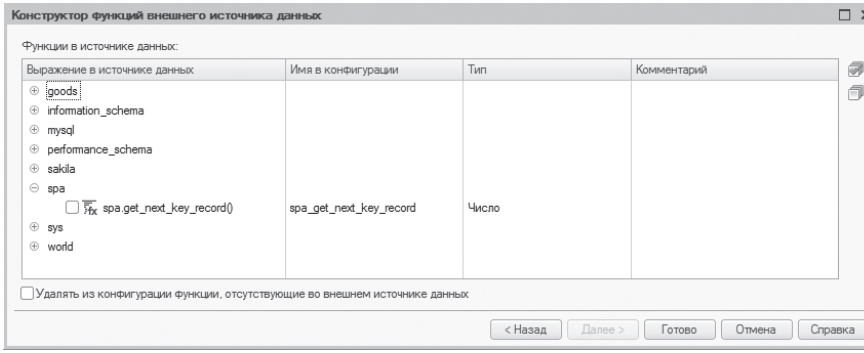


Рис. 2.8. Конструктор функций внешнего источника данных

В конструкторе можно указать, под каким именем функция будет создана в конфигурации «1С:Предприятия».

После настройки указанных параметров, отметки нужных функций и процедур и нажатия кнопки Готово произойдет загрузка функций внешнего источника данных в конфигурацию.

ПОДРОБНЕЕ

> Более подробно загрузка функций из внешнего источника данных будет показана в разделе «Примеры использования».

Помимо этого список функций внешнего источника данных можно полностью заполнить вручную в конфигураторе. То есть функции можно создать обычным образом, как подчиненные внешнему источнику объекты конфигурации, и установить их свойства.

При ручном заполнении списка функций особое внимание нужно обратить на свойство Выражение в источнике данных. Значение этого свойства должно содержать в себе выражение, которое будет использоваться в реальном запросе к базе данных при использовании функции. В данном свойстве можно не только указать обращение к реальной функции или сохраненной процедуре базы данных, но и написать выражение на языке СУБД внешнего источника данных.

Если с помощью функции выполняется обращение к реальной функции базы данных, то значение в этом свойстве должно в точности соответствовать имени и числу параметров исходной функции.

Можно также добавить функцию внешнего источника данных, которая отсутствует в реальной базе данных. Эта функция должна быть написана

на языке используемой СУБД. При этом рекомендуется выражение, помещенное в свойство Выражение в источнике данных, заключать в круглые скобки, чтобы избежать различных побочных эффектов во время исполнения реального запроса.

Если свойство Возвращает значение установлено в значение `Истина`, то в этом случае также необходимо указать, какого типа значение будет возвращаться данной функцией или выражением.

Если функция принимает параметры, то формальные параметры функций внешнего источника данных описываются как `&n`, где `n` – номер параметра. Нумерация параметров начинается с 1. Например, функция, принимающая два параметра, описывается следующим образом: `dbo.uspGetWhereUsedProductID(&1, &2)`.

Управление внешними источниками данных

В режиме «1С:Предприятие» при выполнении любой операции, в ходе которой необходимо получать данные из внешнего источника данных, происходит попытка подключения к этому источнику, если подключение еще не было выполнено.

Если подключение не выполнено, в клиентском приложении возникает исключение. В этом случае пользователю предлагается диалог, в котором можно уточнить параметры подключения и повторить подключение. Если подключение выполнено успешно, то пользователю будет предложено повторить действие, во время которого произошла ошибка подключения.

Также имеется возможность заранее выполнить подключение вручную. Для этого предназначена стандартная функция Управление внешними источниками данных, которая вызывается из меню Все функции > Стандартные функции (рис. 2.9).

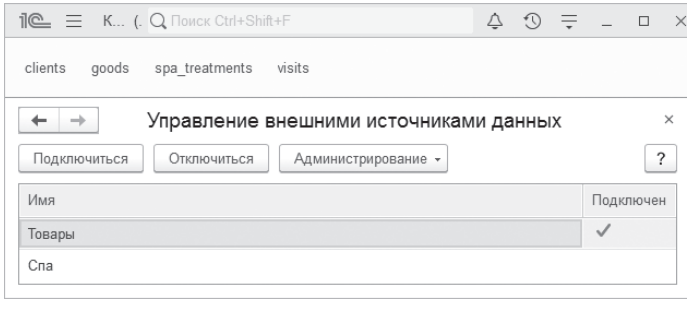


Рис. 2.9. Управление внешними источниками данных

Пользователю, открывшему эту обработку, будут доступны все внешние источники данных, созданные в конфигураторе, на которые он имеет права.

Колонка Подключен указывает состояние подключения к источнику данных в этом сеансе.

С помощью группы команд Администрирование можно указать общие параметры подключения (команда Изменить общие параметры...) и параметры подключения конкретных пользователей в том случае, если эти параметры отличаются от общих параметров (команда Изменить параметры пользователей...) – рис. 2.10.

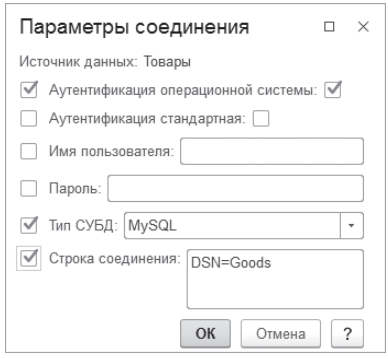


Рис. 2.10. Параметры соединения

Флажок перед именем параметра означает, что данный параметр используется в данном наборе параметров соединения.

Нажатие кнопки Подключиться в обработке Управление внешними источниками данных (см. рис. 2.9) открывает диалог, в котором можно задать (или уточнить) параметры подключения к выбранному внешнему источнику данных (рис. 2.11).

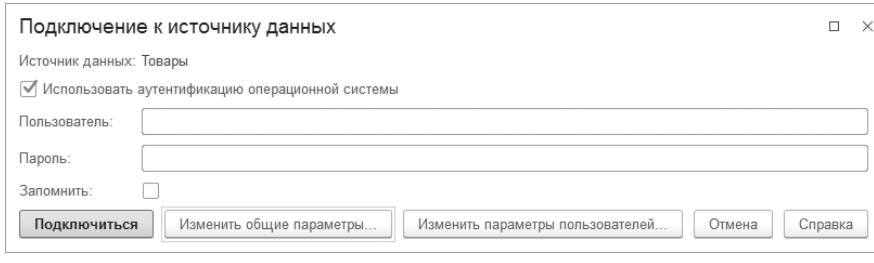


Рис. 2.11. Параметры подключения

Необходимость указания пользователя и пароля в данном диалоге зависит от драйвера ODBC и содержания строки соединения. Возможны ситуации, когда ввод логина и пароля не требуется.

Если пользователь не является администратором внешнего источника данных, то кнопки Общие параметры… и Параметры пользователя… для него недоступны. Флажок Использовать аутентификацию операционной системы доступен только в том случае, если у пользователя имеется право ИзменениеАутентификацииОССеанса.

После нажатия кнопки Подключиться происходит подключение к внешнему источнику данных. В случае успешного подключения изменяется содержимое колонки Подключен списка внешних источников данных.

Примеры использования

В этом разделе мы рассмотрим различные примеры использования и модификации данных во внешних источниках из прикладного решения «1С:Предприятия». В качестве внешних источников данных будут выступать реляционные схемы (базы данных), созданные с помощью СУБД MySQL. Подключение к этим базам данных будет выполняться как с помощью полной строки соединения, так и с помощью заранее созданного DSN.

Сначала мы реализуем настройку внешних источников данных в конфигураторе, затем покажем подключение к ним в режиме 1С:Предприятие и отображение, а также возможность интерактивного редактирования данных, полученных из внешних источников.

После этого рассмотрим, как программно заполнить данные прикладного решения (например, справочник на основе данных, содержащихся во внешней таблице), а при добавлении или изменении записей справочника покажем, как синхронизировать эту информацию в «1С:Предприятии» с данными внешней таблицы.

Также мы рассмотрим работу с функциями внешнего источника, одну из которых мы создадим самостоятельно на языке MySQL.

Кроме того, покажем пример использования данных из внешних источников для работы стандартных учетных механизмов в прикладном решении «1С:Предприятия».

В заключение продемонстрируем небольшой отчет, построенный с помощью системы компоновки данных, показывающий информацию из внешнего источника.

Исходная информация для примеров

Как уже говорилось, в качестве внешних источников данных для наших примеров мы будем использовать схемы, созданные с помощью СУБД MySQL. Эта СУБД предварительно установлена на локальный компьютер, на котором будут демонстрироваться примеры. Вместе с ней также установлена оболочка MySQL Workbench, с помощью которой мы создадим внешние схемы, таблицы и функции.

Итак, сначала создадим схему goods. В этой схеме создадим таблицу goods с ключевым полем id типа `int(10)` и полем description типа `varchar(45)`. У поля ключа включены свойства `Primary Key` и `Not Null`. Это значит, что поле id – это поле первичного ключа, и в этом поле не может содержаться значение `Null`. В поле description, наоборот, по умолчанию записывается `Null`.

Добавим в эту таблицу несколько записей. Структура таблицы goods и содержащаяся в ней информация показаны на рис. 2.12.

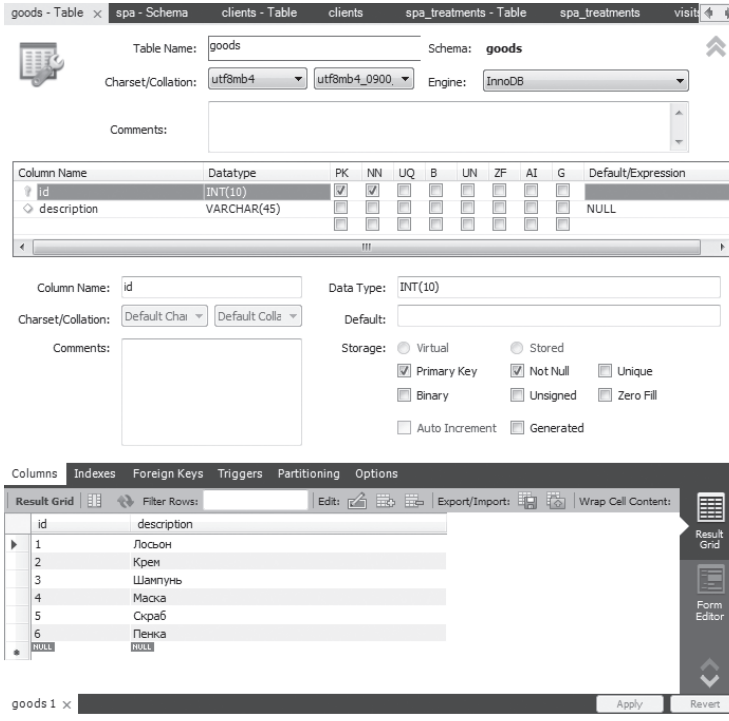


Рис. 2.12. Структура и данные таблицы «goods»

Затем создадим схему spa. В этой схеме создадим таблицу clients с ключевым полем id типа `int(10)` и полем name типа `varchar(30)`. У поля ключа включены свойства `Primary Key`, `Not Null`, `Unsigned`, `Auto Increment`. Это значит, что поле id – это поле первичного ключа, самовозрастающее, беззнаковое и в этом поле не может содержаться значение `Null`. В поле name, наоборот, по умолчанию записывается `Null`.

Добавим в эту таблицу несколько записей. Структура таблицы clients и содержащаяся в ней информация показаны на рис. 2.13.

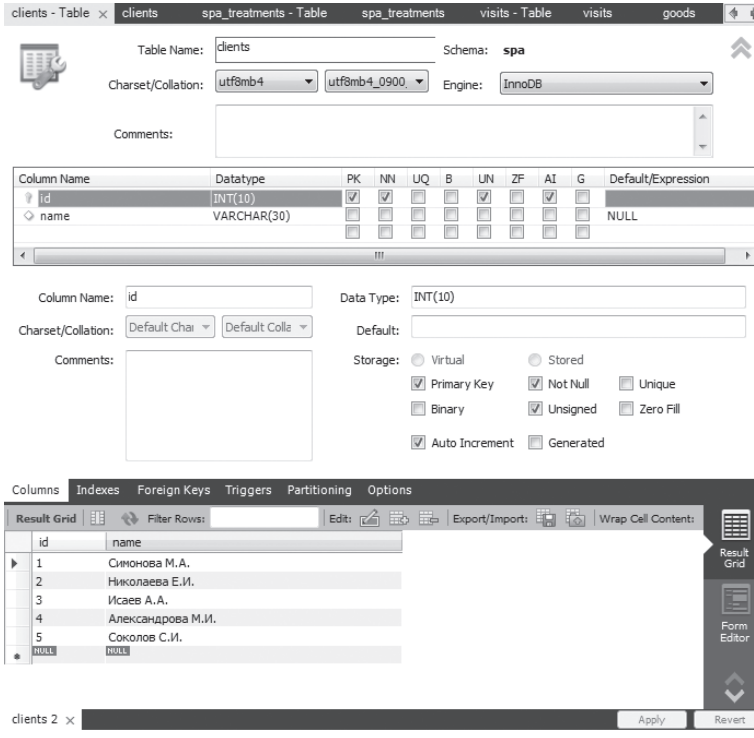


Рис. 2.13. Структура и данные таблицы «clients»

В схеме spa создадим еще одну таблицу spa_treatments с ключевым полем id типа `int(10)` и полем description типа `varchar(45)`. У поля ключа включены свойства `Primary Key`, `Not Null`, `Unsigned`, `Auto Increment`. Это значит, что поле id – это поле первичного ключа, самовозрастающее, беззнаковое и в этом поле не может содержаться значение `Null`. В поле description, наоборот, по умолчанию записывается `Null`.

Добавим в эту таблицу несколько записей. Структура таблицы spa_treatments и содержащаяся в ней информация показаны на рис. 2.14.

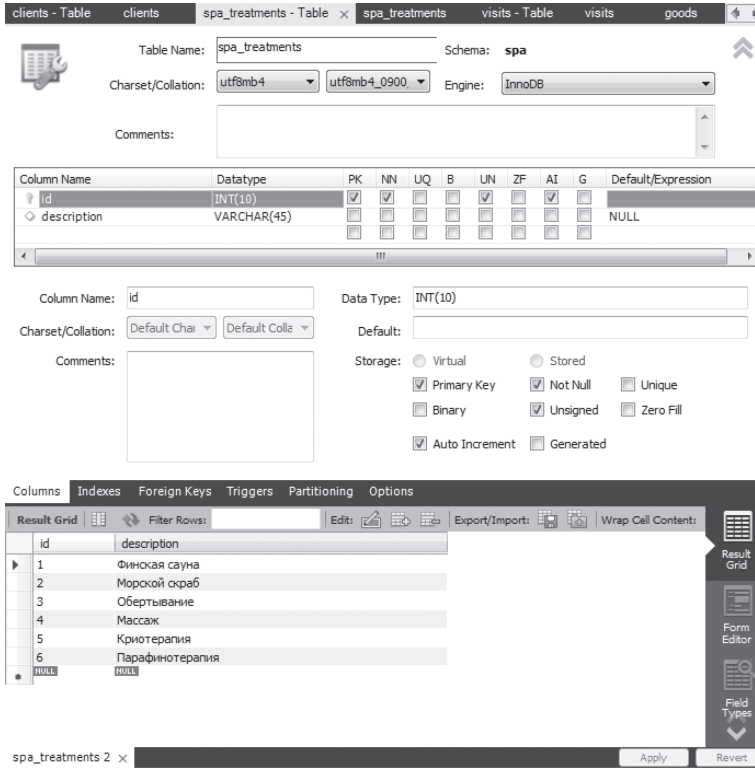


Рис. 2.14. Структура и данные таблицы «spa_treatments»

В схеме spa создадим еще одну таблицу visits. У этой таблицы будут два ключевых поля: client_id типа `int(11)` и spa_treatment_id типа `int(11)` и еще одно поле number типа `int(11)`. У ключевых полей включены свойства `Primary Key`, `Not Null`. Это значит, что поля client_id и spa_treatment_id – это поля первичного ключа и в этих полях не может содержаться значение `Null`. В поле number, наоборот, по умолчанию записывается `Null`.

Добавим в эту таблицу несколько записей. Структура таблицы visits и содержащаяся в ней информация показаны на рис. 2.15.

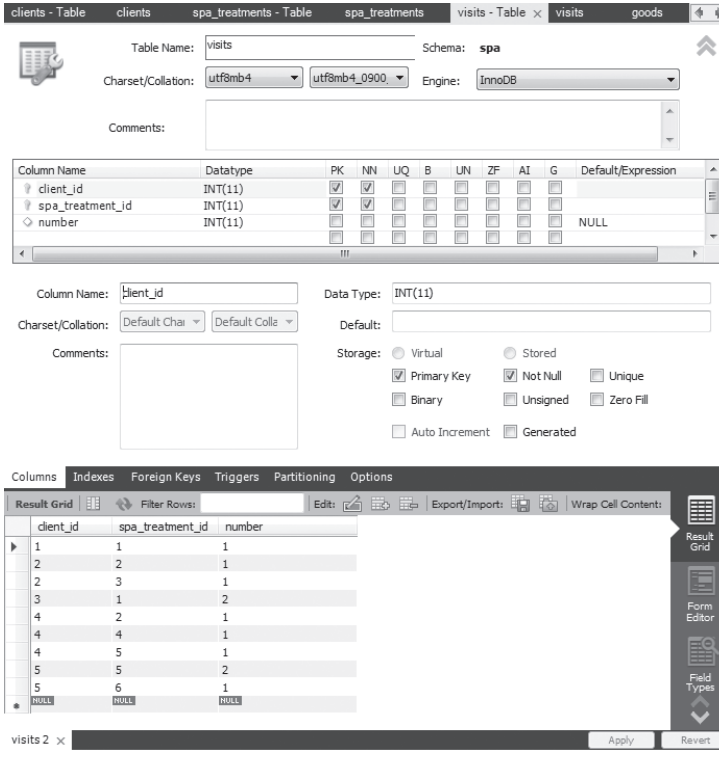


Рис. 2.15. Структура и данные таблицы «visits»

Теперь нам нужно решить, каким образом мы будем соединяться с этими схемами, чтобы получать из них данные в прикладном решении «1С:Предприятия». Мы покажем оба способа: со схемой spa будем соединяться напрямую, с помощью полной строки соединения, а со схемой goods будем соединяться с помощью заранее созданного DSN, о котором речь пойдет в следующем разделе.

DSN

Итак, создадим описание источника данных DSN (Data Source Name), с помощью которого мы будем соединяться со схемой goods.

Для этого мы будем использовать 64-разрядную версию утилиты администрирования ODBC, которая на нашем компьютере (с 64-разрядной версией ОС Windows) расположена в каталоге `C:\Windows\System32\odbcad32.exe`. При этом версия клиентского приложения «1С:Предприятия», которое

будет отправлять запросы к внешнему источнику данных, также должна быть 64-разрядной. В нашем случае мы просто будем запускать «1С:Предприятие» из конфигуратора для 64-разрядной версии ОС Windows.

Для установки соединения нам понадобится также драйвер ODBC для MySQL, который был установлен вместе с самой СУБД. А также мы будем соединяться с помощью веб-сервера, установленного на нашем локальном компьютере («localhost»).

Итак, запустим утилиту администрирования ODBC, перейдем на закладку Системный DSN, нажмем кнопку Добавить... и в списке драйверов выберем MySQL ODBC 8.0 Unicode Driver.

В открывшемся окне зададим произвольное имя и описание источника данных (в нашем случае Goods), в качестве имени сервера укажем localhost, а в качестве имени и пароля пользователя укажем те данные, которые мы задавали при установке СУБД MySQL. В поле Database укажем имя схемы, с которой требуется установить соединение, – goods (рис. 2.16).

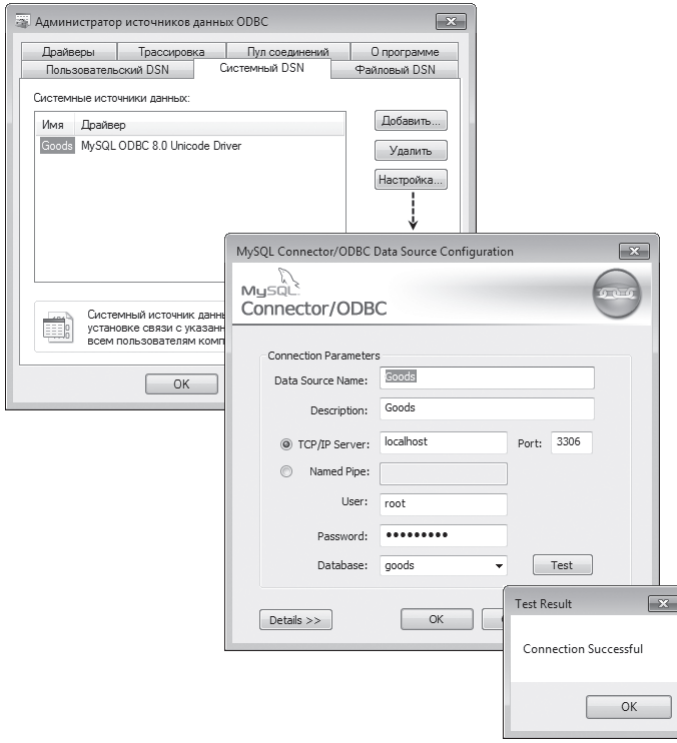


Рис. 2.16. Настройка DSN

В заключение, не покидая окно настройки DSN, можно нажать кнопку Test и убедиться, что соединение с базой данных успешно устанавливается.

ПРИМЕЧАНИЕ

Мы показали пример создания системного DSN, доступного всем пользователям данного компьютера. Точно так же можно создать и пользовательский DSN, но надо понимать, что он будет доступен только тому пользователю, который его настроил.

Работа с внешними источниками данных в конфигураторе и в режиме «1С:Предприятие»

Итак, сначала создадим в конфигураторе объекты для отображения и модификации созданных ранее внешних данных. Для этого предназначены объекты конфигурации `ВнешниеИсточникиДанных`, которые группируются в ветви дерева конфигурации Внешние источники данных. Чтобы не путаться, объекты конфигурации для работы с внешними данными мы будем называть «внешними источниками данных», а собственно хранилища внешних данных (в нашем случае – схемы MySQL) – «внешними схемами» или «внешними таблицами».

Источник данных, подключенный через DSN

Добавим в конфигурацию внешний источник данных с именем Товары. Этот источник будет соединяться со схемой goods с помощью ранее настроенного DSN (см. предыдущий раздел).

Загрузим таблицы внешнего источника данных с помощью конструктора. Для этого в окне редактирования свойств внешнего источника на закладке Данные нажмем кнопку Добавить и выберем пункт Выбрать из списка таблиц внешнего источника данных (рис. 2.17).

В открывшемся окне Подключение к источнику данных укажем строку соединения с внешней схемой:

`DSN`=Goods (регистр символов не имеет значения).

Затем выберем тип используемой СУБД – MySQL (рис. 2.18).

Флажок Использовать аутентификацию операционной системы установлен автоматически. Имя пользователя и пароль, которые использовались при установке СУБД MySQL, здесь можно не задавать, так как они уже указаны в DSN.

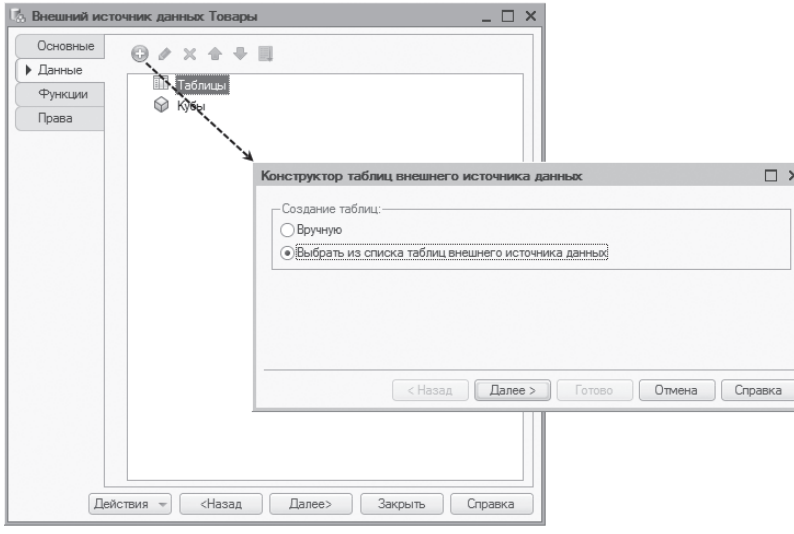


Рис. 2.17. Загрузка таблиц внешнего источника данных с помощью конструктора

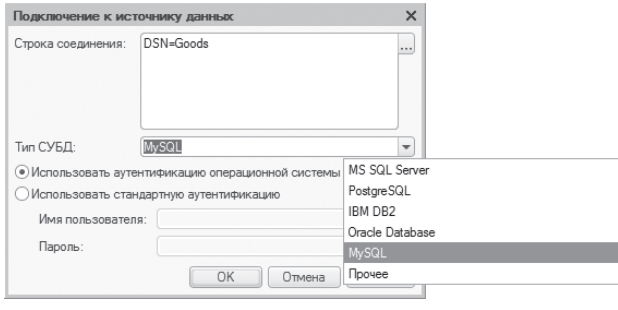


Рис. 2.18. Параметры подключения к внешней схеме

После того как подключение к внешней схеме успешно выполнено, в конструкторе таблиц внешнего источника данных будет открыт список с перечнем таблиц и полей внешней схемы.

Как мы знаем, в базе данных goods содержится единственная таблица goods. Платформа автоматически определила тип таблицы (объектные данные), тип и длину ее полей, определила, что поле id – это поле ключа и включила отметку Разрешить Null только для поля description (рис. 2.19).

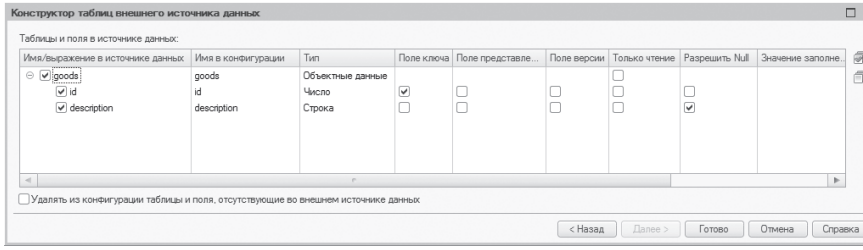


Рис. 2.19. Конструктор таблиц внешнего источника данных «Товары»

Все это нас полностью устраивает. Отметим к переносу во внешний источник данных всю внешнюю таблицу и нажмем Готово. В результате таблица goods будет добавлена в список таблиц внешнего источника данных Товары.

Открыв окно редактирования свойств этой таблицы, можно увидеть ее свойства, установленные в конструкторе, и изменить их (рис. 2.20).

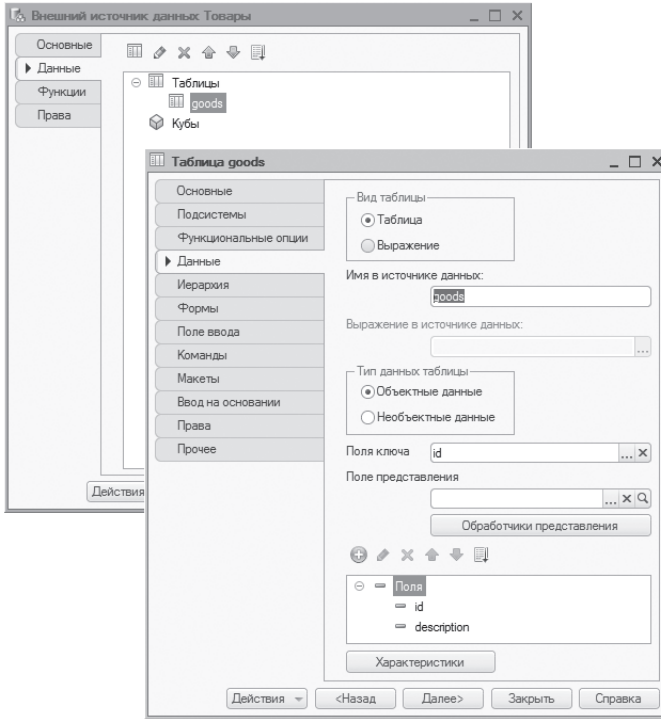


Рис. 2.20. Свойства таблицы «goods»

Источник данных, подключенный с помощью полной строки соединения

Теперь добавим в конфигурацию еще один внешний источник данных с именем Спа. Этот источник будет соединяться с базой данных spa не через DSN, а напрямую, с помощью полной строки соединения.

Загрузим таблицы внешнего источника данных с помощью конструктора. Для этого в окне редактирования свойств этого объекта на закладке Данные нажмем кнопку Добавить и выберем пункт Выбрать из списка таблиц внешнего источника данных.

В открывшемся окне Подключение к источнику данных укажем строку соединения с внешней базой данных:

```
DRIVER=MySQL ODBC 8.0 Unicode Driver;Data Source=localhost;Database=spa
```

Как видно из строки соединения, в качестве драйвера ODBC (`DRIVER`) здесь используется MySQL ODBC 8.0 Unicode Driver, в качестве сервера (`Data Source`) – localhost, а соединение выполняется со схемой (`Database`) spa. Затем выберем тип используемой СУБД – MySQL (рис. 2.21).

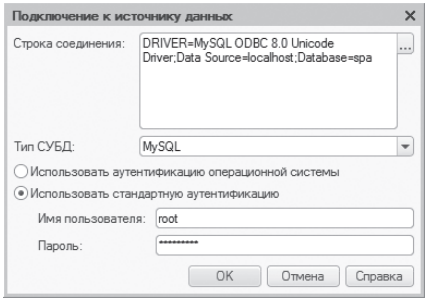


Рис. 2.21. Параметры подключения к внешней схеме

В этом варианте подключения требуется задать имя пользователя и пароль, которые использовались при установке СУБД MySQL. Установим флажок Использовать стандартную аутентификацию и введем имя и пароль пользователя в соответствующие поля под этим флажком.

После того как подключение к внешней схеме успешно выполнено, в конструкторе таблиц внешнего источника данных будет открыт список с перечнем таблиц и полей внешней схемы.

Как мы знаем, в схеме spa содержится три таблицы: clients, spa_treatments, visits. Платформа автоматически определила тип таблиц clients и spa_treatments как объектные данные, а тип таблицы visits как необъектные данные, так как у нее два ключевых поля. Также платформа определила тип и длину простых полей таблиц, правильно установила ключевые поля и включила отметку Разрешить Null в соответствии со свойствами полей в схеме spa (рис. 2.22).

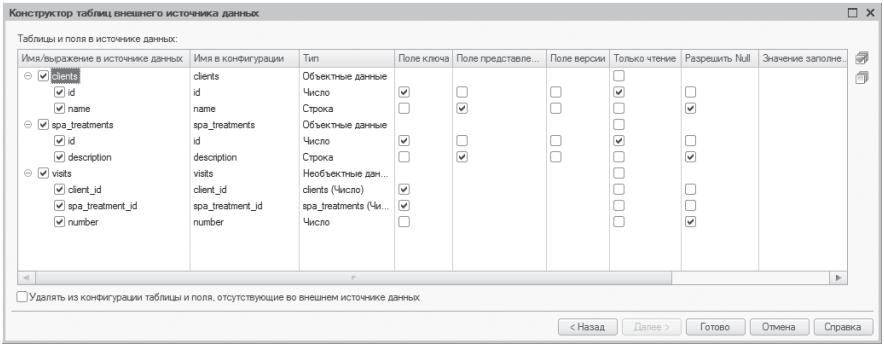


Рис. 2.22. Конструктор таблиц внешнего источника данных «Спа»

Не откладывая на потом, прямо в конструкторе установим, что поле name будет являться полем представления (свойство Поле представления) для таблицы clients, а поле description будет являться полем представления для таблицы spa_treatments.

Укажем, что ключевые поля таблицы visits будут ссылочными. То есть для поля client_id укажем, что оно будет ссылаться на таблицу clients, а поле spa_treatment_id – на таблицу spa_treatments.

Затем включим отметку у свойства Только чтение для поля client_id таблицы clients, так как это ключевое поле является самовозрастающим (`Auto Increment`), т. е. ключ будет автоматически увеличиваться на единицу при добавлении новой записи в таблицу. Поэтому запись в это поле должна быть запрещена. И также по аналогичным причинам включим отметку у свойства Только чтение для поля spa_treatment_id таблицы spa_treatments.

Отметим к переносу во внешний источник данных все три внешние таблицы и нажмем Готово. В результате таблицы clients, spa_treatments, visits будут добавлены в список таблиц внешнего источника данных Спа.

Открыв окно редактирования свойств каждой таблицы, можно увидеть ее свойства, установленные в конструкторе, и изменить их (рис. 2.23, 2.24).

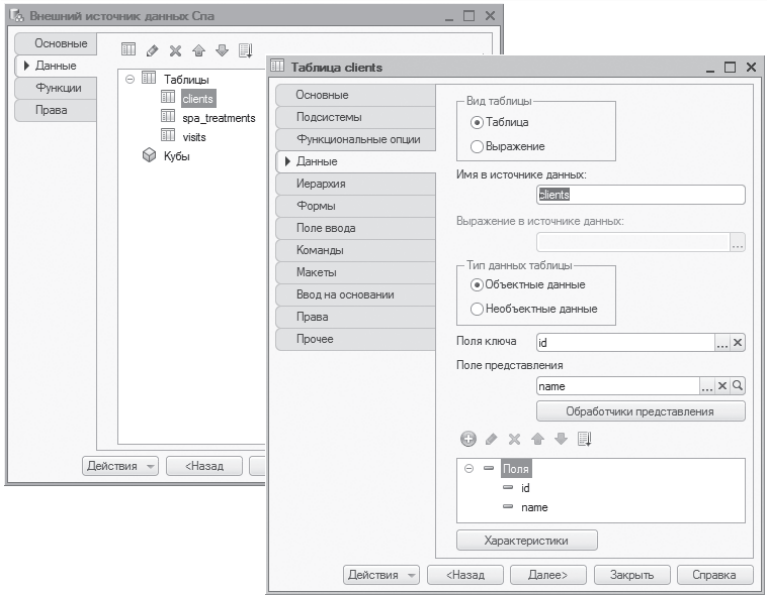


Рис. 2.23. Свойства таблицы «clients»

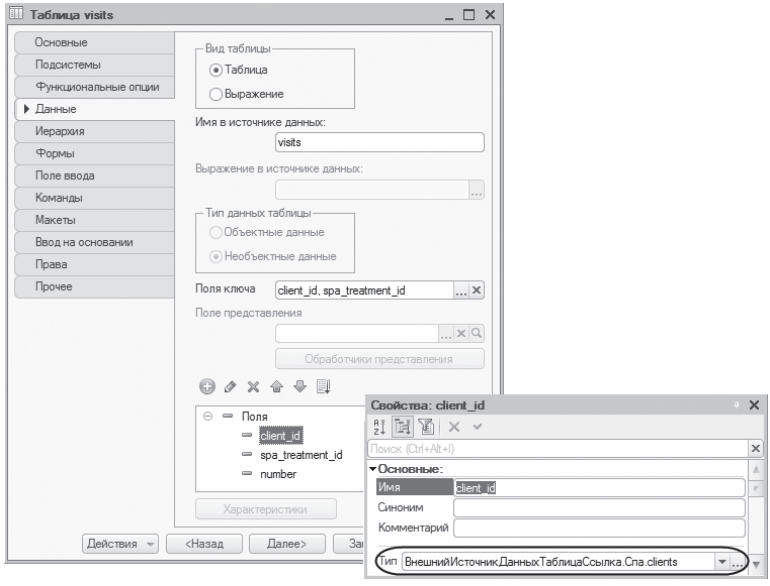


Рис. 2.24. Свойства таблицы «visits»

Подключение и отображение внешних источников данных в режиме «1С:Предприятие»

Теперь проверим, как все это работает. Как видите, мы ничего не программировали. Главное – все правильно настроить, и все «заработает само».

Запустим «1С:Предприятие». В панели функций основного раздела мы видим команды для открытия форм списков таблиц (clients, goods, spa_treatments, visits), которые содержатся во внешних источниках данных (Товары и Спа), описанных в нашей конфигурации.

Нажмем на команду goods для открытия списка товаров. Напомним, что эта таблица принадлежит источнику данных Товары, который подключается к прикладному решению с помощью заранее созданного DSN. Поскольку подключение к этому источнику данных в режиме 1С:Предприятие еще ни разу не выполнялось, откроется окно подключения к источнику данных Товары.

Надо понимать, что, хотя мы уже подключались к этому источнику данных в конфигураторе, это лишь частный случай, когда параметры соединения в режиме «1С:Предприятия» и в конфигураторе совпадают. На самом деле в каждом клиентском приложении настройку соединения нужно выполнять заново. Но делать это нужно только один раз. Впоследствии параметры соединения с источником данных запоминаются и соединение происходит автоматически.

Нажмем на кнопку Изменить общие параметры… и зададим такие же параметры соединения с этим источником данных, как и в конфигураторе (рис. 2.25).

После нажатия на кнопку Подключиться соединение устанавливается и платформа предлагает повторить действие. Обновим данные в списке товаров клавишей F5. В результате мы увидим содержимое таблицы goods, которую мы заполнили в оболочке СУБД MySQL (рис. 2.26).

Обратите внимание, что, хотя мы не создавали никаких форм для наших таблиц, платформа автоматически генерирует для них формы списков, объектов и т. п. Вообще же для таблиц в конфигураторе можно разработать различные собственные формы, которые будут использоваться в прикладном решении. То есть работа с формами таблиц внешнего источника данных аналогична работе с формами других прикладных объектов.

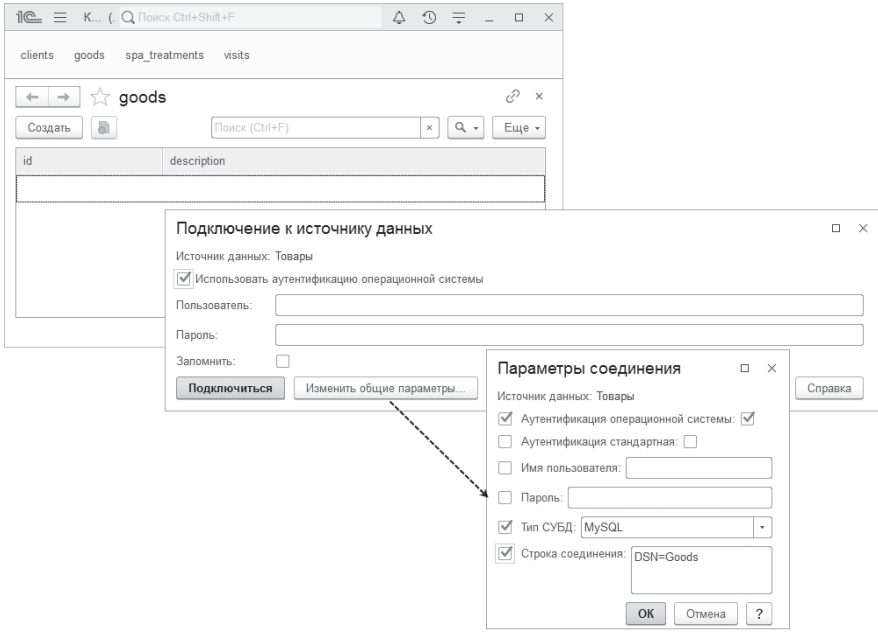


Рис. 2.25. Подключение к источнику данных «Товары»

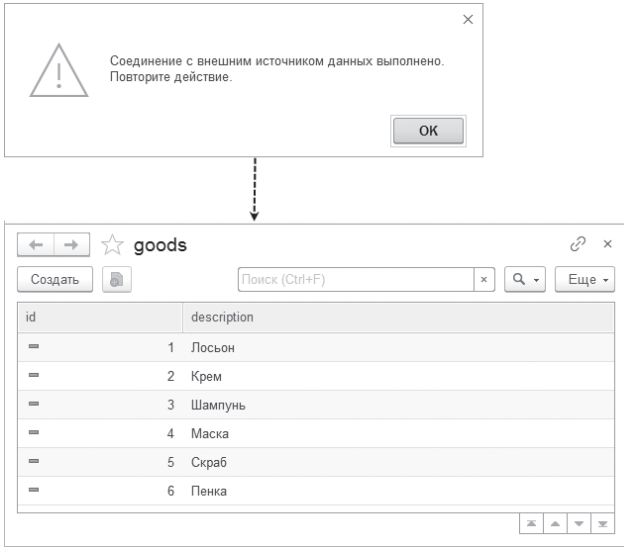


Рис. 2.26. Список товаров

Таким образом, содержимое внешних данных сразу после подключения, без всяких дополнительных усилий, отображается в интерфейсе прикладного решения. Это очень удобно, так как позволяет быстро посмотреть содержимое таблиц и понять, где находятся нужные данные.

Кроме того, разработчик может интерактивно добавить запись во внешний источник, изменить запись в нем или совсем ее удалить. Это тоже может быть очень полезной возможностью для отладки различных алгоритмов обмена данными, получения или импорта внешних данных. Например, если разработчику нужны какие-то элементарные тестовые данные во внешнем источнике, он может ввести их вручную в режиме 1С:Предприятие, ничего при этом не программируя.

Добавим новый товар (с наименованием «Бальзам») в таблицу goods нашего прикладного решения. В результате после обновления данных мы увидим его и в исходной таблице в оболочке СУБД MySQL (рис. 2.27).

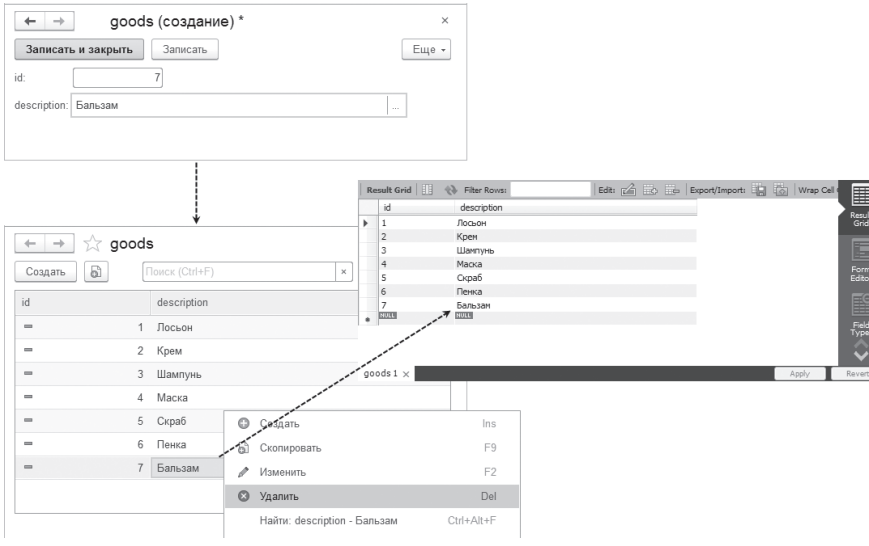


Рис. 2.27. Интерактивное изменение данных во внешней таблице

Необходимо помнить, что во внешних таблицах выполняется только непосредственное удаление записей.

И в обратную сторону все также работает – при добавлении записи в таблицу goods в оболочке СУБД MySQL мы увидим ее в списке товаров в «1С:Предприятии».

Однако возможность интерактивного редактирования, конечно же, не является основной целью использования внешних источников данных в прикладном решении. Как будет показано в следующем разделе, наиболее актуальна задача синхронизации данных между прикладным решением и внешней базой данных.

Теперь из меню Все функции откроем стандартную обработку Управление внешними источниками данных. Мы видим здесь список наших источников данных, причем источник данных Спа еще не подключен. Можно нажать на кнопку Подключиться и соединиться с источником прямо сейчас (рис. 2.28).

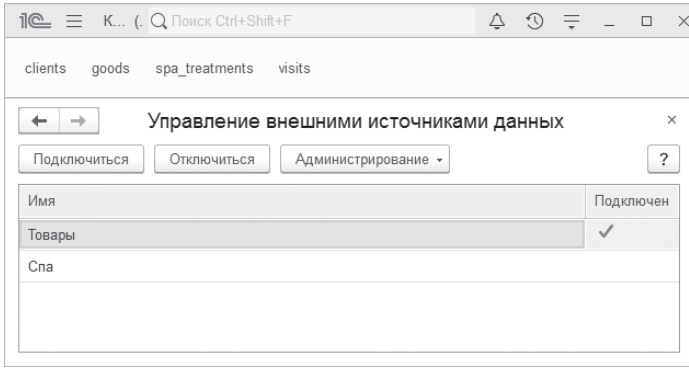


Рис. 2.28. Стандартная обработка «Управление внешними источниками данных»

Выделим внешний источник данных Спа, в открывшемся окне Подключение к источнику данных нажмем на кнопку Изменить общие параметры… и зададим такие же параметры соединения с этим источником данных, как и в конфигураторе. После возврата в окно Подключение к источнику данных нужно продублировать еще раз имя и пароль пользователя (рис. 2.29).

После нажатия на кнопку Подключиться соединение устанавливается. Но можно и не соединяться с источником данных заранее, поскольку при первом обращении к любой из таблиц источника данных подключение будет происходить автоматически с учетом тех параметров соединения, которые мы указали ранее в режиме 1С:Предприятие.

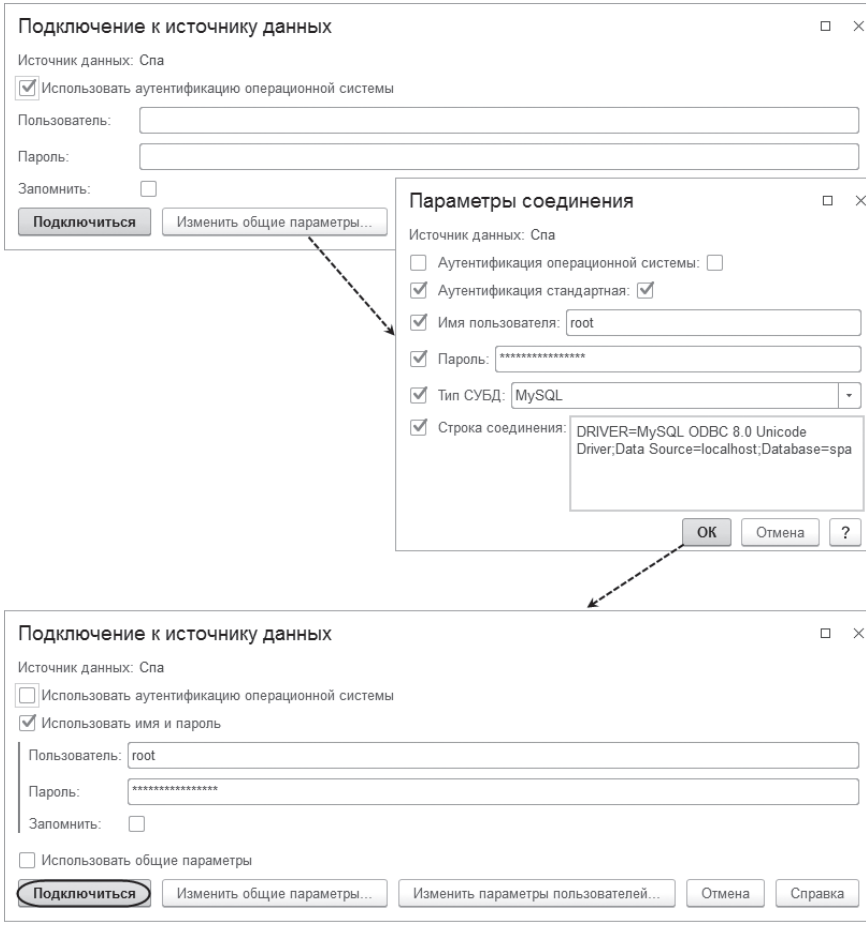


Рис. 2.29. Подключение к источнику данных «Спа»

В заключение посмотрим, как отображается в интерфейсе необъектная таблица visits. Напомним, что у этой таблицы два ссылочных поля: поле client_id ссылается на таблицу clients и поле spa_treatment_id ссылается на таблицу spa_treatments. А у этих двух объектных таблиц (clients, spa_treatments) полями представления являются соответственно name и description. Поэтому в таблице visits мы видим вместо ключевых полей фамилии клиентов и наименования косметических процедур, которые были им выполнены (рис. 2.30).

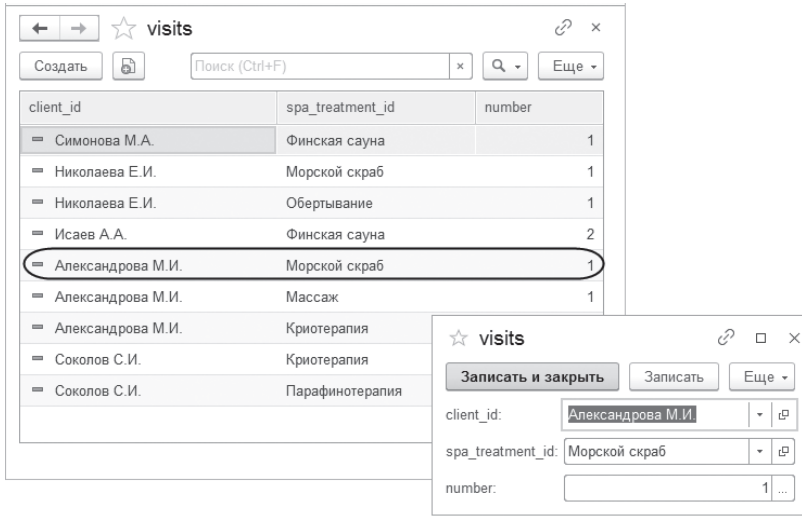


Рис. 2.30. Таблица «visits»

Программная синхронизация

В этом разделе мы рассмотрим примеры синхронизации данных между структурами хранения информации «1С:Предприятия» и внешними данными.

При работе с внешними источниками данных из встроенного языка есть возможность не только получать, но и записывать данные в эти источники. Ниже мы рассмотрим примеры заполнения справочника внешними данными, и, наоборот, при добавлении или изменении данных в прикладном решении эти же данные будут записываться и во внешнюю таблицу.

Таким образом, с помощью внешних источников данных могут решаться различные задачи интеграции прикладных решений «1С:Предприятия» с другими информационными системами.

Предположим, в нашей конфигурации есть справочник Товары, который должен программно пополняться новыми товарами, которые появляются в таблице goods внешней схемы goods. А при добавлении записей в справочник товаров нужно записывать их и во внешнюю таблицу.

Для решения этой задачи нам нужно добавить в конфигурацию две общие команды: ЗагрузитьТоварыИзВнешнегоИсточника и ВыгрузитьТоварыВоВнешнийИсточник. Первая команда будет читать новые товары, если они появились во внешней таблице, и записывать их в справочник. Вторая команда будет записывать новые товары, если они появились в справочнике, во внешнюю таблицу.

На самом деле алгоритм программной синхронизации данных может вызываться, например, регламентным заданием, которое запускается по расписанию, но мы для упрощения примера пока используем команды.

Связь со схемой goods устанавливается через внешний источник данных Товары, который мы описали в разделе «Источник данных, подключенный через DSN». Список товаров отображается в таблице goods этого источника данных.

Итак, добавим в нашу конфигурацию общую команду ЗагрузитьТоварыИзВнешнегоИсточника. Из обработчика этой команды будет вызываться процедура `ЗагрузитьТоварыИзВнешнегоИсточника()`. Заполним ее следующим образом (листинг 2.1).

Листинг 2.1. Процедура «ЗагрузитьТоварыИзВнешнегоИсточника»

```
&НаСервере
Процедура ЗагрузитьТоварыИзВнешнегоИсточника()

	// Создать менеджер справочника Товары.
	Товары = Справочники.Товары;

	// Получить список товаров с помощью запроса к таблице goods внешнего источника данных Товары.
	Запрос = Новый Запрос;
	Запрос.Текст =
		"ВЫБРАТЬ
		|	Товары.id КАК Товар,
		|	Товары.description КАК Наименование
		|ИЗ
		|	ВнешнийИсточникДанных.Товары.Таблица.goods КАК Товары
		|
		|УПОРЯДОЧИТЬ ПО
		|	Товар";

	РезультатЗапроса = Запрос.Выполнить();

	Выборка = РезультатЗапроса.Выбрать();

	// Добавить товары из таблицы goods в справочник Товары, если их там еще нет.
	Пока Выборка.Следующий() Цикл
		ТоварСсылка = Товары.НайтиПоНаименованию(Выборка.Наименование, Истина);
		Если ТоварСсылка = Товары.ПустаяСсылка() ИЛИ ТоварСсылка = Неопределено Тогда
			Товар = Товары.СоздатьЭлемент();
		Иначе
			Товар = ТоварСсылка.ПолучитьОбъект();
		КонецЕсли;

		Товар.Наименование = Выборка.Наименование;
		Товар.Записать();
	КонецЦикла;

КонецПроцедуры
```

В этой процедуре мы сначала создаем менеджер для обращения к справочнику Товары. Затем с помощью запроса к таблице goods внешнего источника данных Товары мы получаем список товаров, находящихся во внешней таблице. Таблица внешнего источника данных описывается в языке запросов как `ВнешнийИсточникДанных.<Имя источника>.Таблица.<Имя таблицы>`. Остальные конструкции в тексте запроса нам хорошо знакомы и не требуют пояснений.

По мере обхода выборки из результата запроса мы добавляем отсутствующие товары из таблицы goods в справочник Товары. Поиск товара в справочнике происходит по полному соответствию наименования товара в справочнике и во внешней таблице. Если же товар уже существует в справочнике, то от найденной ссылки на товар получается объект и данные этого товара синхронизируются между внешней таблицей и справочником.

Проверим, как это работает. Запустим «1С:Предприятие» и из меню Сервис выполним команду Загрузить товары из внешнего источника. В результате в справочник Товары (который пока пуст) будут скопированы все товары из таблицы goods внешнего источника данных (рис. 2.31).

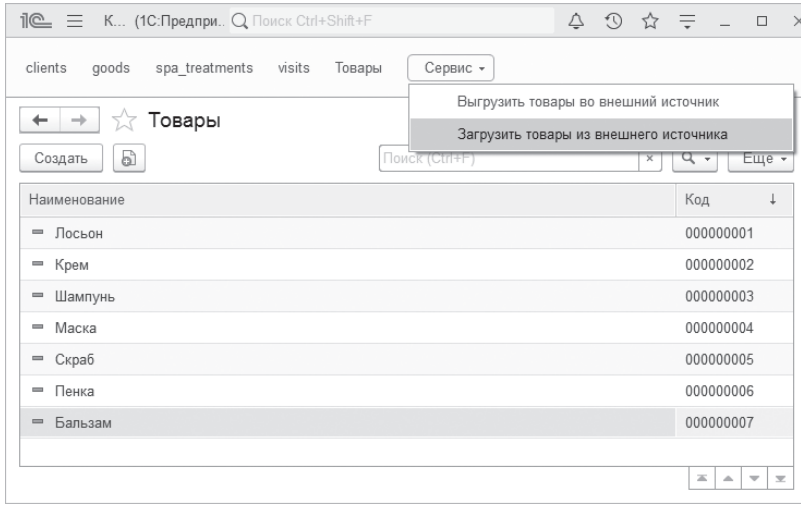


Рис. 2.31. Программная синхронизация данных между справочником и внешним источником

Теперь создадим новый товар (с наименованием «Кондиционер») в списке товаров внешнего источника данных. После этого выполним еще раз команду Загрузить товары из внешнего источника. В результате новый товар появится в справочнике Товары (рис. 2.32).

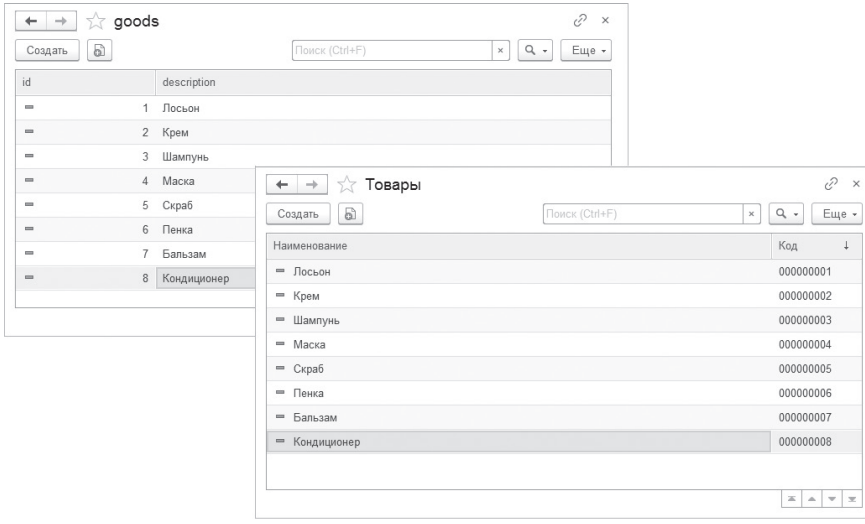


Рис. 2.32. Программная синхронизация данных между справочником и внешним источником

ПРИМЕЧАНИЕ

Чтобы увидеть программно добавленные записи, нужно обновить данные в списках клавишей F5.

Теперь реализуем синхронизацию данных в обратную сторону – от справочника к внешнему источнику.

Итак, добавим в нашу конфигурацию общую команду ВыгрузитьТоварыВоВнешнийИсточник. Из обработчика этой команды будет вызываться процедура `ВыгрузитьТоварыВоВнешнийИсточник()`. Заполним ее следующим образом (листинг 2.2).

Листинг 2.2. Процедура «ВыгрузитьТоварыВоВнешнийИсточник»

```
&НаСервере
Процедура ВыгрузитьТоварыВоВнешнийИсточник()

	// Создать менеджер таблицы goods внешнего источника данных Товары.
	Товары = ВнешниеИсточникиДанных.Товары.Таблицы.goods;

	// Получить список товаров с помощью запроса к справочнику Товары.
	Запрос = Новый Запрос;
	Запрос.Текст =
		"ВЫБРАТЬ
		|	Товары.Код КАК Код,
		|	Товары.Наименование КАК Наименование
		|ИЗ
		|	Справочник.Товары КАК Товары
		|
		|УПОРЯДОЧИТЬ ПО
		|	Код";

	РезультатЗапроса = Запрос.Выполнить();

	Выборка = РезультатЗапроса.Выбрать();

	// Добавить товары из справочника Товары во внешнюю таблицу goods, если их там еще нет.
	Пока Выборка.Следующий() Цикл
		ТоварСсылка = Товары.НайтиПоПолю("description", Выборка.Наименование);
		Если ТоварСсылка = Товары.ПустаяСсылка() ИЛИ ТоварСсылка = Неопределено Тогда
			Товар = Товары.СоздатьОбъект();
		Иначе
			Товар = ТоварСсылка.ПолучитьОбъект();
		КонецЕсли;

		Товар.id = Число(Выборка.Код);
		Товар.description = Выборка.Наименование;
		Товар.Записать();
	КонецЦикла;

КонецПроцедуры
```

В этой процедуре мы сначала создаем менеджер таблицы goods внешнего источника данных Товары (`ВнешниеИсточникиДанных.Товары.Таблицы.goods`).

Затем с помощью запроса к справочнику Товары мы получаем список товаров, который нужно синхронизировать со списком товаров во внешней таблице.

По мере обхода выборки из результата запроса мы добавляем отсутствующие товары из справочника Товары в таблицу goods. Поиск товара во внешней таблице происходит методом менеджера таблицы `НайтиПоПолю()` по полному соответствию наименования товара в таблице goods (поле description) и в справочнике.

Если такого товара в таблице еще нет, то он добавляется методом менеджера таблицы `СоздатьОбъект()`. Если есть, то от найденной ссылки на товар получается объект и данные этого товара синхронизируются между внешней таблицей и справочником.

Запустим «1С:Предприятие», откроем справочник Товары и добавим новый элемент. Из меню Сервис выполним команду Выгрузить товары во внешний источник. В результате новый товар появится в таблице goods внешнего источника данных (рис. 2.33).

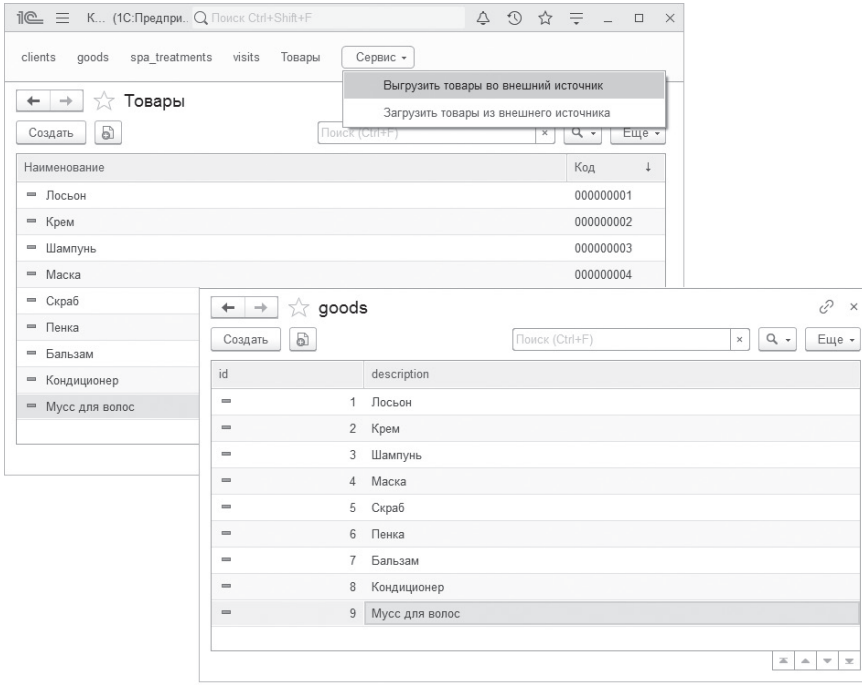


Рис. 2.33. Программная синхронизация данных между справочником и внешним источником

Работа с функциями

В этом разделе мы покажем примеры использования функций внешних источников данных. Причем можно использовать как реально существующую во внешней базе данных функцию, так и написать собственную функцию (на языке используемой СУБД), которая отсутствует в реальной базе данных.

Для примера рассмотрим такую задачу: предположим, что во внешней таблице существуют определенные правила, по которым создается ключ новой записи. Эти правила описываются функцией (внешней или хранимой в СУБД). Нам нужно обеспечить возможность ввода нового элемента в прикладное решение «1С:Предприятия» и синхронного создания такого же элемента во внешней таблице. Но при этом его ключ должен создаваться по этим принятым правилам. Для простоты будем считать, что правило заключается в том, чтобы увеличить значение кода на единицу, но в реальности это может быть более сложное правило, зависящее от алгоритмов внешнего источника.

Реализуем эту задачу двумя способами: сначала самостоятельно создадим нужную функцию во внешнем источнике данных в конфигураторе, а потом импортируем ту функцию, которая существует во внешней схеме, и используем ее.

Для простоты вызывать эти функции будем так же, как и в предыдущем разделе, с помощью общих команд. Показанные ниже примеры являются просто демонстрационными и не имеют никакого прикладного значения.

Собственная функция

Предположим, при программном добавлении косметической процедуры в таблицу spa_treatments внешнего источника данных нам нужно также заполнять ключевое поле этой таблицы автоматически возрастающим значением.

Для этого мы будем использовать собственную функцию, которая возвращает максимальное значение ключевого поля в таблице spa_treatments схемы spa, а затем увеличивать это значение на единицу и записывать в ключевое поле таблицы.

Связь со схемой spa устанавливается через внешний источник данных Спа, который мы описали в разделе «Источник данных, подключенный с помощью полной строки соединения». Список косметических процедур отображается в таблице spa_treatments этого источника данных.

Откроем окно редактирования источника данных Спа, содержащего эту таблицу, и на закладке Функции нажмем кнопку Добавить. В появившемся окне конструктора функций внешнего источника данных выберем пункт Вручную и нажмем Готово. Дадим функции имя ПолучитьМаксимальныйКлючТаблицы, укажем, что она возвращает значение типа `Число (10, 0)`, а в поле Выражение в источнике данных напишем выражение: `MAX(id) FROM spa.spa_treatments` (рис. 2.34).

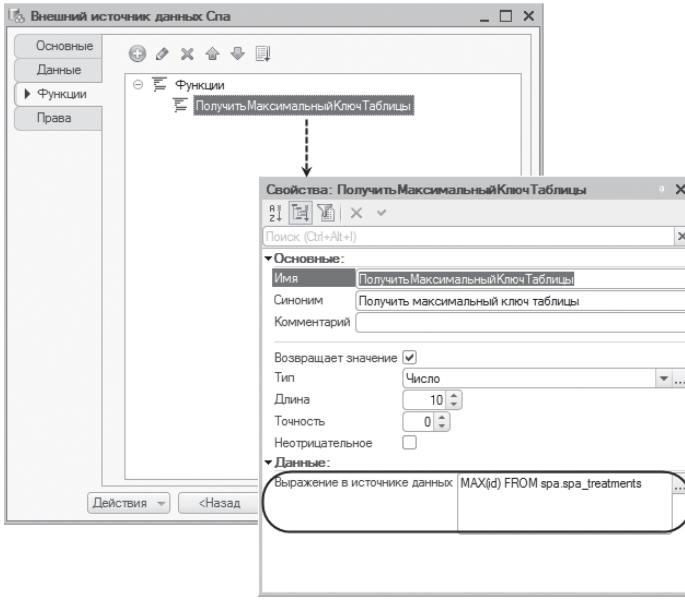


Рис. 2.34. Добавление собственной функции внешнего источника данных

Надо понимать, что при использовании функции во встроенном языке выражение, указанное в свойстве Выражение в источнике данных, будет использоваться в реальном запросе к базе данных. Это выражение непосредственно подставляется в конструкцию `select <...>`, поэтому слово «select» писать уже не нужно.

Теперь добавим в нашу конфигурацию еще одну общую команду ДобавитьСпаПроцедуру. Обработчик команды заполним следующим образом (листинг 2.3).

Листинг 2.3. Обработчик команды «ДобавитьСпаПроцедуру»

```
&НаКлиенте
Процедура ОбработкаКоманды(ПараметрКоманды, ПараметрыВыполненияКоманды)

    Оповещение = Новый ОписаниеОповещения("ПослеВводаСтроки", ЭтотОбъект);
    ПоказатьВводСтроки(Оповещение, "Введите имя процедуры");

КонецПроцедуры
```

Ввод наименования новой спа-процедуры мы будем выполнять с помощью немодального метода `ПоказатьВводСтроки()`. В этот метод первым параметром мы передаем описание оповещения, указывающее на экспортную процедуру `ПослеВводаСтроки()`, которая будет вызвана по окончании ввода наименования спа-процедуры (листинг 2.4).

Листинг 2.4. Процедура «ПослеВводаСтроки»

```
&НаКлиенте
Процедура ПослеВводаСтроки(Строка, Параметры) Экспорт

    Если НЕ Строка = Неопределено Тогда
        ДобавитьПроцедуруВоВнешнийИсточник(Строка);
    КонецЕсли;

КонецПроцедуры
```

В этом обработчике оповещения, в случае если строка с наименованием спа-процедуры (содержащаяся в параметре `Строка`) задана, мы вызываем процедуру `ДобавитьПроцедуруВоВнешнийИсточник()`, в которую передаем введенное наименование (листинг 2.5).

Листинг 2.5. Процедура «ДобавитьПроцедуруВоВнешнийИсточник»

```
&НаСервере
Процедура ДобавитьПроцедуруВоВнешнийИсточник(СтрокаПроцедуры)

    // Получить значение ключа с помощью функции ПолучитьМаксимальныйКлючТаблицы()
    // внешнего источника данных Спа.
    ЗначениеКлюча = ВнешниеИсточникиДанных.Спа.ПолучитьМаксимальныйКлючТаблицы() + 1;

    // Создать менеджер таблицы spa_treatments внешнего источника данных Спа.
    Процедуры = ВнешниеИсточникиДанных.Спа.Таблицы.spa_treatments;

    // Найти запись в таблице spa_treatments по наименованию добавляемой процедуры.
    Ссылка = Процедуры.НайтиПоПолю("description", СтрокаПроцедуры);

    // Добавить новую процедуру в таблицу spa_treatments, если ее там еще нет.
    Если Ссылка = Процедуры.ПустаяСсылка() ИЛИ Ссылка = Неопределено Тогда
        НоваяПроцедура = Процедуры.СоздатьОбъект();
```

```
            // Получить и установить изменяемые поля таблицы.
            ИзменяемыеПоля = НоваяПроцедура.ПолучитьИзменяемыеПоля();
            Если ИзменяемыеПоля.Найти("id") = Неопределено Тогда
                  // Включить ключевое поле в состав изменяемых полей для новой записи.
                  ИзменяемыеПоля.Добавить("id");
                  НоваяПроцедура.УстановитьИзменяемыеПоля(ИзменяемыеПоля);
            КонецЕсли;

            // Записать значение ключа в ключевое поле.
            НоваяПроцедура.id = ЗначениеКлюча;
            НоваяПроцедура.description = СтрокаПроцедуры;
            НоваяПроцедура.Записать();
      КонецЕсли;

КонецПроцедуры
```

В этой процедуре сначала мы получаем максимальное значение ключевого поля id в таблице spa_treatments с помощью функции `ПолучитьМаксимальныйКлючТаблицы()` внешнего источника данных Спа. И увеличиваем это значение на единицу.

Затем создаем менеджер таблицы spa_treatments внешнего источника данных Спа (`ВнешниеИсточникиДанных.Спа.Таблицы.spa _ treatments`). Затем методом этого менеджера `НайтиПоПолю()` ищем запись в таблице spa_treatments, у которой в поле description содержится точное соответствие наименованию введенной процедуры (`СтрокаПроцедуры`).

Если такой процедуры в таблице еще нет, то она добавляется методом менеджера таблицы `СоздатьОбъект()`. При создании новой процедуры наряду с наименованием мы записываем получившееся новое значение ключа в ключевое поле (`НоваяПроцедура.id = ЗначениеКлюча`).

Но сначала мы должны включить ключевое поле в состав изменяемых полей для новой записи. Если этого не сделать, то при записи новой процедуры будет получена ошибка (рис. 2.35).

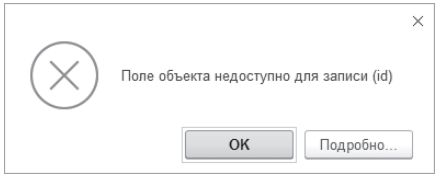


Рис. 2.35. Ошибка при записи в ключевое поле таблицы «spa_treatments»

Так произошло потому, что для ключевого поля id в таблице spa_treatments мы установили свойство Только чтение, так как это поле в исходной таблице внешней схемы определено как самовозрастающее.

Поэтому теперь, прежде чем записывать значение в это поле, нужно включить его в состав изменяемых полей таблицы. Для этого нужно получить изменяемые поля методом `ПолучитьИзменяемыеПоля()` для новой записи таблицы, добавить имя поля id в массив изменяемых полей и передать этот массив в метод `УстановитьИзменяемыеПоля()`.

Теперь все должно быть хорошо. Запустим «1С:Предприятие» и из меню Сервис выполним команду Добавить спа процедуру. В появившемся окне введем наименование новой процедуры и нажмем OK. В результате введенная процедура со следующим по порядку значением ключевого поля появится в таблице spa_treatments внешнего источника данных (рис. 2.36).

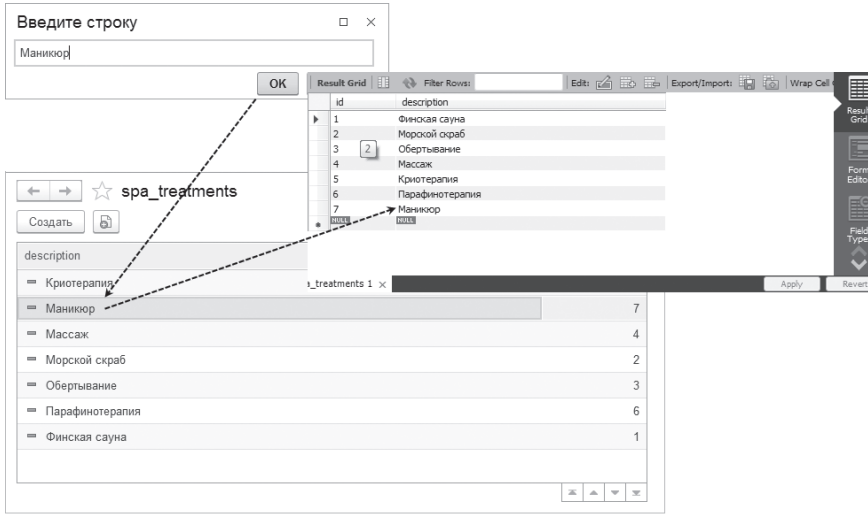


Рис. 2.36. Добавление процедуры в таблицу внешнего источника данных

Функция внешней схемы

Теперь покажем, как использовать уже существующую во внешней схеме функцию в прикладном решении «1С:Предприятия».

Предположим, в схеме spa существует функция `get _ next _ key _ record()`, которая возвращает ключ следующей записи из таблицы spa_treatments.

Нам нужно, так же как и в предыдущем примере, программно добавить косметическую процедуру во внешнюю таблицу и заполнить ее ключ с помощью функции `get _ next _ key _ record()`.

Сначала загрузим функции внешнего источника данных Спа, к которому относится таблица spa_treatments, с помощью конструктора. Для этого в окне редактирования свойств этого объекта на закладке Функции нажмем кнопку Добавить и выберем пункт Выбрать из списка функций внешнего источника данных. Ранее (при загрузке списка таблиц) мы уже настраивали в конфигураторе соединение с этим источником данных. Поэтому в открывшемся окне Подключение к источнику данных подтвердим те же параметры соединения и нажмем ОК.

После установки соединения в конструкторе функций внешнего источника данных будет открыт список функций внешней схемы spa. Отметим к переносу функцию spa.get_next_key_record() и нажмем Готово (рис. 2.37).

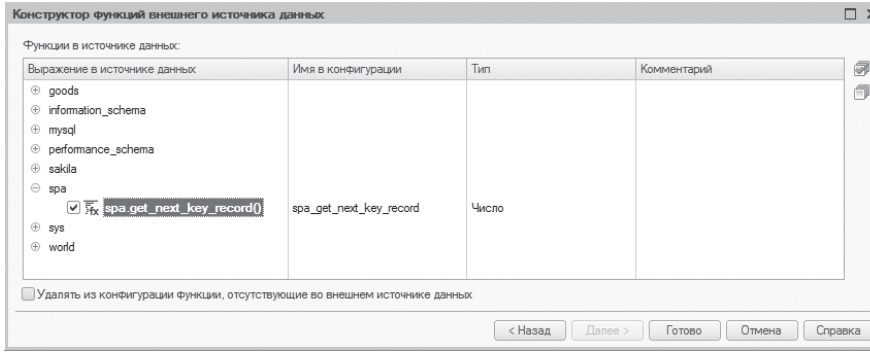


Рис. 2.37. Конструктор функций внешнего источника данных «Спа»

В результате функция с именем spa_get_next_key_record будет добавлена в список функций внешнего источника данных Спа. Открыв палитру свойств этой функции, можно увидеть ее свойства, установленные в конструкторе, и изменить их.

Свойство Выражение в источнике данных платформа определила правильно – `spa.get _ next _ key _ record()`. Укажем также, что функция возвращает значение типа `Число (10, 0)` (рис. 2.38).

Теперь добавим в нашу конфигурацию еще одну общую команду ДобавитьКосметическуюПроцедуру. Обработчик команды заполним следующим образом (листинг 2.6).

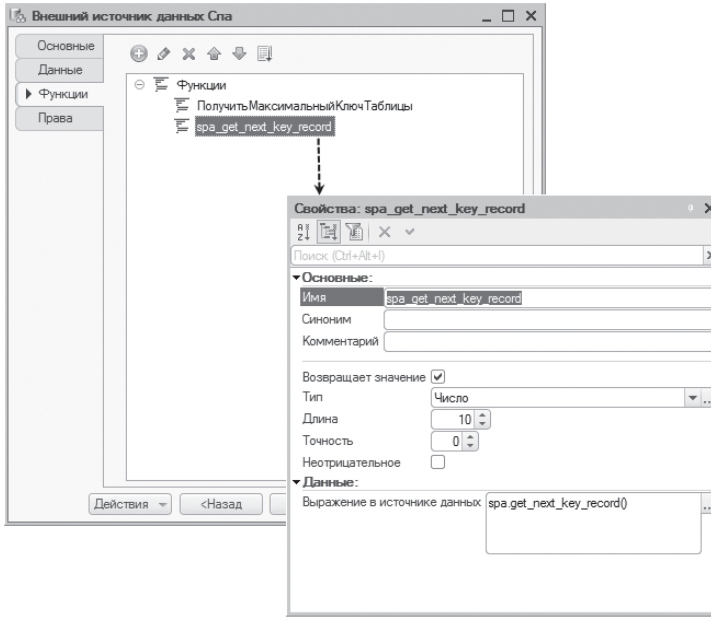


Рис. 2.38. Свойства функции «spa_get_next_key_record»

Листинг 2.6. Обработчик команды «ДобавитьКосметическуюПроцедуру»

```
&НаКлиенте
Процедура ОбработкаКоманды(ПараметрКоманды, ПараметрыВыполненияКоманды)

    Оповещение = Новый ОписаниеОповещения("ПослеВводаСтроки", ЭтотОбъект);
    ПоказатьВводСтроки(Оповещение, "Введите имя процедуры");

КонецПроцедуры
```

Ввод наименования новой спа-процедуры мы будем выполнять с помощью немодального метода `ПоказатьВводСтроки()`. В этот метод первым параметром мы передаем описание оповещения, указывающее на экспортную процедуру `ПослеВводаСтроки()`, которая будет вызвана по окончании ввода наименования спа-процедуры (листинг 2.7).

Листинг 2.7. Процедура «ПослеВводаСтроки»

```
&НаКлиенте
Процедура ПослеВводаСтроки(Строка, Параметры) Экспорт

    Если НЕ Строка = Неопределено Тогда
        ДобавитьПроцедуруВоВнешнийИсточник(Строка);
    КонецЕсли;

КонецПроцедуры
```

В этом обработчике оповещения, в случае если строка с наименованием спа-процедуры (содержащаяся в параметре `Строка`) задана, мы вызываем процедуру `ДобавитьПроцедуруВоВнешнийИсточник()`, в которую передаем введенное наименование (листинг 2.8).

Листинг 2.8. Процедура «ДобавитьПроцедуруВоВнешнийИсточник»

```
&НаСервере
Процедура ДобавитьПроцедуруВоВнешнийИсточник(СтрокаПроцедуры)

    // Получить значение ключа с помощью функции spa_get_next_key_record()
    // внешнего источника данных Спа.
    ЗначениеКлюча = ВнешниеИсточникиДанных.Спа.spa_get_next_key_record();

    // Создать менеджер таблицы spa_treatments внешнего источника данных Спа.
    Процедуры = ВнешниеИсточникиДанных.Спа.Таблицы.spa_treatments;

    // Найти запись в таблице spa_treatments по наименованию добавляемой процедуры.
    Ссылка = Процедуры.НайтиПоПолю("description", СтрокаПроцедуры);

    // Добавить новую процедуру в таблицу spa_treatments, если ее там еще нет.
    Если Ссылка = Процедуры.ПустаяСсылка() ИЛИ Ссылка = Неопределено Тогда
        НоваяПроцедура = Процедуры.СоздатьОбъект();

        // Получить и установить изменяемые поля таблицы.
        ИзменяемыеПоля = НоваяПроцедура.ПолучитьИзменяемыеПоля();
        Если ИзменяемыеПоля.Найти("id") = Неопределено Тогда
            // Включить ключевое поле в состав изменяемых полей для новой записи.
            ИзменяемыеПоля.Добавить("id");
            НоваяПроцедура.УстановитьИзменяемыеПоля(ИзменяемыеПоля);
        КонецЕсли;

        // Записать значение ключа в ключевое поле.
        НоваяПроцедура.id = ЗначениеКлюча;
        НоваяПроцедура.description = СтрокаПроцедуры;
        НоваяПроцедура.Записать();
    КонецЕсли;

КонецПроцедуры
```

Прокомментируем только строку для получения значения ключа следующей записи внешней таблицы spa_treatments, которая выделена жирным шрифтом, так как все остальное объяснялось в предыдущем примере (см. листинг 2.5). Значение ключа получается с помощью функции `spa _ get _ next _ key _ record()` внешнего источника данных Спа.

Проверим, как это работает. Запустим «1С:Предприятие» и из меню Сервис выполним команду Добавить косметическую процедуру. В появившемся окне введем наименование новой процедуры и нажмем ОК. В результате введенная процедура со следующим по порядку значением ключевого поля появится в таблице spa_treatments внешнего источника данных (рис. 2.39).

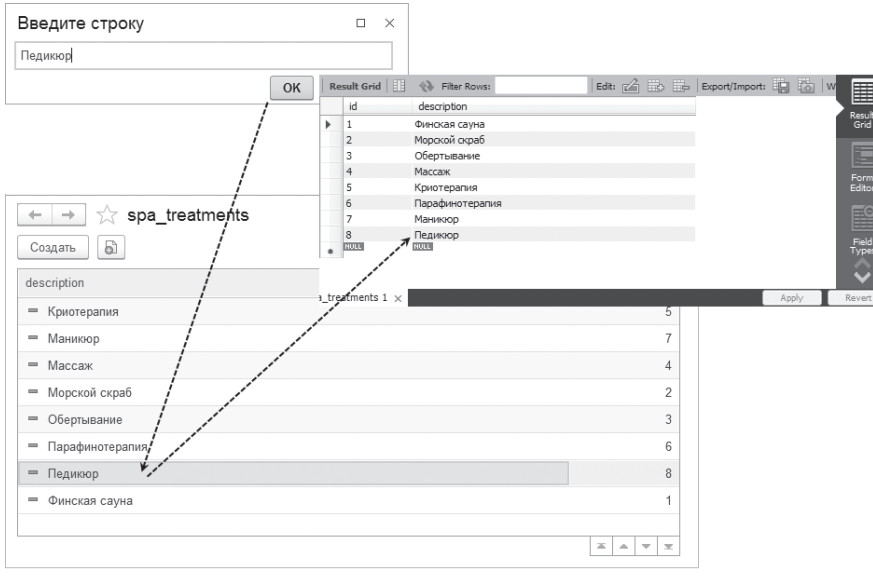


Рис. 2.39. Добавление процедуры в таблицу внешнего источника данных

Прикладное использование данных из внешних источников

Учет

Кроме примеров, описанных выше, данные из внешних источников могут использоваться также и в прикладных целях. Например, во время разработки и тестирования каких-то новых механизмов учета разработчику требуется подстраховать себя. То есть нужно «незаметно» для пользователя продублировать данные, введенные им во внешнюю таблицу, в какую-то структуру хранения данных «1С:Предприятия», чтобы потом проанализировать эти данные.

Например, чтобы не заставлять пользователя дважды вводить данные о посещении клиентами спа-салона и выполненных им процедурах, «приспособим» для этого внешнюю необъектную таблицу visits внешнего источника данных Спа. Дадим таблице синоним – Визиты, а ее полям client_id, spa_treatment_id и number дадим соответственно синонимы Клиент, Спа-процедура и Количество. Для поля number установим также свойство Значение заполнения в значение 1 (рис. 2.40).

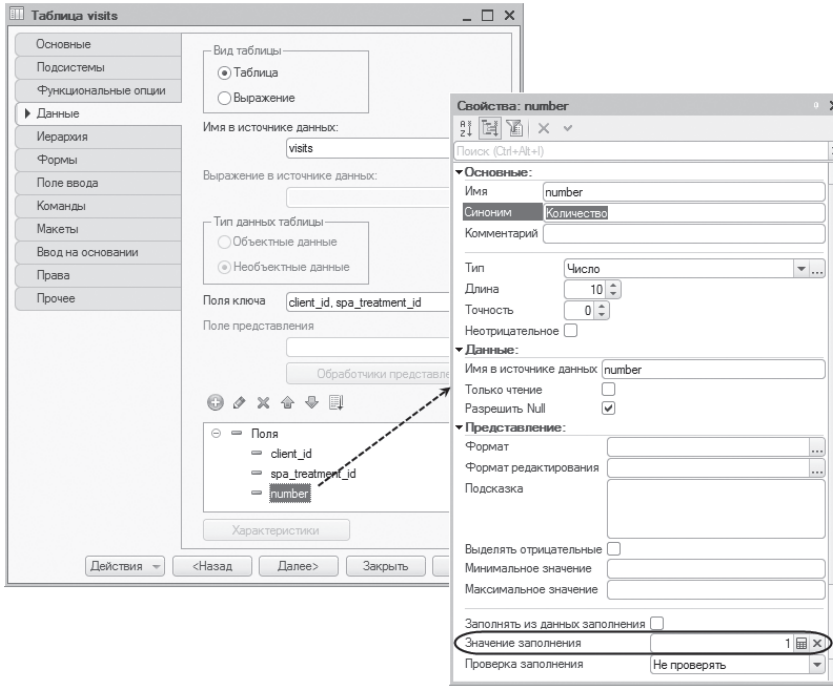


Рис. 2.40. Свойства таблицы «visits»

Ранее, при настройке таблицы visits в конфигураторе, мы установили, что у этой таблицы два ссылочных поля: поле client_id ссылается на таблицу clients и поле spa_treatment_id ссылается на таблицу spa_treatments. А у этих двух объектных таблиц (clients, spa_treatments) полями представления являются соответственно name и description.

Запустим «1С:Предприятие». Таблица visits теперь имеет привычный вид. Вместо ключевых полей пользователь увидит фамилии клиентов и наименования косметических процедур, которые были им выполнены. Кроме того, у самой таблицы и у ее полей теперь вполне понятные названия, и у пользователя может возникнуть ощущение, что он работает с привычным регистром сведений, например (рис. 2.41).

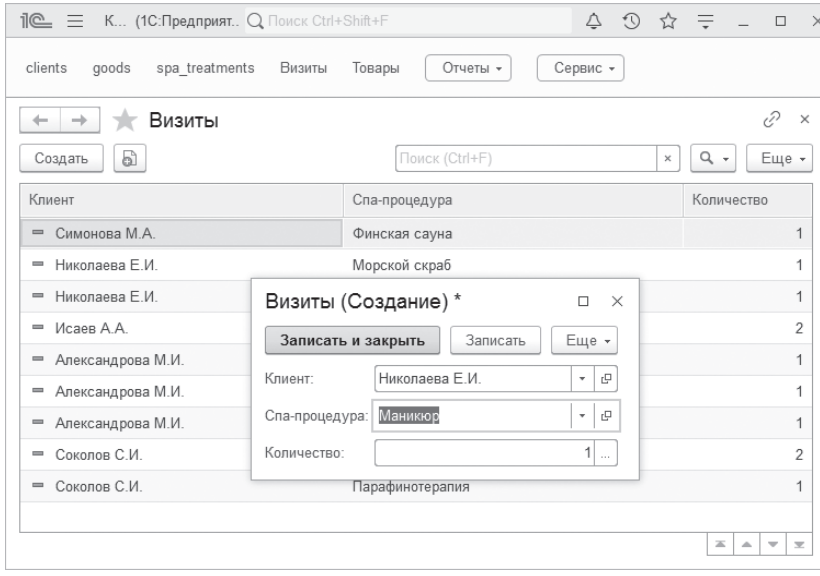


Рис. 2.41. Ввод данных о посещениях клиентами спа-салона во внешнюю таблицу «visits»

На самом деле пользователь будет вводить данные непосредственно во внешнюю таблицу, а мы добавим в нашу конфигурацию «служебный» регистр сведений, которого не будет видеть пользователь. Но при интерактивной записи данных во внешнюю таблицу соответствующая информация также будет программно сохраняться и в этом регистре сведений. На основе этого регистра можно затем построить различные отчеты и проверить полученную информацию.

Итак, добавим в конфигурацию периодический регистр сведений ВизитыВСпа и снимем у него флажок Использовать стандартные команды. Создадим у регистра два измерения: измерение Клиент, имеющее тип ссылки на таблицу clients внешнего источника данных Спа, и измерение Процедура, имеющее тип ссылки на таблицу spa_treatments. А также создадим ресурс Количество числового типа (рис. 2.42).

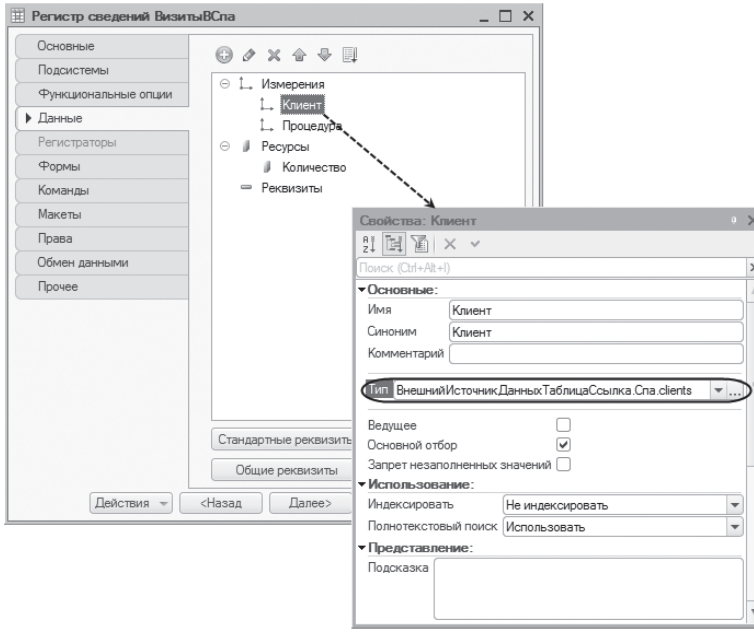


Рис. 2.42. Структура регистра «ВизитыВСпа»

Теперь создадим форму записи таблицы visits. Затем создадим обработчик события формы `ПриЗаписиНаСервере()` и заполним его следующим образом (листинг 2.9).

Листинг 2.9. Обработчик события формы «ПриЗаписиНаСервере»

```
&НаСервере
Процедура ПриЗаписиНаСервере(Отказ, ТекущийОбъект, ПараметрыЗаписи)

    // Подготовить менеджер записи.
    ЗаписьРегистра = РегистрыСведений.ВизитыВСпа.СоздатьМенеджерЗаписи();

    // Установить ключевые поля менеджера записи.
    ЗаписьРегистра.Период = ТекущаяДата();
```

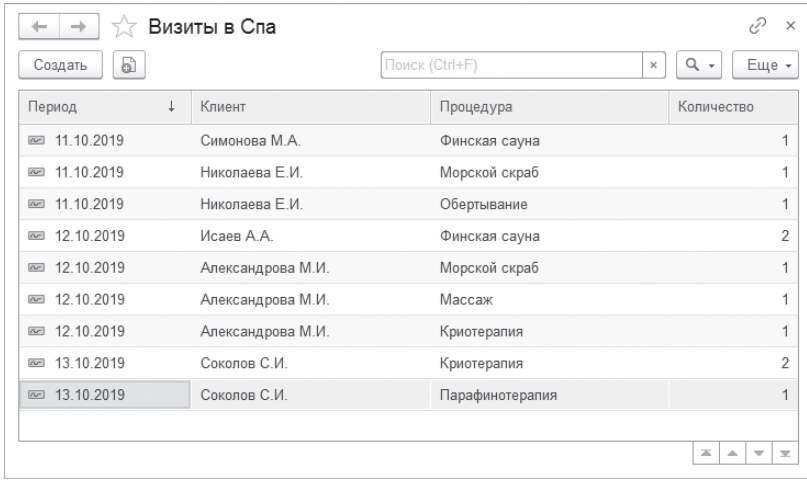
```
        ЗаписьРегистра.Клиент = Запись.client_id;
        ЗаписьРегистра.Процедура = Запись.spa_treatment_id;
        ЗаписьРегистра.Количество = Запись.number;

        ЗаписьРегистра.Записать();

КонецПроцедуры
```

В этом обработчике мы с помощью менеджера записи добавляем запись в регистр сведений ВизитыВСпа. Поле Период заполняем текущей датой, поле измерения Клиент заполняем ссылкой на клиента, поле измерения Процедура – ссылкой на процедуру, а поле ресурса Количество – количеством выполненных клиенту процедур.

В результате регистр сведений может иметь следующее содержимое (рис. 2.43).



Период	Клиент	Процедура	Количество
11.10.2019	Симонова М.А.	Финская сауна	1
11.10.2019	Николаева Е.И.	Морской скраб	1
11.10.2019	Николаева Е.И.	Обертывание	1
12.10.2019	Исаев А.А.	Финская сауна	2
12.10.2019	Александрова М.И.	Морской скраб	1
12.10.2019	Александрова М.И.	Массаж	1
12.10.2019	Александрова М.И.	Криотерапия	1
13.10.2019	Соколов С.И.	Криотерапия	2
13.10.2019	Соколов С.И.	Парафинотерапия	1

Рис. 2.43. Содержимое регистра сведений «ВизитыВСпа»

Поясним еще раз, что содержимое регистра, показанное на рис. 2.43, приведено просто для ознакомления. В интерфейсе приложения регистр не будет виден.

На основе информации, содержащейся в регистре, можно сформировать, например, следующий журнал посещений спа-салона (рис. 2.44).

Период		Количество
Клиент	Процедура	
11.10.2019		3
Симонова М.А.	Финская сауна	1
Николаева Е.И.	Морской скраб	1
Николаева Е.И.	Обертывание	1
12.10.2019		5
Исаев А.А.	Финская сауна	2
Александрова М.И.	Морской скраб	1
Александрова М.И.	Массаж	1
Александрова М.И.	Криотерапия	1
13.10.2019		3
Соколов С.И.	Криотерапия	2
Соколов С.И.	Парафинотерапия	1
Итого		11

Рис. 2.44. Журнал посещений спа-салона

Отчет

С помощью системы компоновки данных можно разрабатывать отчеты, основанные на данных таблиц внешних источников данных, точно так же, как и для других объектов конфигурации.

Продемонстрируем это на примере все той же таблицы visits внешнего источника данных Спа. Добавим в конфигурацию новый отчет с именем ВизитыКлиентов. Откроем конструктор схемы компоновки данных и добавим новый набор данных – запрос. Откроем конструктор запроса.

В качестве источника данных для запроса выберем таблицу visits внешнего источника данных Спа. В список полей перенесем все поля из этой таблицы (рис. 2.45).

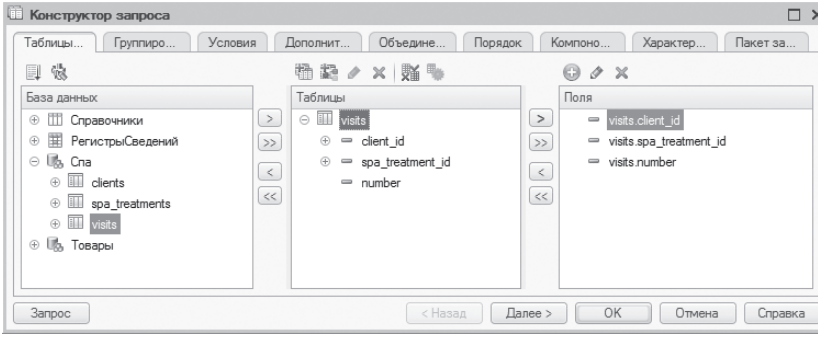


Рис. 2.45. Конструктор запроса

Нажмем ОК и вернемся в схему компоновки данных. На закладке Ресурсы перенесем все доступные ресурсы в список ресурсов отчета (рис. 2.46).

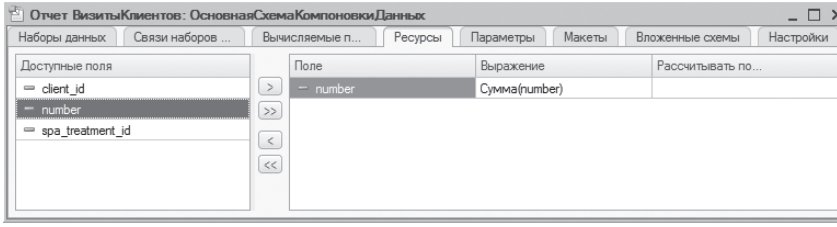


Рис. 2.46. Ресурсы отчета

Перейдем на закладку Настройки. Добавим в структуру отчета группировку по полю client_id, а в нее вложим группировку без указания группировочного поля – Детальные записи. Затем на закладке Выбранные поля выберем поля, которые будут выводиться в отчет: поле spa_treatment_id и ресурс number (рис. 2.47).

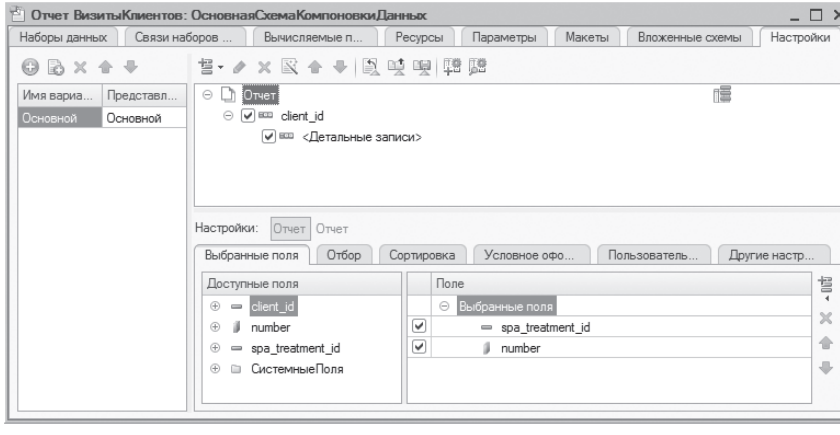
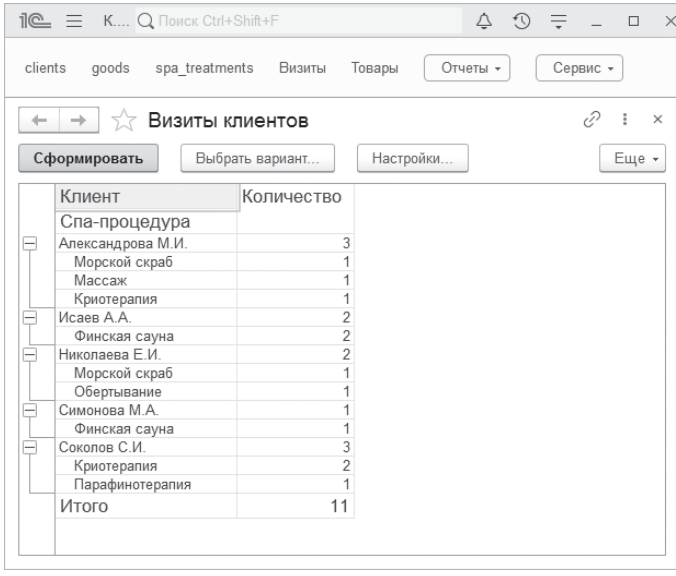


Рис. 2.47. Группировки и поля отчета

В результате в режиме 1С:Предприятие сформированный отчет примет следующий вид (рис. 2.48).



Клиент	Количество
Спа-процедура	
Александрова М.И.	3
Морской скраб	1
Массаж	1
Криотерапия	1
Исаев А.А.	2
Финская сауна	2
Николаева Е.И.	2
Морской скраб	1
Обертывание	1
Симонова М.А.	1
Финская сауна	1
Соколов С.И.	3
Криотерапия	2
Парафинотерапия	1
Итого	11

Рис. 2.48. Результат отчета

Обратите внимание, что вместо ключевых полей таблицы visits в отчет выводятся фамилии клиентов и наименования выполненных им процедур, поскольку эти поля являются полями представления таблиц clients и spa_treatments, на которые они ссылаются.

Глава 3. Обмен данными

Механизмы обмена данными позволяют решать различные задачи интеграции прикладных решений «1С:Предприятия» как друг с другом, так и с другими информационными системами.

Обмен данными в системе «1С:Предприятие» реализуется благодаря использованию ряда средств технологической платформы, которые разработчик может применять как по отдельности, так и в различных комбинациях, в зависимости от конкретной решаемой задачи. Такой подход позволяет обеспечить гибкость механизмов обмена и их настраиваемость на решение как можно большего круга задач.

В состав средств платформы, используемых для построения схем обмена данными, входят:

- объекты конфигурации `План обмена`,
- базовые средства работы с XML,
- средства XML-сериализации.

При помощи этих средств могут быть реализованы два механизма обмена данными:

- универсальный механизм обмена данными,
- механизм распределенных информационных баз.

Универсальный механизм обмена данными позволяет создавать произвольные распределенные системы и практически не накладывает никаких ограничений на структуру создаваемой системы. Можно как связывать в единое целое базы «1С:Предприятия» с отличными друг от друга конфигурациями, так и осуществлять обмен с принципиально отличными информационными системами (базами данных).

Механизм распределенных информационных баз, напротив, предназначен для обмена данными только с идентичными конфигурациями «1С:Предприятия 8» и жестко регламентирует структуру создаваемой системы.

Основные отличия механизма универсального обмена данными от механизма распределенных информационных баз можно проиллюстрировать следующей таблицей (табл. 3.1).

Таблица 3.1. Сравнение механизмов обмена данными

	Универсальный механизм обмена данными	**Механизм распределенных информационных баз**
В узлах плана обмена находятся	Произвольные базы данных, в том числе информационные базы «1С:Предприятия»	Только информационные базы «1С:Предприятия 8»
Конфигурации информационных баз	Могут быть разные (применительно к информационным базам «1С:Предприятия»)	Только идентичные
В сообщениях обмена передаются	Только изменения данных	Изменения данных, изменения конфигурации
Направление передачи	Произвольное, между двумя связанными узлами	Изменения данных – произвольное, изменения конфигурации – от главного к подчиненному узлу
Формат файлов обмена	Произвольный, чаще всего XML	XML
Создание распределенной базы и выполнение обмена	Требуется написание кода (определение порядка разрешения коллизий, стратегии распространения данных, создания начальных выгрузок, решение задачи синхронизации данных)	Может быть выполнено исключительно интерактивными средствами, без кодирования (действуют соглашения по умолчанию)
Структура распределенной системы	Произвольная. Может отсутствовать понятие «главный – подчиненный» (отсутствовать иерархия)	Древовидная. Любой узел (кроме корневого) имеет один главный и произвольное количество подчиненных узлов

ПОДРОБНЕЕ

Более подробно оба механизма обмена данными описаны в документации «1С:Предприятия» в разделе «Глава 15. Механизмы обмена данными».

Перед тем как перейти к рассмотрению возможностей организации как универсального обмена, так и создания распределенных информационных баз «1С:Предприятия», рассмотрим функциональность такого объекта платформы, как `План обмена`.

Планы обмена

При организации постоянного обмена может возникнуть ряд задач:

- с кем будет производиться обмен (определение состава участников обмена);
- какими данными будет производиться обмен (с одной стороны это определение перечня типов объектов; с другой стороны – определение «экземпляров»);
- определение регламента обмена (например, нумерация сообщений, адресация, процесс разрешения коллизий и т. п.).

Все эти задачи в той или иной мере могут решаться с использованием функциональности планов обмена. Рассмотрим данный объект более подробно.

Как объект конфигурации `ПланОбмена` характеризуется составом реквизитов, табличных частей (составом реквизитов табличных частей), определенными для него формами, макетами. У узла плана обмена существуют стандартные реквизиты Код, Наименование, ЭтотУзел т. п. В конфигурации может быть определено любое количество планов обмена.

Элементами данных плана обмена являются узлы плана обмена, подобно тому как элементами данных справочника являются элементы справочника. Каждый из узлов плана обмена обозначает участника обмена данными по данному плану обмена. Один из узлов (он является предопределенным) соответствует данной информационной базе, а остальные – другим участникам, с которыми данная информационная база может обмениваться данными (рис. 3.1).

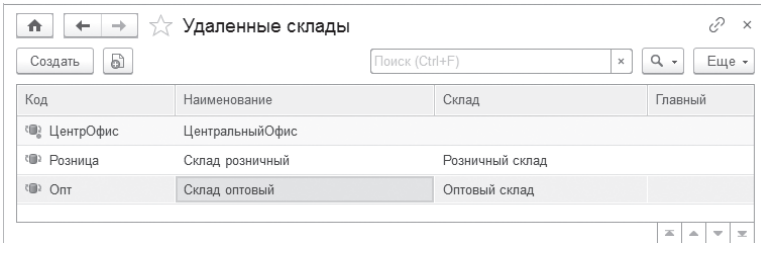


Рис. 3.1. Узлы плана обмена

Реквизиты и табличные части узла обмена могут использоваться для указания специфических данных по участнику обмена. С их помощью определяется порядок взаимодействия с данным участником, привязка его к другим объектам базы. Например, может указываться, что данный узел с точки зрения базы данных является «таким-то» складом (элементом справочника Склады), файл выгрузки данному получателю необходимо отправлять по «такому-то» FTP-адресу и т. п. (рис. 3.2).

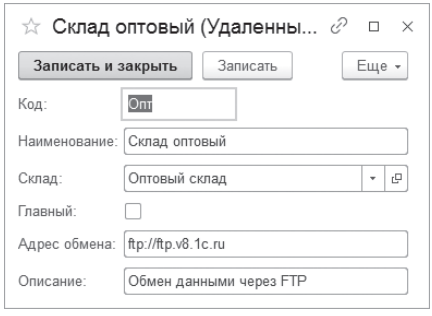


Рис. 2.3. Пример узла плана обмена

Данные переносятся между узлами с помощью сообщений. Средства работы с сообщениями образуют инфраструктуру сообщений. Каждое сообщение относится к определенному плану обмена, имеет определенный узел-отправитель и определенный узел-получатель. Сообщение не может быть отправлено неизвестному узлу и не может быть принято от неизвестного узла. Каждое сообщение имеет свой собственный целочисленный номер.

С точки зрения инфраструктуры сообщений у узла существуют два свойства:

- номер отправленного сообщения,
- номер принятого сообщения.

Для предопределенного узла эти свойства смысла не имеют (база данных сама с собой данными не обменивается).

Служба регистрации изменений предназначена для регистрации изменений данных, производимых «1С:Предприятием», чтобы при обмене данными иметь возможность передавать не все данные, а только новые, измененные и удаленные.

Настройка состава объектов, для которых включается регистрация изменений, производится в режиме Конфигуратор, на закладке Основные объекта конфигурации `ПланОбмена` (необходимо нажать кнопку Состав) – рис. 3.3.

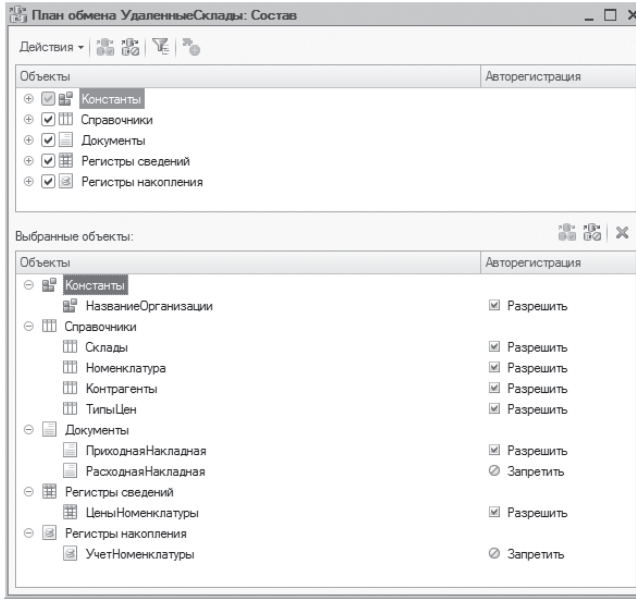


Рис. 3.3. Состав плана обмена

Можно сказать, что объекты, для которых включается регистрация изменений, являются входными данными для службы регистрации изменений. Задача этой службы состоит в том, чтобы, опираясь на данный перечень объектов, отслеживать изменения объектов, их удаление и производить соответствующие записи в таблицах регистрации изменений объектов. При этом отслеживаются ситуации повторного изменения (но об этом более подробно в разделе, посвященном именно службе регистрации изменений).

Обобщив изложенный ранее материал, можно сказать, что планы обмена:

- определяют состав участников обмена (любой узел, кроме предопределенного, соответствует какому-либо участнику обмена);
- позволяют производить регистрацию изменений объектов (служба регистрации изменений – как объектов, так и самой конфигурации);
- реализуют инфраструктуру сообщений.

Служба регистрации изменений

Довольно сложно разделить службу регистрации изменений и инфраструктуру сообщений: их функциональность тесно связана. Но все же постараемся рассмотреть особенности их работы по отдельности. Начнем со службы регистрации изменений.

Целью регистрации изменений является получение списка измененных элементов данных, которые должны быть переданы в очередном сообщении тому или иному узлу, с которым производится обмен данными. При каждом изменении данных должно быть зарегистрировано, что имеются изменения, которые предстоит передать во все узлы, с которыми поддерживается обмен этими данными. При получении подтверждения приема сообщения, в котором были отправлены изменения, записи регистрации изменений должны быть удалены.

Регистрация изменений может выполняться для следующих элементов данных:

- `КонстантаМенеджерЗначения.<имя>`.
- Объекты базы данных:
  - `СправочникОбъект.<имя>`;
  - `ДокументОбъект.<имя>`;
  - `ПланСчетовОбъект.<имя>`;
  - `ПланВидовХарактеристикОбъект.<имя>`;
  - `ПланВидовРасчетаОбъект.<имя>`;
  - `БизнесПроцессОбъект.<имя>`;
  - `ЗадачаОбъект.<имя>`.
- Наборы записей:
  - `РегистрСведенийНаборЗаписей.<имя>`;
  - `РегистрБухгалтерииНаборЗаписей.<имя>`;
  - `РегистрНакопленияНаборЗаписей.<имя>`;
  - `ПоследовательностьНаборЗаписей.<имя>`;
  - `РегистрРасчетаНаборЗаписей.<имя>`;
  - `ПерерасчетНаборЗаписей.<имя>`.

Для каждого из приведенных элементов данных ведется своя таблица регистрации изменений. Таблицы имеют разную структуру в зависимости от того, для каких элементов данных регистрируются изменения, но все-таки структуры таблиц подобны. В структуре можно выделить три составляющих:

- ключ элемента данных, для которого регистрируются изменения;
- ссылка на узел, в который изменение должно быть передано;
- номер сообщения, в котором изменение передано в первый раз.

Следует отметить, что в конфигурации может быть определено несколько планов обмена. Для каждого из них может включаться регистрация изменений какого-либо объекта. Так вот, вне зависимости от количества планов обмена, в которых участвует какой-либо объект, таблица регистрации изменений у объекта одна (можно сказать, что поле, содержащее ссылку на узел плана обмена, может иметь составной тип).

Структуры таблиц регистрации изменений для разных данных отличаются ключом, так как ключи у разных данных разные.

- Для константы ключом является идентификатор константы.
- Для объектов базы данных в качестве ключа используется ссылка на объект.
- Для наборов записей, для которых определен регистратор, в качестве ключа используется ссылка на объект-регистратор.
- Для набора записей регистра сведений, если регистратор не определен, в качестве ключа используется совокупность измерений, входящих в основной отбор. А если регистр сведений является периодическим и включен основной отбор по периоду, то в ключ входит еще и период.

Основной отбор (для регистров сведений) позволяет определять логическую единицу обмена данными. Рассмотрим это понятие более подробно (рис. 3.4).

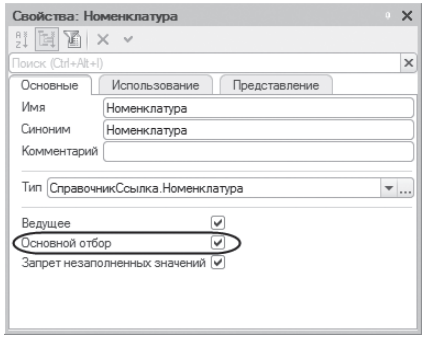


Рис. 3.4. Свойство «Основной отбор»

Понятно, что физически минимальной единицей обмена является запись регистра сведений. Но всегда ли правильно обмениваться записями? Рассмотрим пример. Есть регистр сведений, в котором содержатся данные

о дополнительных свойствах номенклатурных позиций. Речь идет именно о свойствах, которые четко характеризуют товарную позицию. Если для какого-то товара свойство имеет другое значение, то, значит, это другой товар. Например, такое свойство, как количество компрессоров холодильника. У данного регистра сведений два измерения: ссылка на номенклатуру и ссылка на свойство.

Если обмен будет производиться по каждой записи отдельно, у одного участника обмена добавили одно свойство, а у другого – другое, то в результате обмена у элемента номенклатуры появятся два свойства. Но в данном случае более правильно, чтобы свойства отдельно взятого товара представляли собой единое логическое целое (входили в некий логический квант данных).

Как раз для того, чтобы объединить несколько физических записей в один логический набор данных, и используется понятие *Основной отбор*. Если в приведенном примере для измерения Номенклатура (содержащего ссылку на номенклатурную позицию) установить свойство `Основной отбор`, то при изменении хотя бы одной физической записи обмен будет производиться всеми свойствами данной номенклатуры.

В качестве примера рассмотрим следующие данные (наполнение регистра сведений) – табл. 3.2.

Таблица 3.2. Записи регистра сведений

Измерение «Номенклатура» (входит в основной отбор)	**Измерение «Свойство» (не входит в основной отбор)**	**Значение**
Атлант МХМ 1704-00	Количество компрессоров	2
Атлант МХМ 1704-00	Цвет	Белый
Ardo TL 1000 EX-1	Тип загрузки	Вертикальная
Ardo TL 1000 EX-1	Количество подшипников	1

Добавим в регистр запись о новом свойстве (выделена серым фоном) – табл. 3.3.

Таблица 3.3. В регистр сведений добавлена запись

Измерение «Номенклатура» (входит в основной отбор)	**Измерение «Свойство» (не входит в основной отбор)**	**Значение**
Атлант МХМ 1704-00	Количество компрессоров	2
Атлант МХМ 1704-00	Цвет	Белый
Атлант МХМ 1704-00	Отделов в морозильной камере	3
Ardo TL 1000 EX-1	Тип загрузки	Вертикальная
Ardo TL 1000 EX-1	Количество подшипников	1

В результате будут зарегистрированы следующие изменения (табл. 3.4).

Таблица 3.4. Записи, попавшие в регистрацию изменений

Измерение «Номенклатура» (входит в основной отбор)	Измерение «Свойство» (не входит в основной отбор)	Значение
Атлант МХМ 1704-00	Количество компрессоров	2
Атлант МХМ 1704-00	Цвет	Белый
Атлант МХМ 1704-00	Отделов в морозильной камере	3

Если в основной отбор включить все измерения (если регистр сведений периодический, то включить в основной отбор и период), то логической единицей обмена будет отдельно взятая запись регистра сведений.

Следует отметить тот факт, что при записи набора в регистр сведений состав этого набора определяется логикой работы конфигурации и не обязательно совпадает с набором, получаемым «при использовании» основного набора. А при принудительной регистрации изменений для набора записей обязательно должен быть установлен отбор, совпадающий с основным (иначе будет вызвано исключение). Такой порядок работы определяется тем, что основной отбор является специфической чертой именно механизма обмена данными. Из сказанного выше следует, что при изменении основного отбора необходимо будет менять и программный код, в котором производится принудительная регистрация записей регистра сведений (во всех вхождениях в конфигурацию).

Изменение элемента данных должно быть зарегистрировано для всех узлов, в которые изменение должно быть передано. Таким образом, в результате изменения элемента данных в таблице регистрации изменений должно появиться N записей (наборов записей), где N – количество узлов, для которых регистрируются изменения (все узлы плана обмена, за исключением предопределенного). В каждой из этих записей (наборе записей) указано одно и то же значение ключа элемента данных и различные значения ссылки на узел.

Непосредственно после выполнения регистрации изменения номер сообщения имеет значение `NULL`. При первой отправке изменения в составе сообщения в данное поле помещается номер сообщения, в котором изменение отправлено.

Рассмотрим приведенный алгоритм на примере. Считаем, что в плане обмена включена регистрация для документа РасходнаяНакладная. В плане обмена определено три узла: один – предопределенный, другие – с кодами Оптовый и Розничный. После включения регистрации было изменено два документа. В соответствии с этим таблица регистрации изменения примет следующий вид (табл. 3.5).

Таблица 3.5. Состав таблицы регистрации изменений

ДокументСсылка.РасходнаяНакладная	Узел	Номер сообщения
Расходная накладная № 1 от…	Оптовый	Null
Расходная накладная № 1 от…	Розничный	Null
Расходная накладная № 2 от…	Оптовый	Null
Расходная накладная № 2 от…	Розничный	Null

После этого были произведены формирование и отправка сообщения с номером 1 для узла Оптовый. Таблица приняла следующий вид (табл. 3.6).

Таблица 3.6. Состав таблицы регистрации изменений

ДокументСсылка.РасходнаяНакладная	Узел	Номер сообщения
Расходная накладная № 1 от…	Оптовый	1
Расходная накладная № 1 от…	Розничный	Null
Расходная накладная № 2 от…	Оптовый	1
Расходная накладная № 2 от…	Розничный	Null

Далее было зарегистрировано изменение еще одного документа (табл. 3.7).

Таблица 3.7. Состав таблицы регистрации изменений

ДокументСсылка.РасходнаяНакладная	Узел	Номер сообщения
Расходная накладная № 1 от…	Оптовый	1
Расходная накладная № 1 от…	Розничный	Null
Расходная накладная № 2 от…	Оптовый	1
Расходная накладная № 2 от…	Розничный	Null
Расходная накладная № 3 от…	Оптовый	Null
Расходная накладная № 3 от…	Розничный	Null

После этого были сформированы и отправлены сообщения для узлов Оптовый и Розничный. Фактически были проведены две выгрузки (сформировано два сообщения). Для узла Розничный сообщение имело номер 1, для узла Оптовый – номер 2 (табл. 3.8).

Таблица 3.8. Состав таблицы регистрации изменений

ДокументСсылка.РасходнаяНакладная	Узел	Номер сообщения
Расходная накладная № 1 от…	Оптовый	1
Расходная накладная № 1 от…	Розничный	1
Расходная накладная № 2 от…	Оптовый	1
Расходная накладная № 2 от…	Розничный	1
Расходная накладная № 3 от…	Оптовый	2
Расходная накладная № 3 от…	Розничный	1

После выгрузки для узла Оптовый запись в таблице для документа Расходная накладная № 3 отмечается номером сообщения 2 (первый раз изменение было передано именно в сообщении с таким номером), другие записи (для которых в поле НомерСообщения содержалось значение 1) остались без изменений (именно таким образом работает служба регистрации изменений).

Для узла Розничный все изменения были отнесены сообщением с номером 1.

Предположим, что после выполнения вышеуказанных действий произошло повторное изменение документа Расходная накладная № 1. В этом случае таблица регистрации изменений будет иметь следующий вид (табл. 3.9).

Таблица 3.9. Состав таблицы регистрации изменений

ДокументСсылка.РасходнаяНакладная	Узел	Номер сообщения
Расходная накладная № 1 от…	Оптовый	Null
Расходная накладная № 1 от…	Розничный	Null
Расходная накладная № 2 от…	Оптовый	1
Расходная накладная № 2 от…	Розничный	1
Расходная накладная № 3 от…	Оптовый	2
Расходная накладная № 3 от…	Розничный	1

Продолжая разговор о службе регистрации изменений, следует отметить, что состав плана обмена распространяется на все узлы данного плана обмена. Поэтому если, например, с разными узлами нужно обмениваться разным составом информации, то необходимо создавать несколько планов обмена, в которых объединять узлы, обменивающиеся одинаковым составом информации.

Например, если с одним узлом необходимо обмениваться изменениями в справочнике Номенклатура, а с другим изменениями – в документах РасходнаяНакладная, то не стоит создавать один план обмена, лучше создать план

обмена ПоНоменклатуре (с соответствующей настройкой состава данных, по которым ведется регистрация изменений) и план обмена Расходные (с другим составом).

Управление регистрацией изменений

При рассмотрении вышеприведенного примера акцент делался на заполнение таблицы регистрации изменений. Каким образом производилась регистрация, не учитывалось. Можно сказать, что рассматривалась функциональность службы регистрации изменений с учетом того, что для всех объектов включена авторегистрация изменений. Пришло время рассмотреть и этот аспект работы программного комплекса.

При определении состава объектов, для которых производится регистрация изменений (кнопка Состав на закладке Основные объекта конфигурации `План обмена`), для каждого объекта можно определить свойство `Авторегистрация` (рис. 3.5).

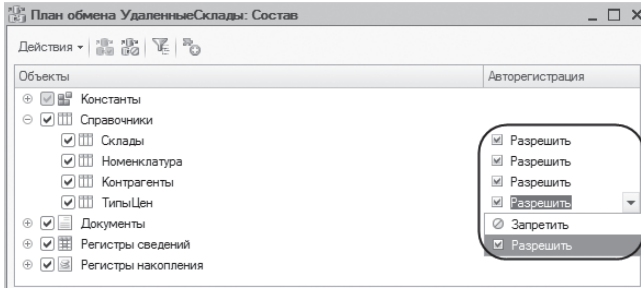


Рис. 3.5. Свойство «Авторегистрация»

Авторегистрацию можно разрешить или запретить. Если авторегистрация разрешена, то при изменении данных регистрация будет выполнена автоматически. Если запрещена, то регистрацию изменения необходимо выполнять «вручную» (определив код на встроенном языке). Следует отметить, что можно «корректировать» результат регистрации и в том случае, если авторегистрация объекта включена. Оба действия будут подробнее рассмотрены несколько позже.

У каждого объекта конфигурации, входящего в состав плана обмена, имеется свойство `ОбменДанными`, имеющее тип `ПараметрыОбменаДанными`. Данное свойство может быть использовано только для чтения и предназначено как для управления различными параметрами при обмене данными, так и для реализации других механизмов, связанных с изменением объекта. Например, такое свойство, как `Загрузка`, может определять необходимость

проведения (не проведения) каких-либо проверок в момент записи объекта (при записи в режиме загрузки можно отказаться от проверки номеров, кодов объектов, наличия каких-либо связанных с загружаемыми объектами данных и т. п., так как эти данные могут быть загружены после).

Следует обратить внимание на тот факт, что значение данного свойства не хранится в базе данных. В соответствии с этим все изменения (определения различных коллекций, свойств), которые выполняются через данное свойство, «работают» только в течение существования объекта.

У объекта `ПараметрыОбменаДанными` есть свойство `Получатели`, имеющее тип `НаборУзлов`. В данном свойстве хранится перечень узлов, для которых будет выполняться регистрация изменений при записи или удалении данных.

Рассмотрим особенности регистрации изменений как при автоматической регистрации изменений, так и в обратном случае.

Автоматическая регистрация изменений

В случае если для объекта включена автоматическая регистрация изменений, этот список получателей (свойство `Получатели`) заполняется автоматически, перед тем как будет вызван обработчик события `ПередЗаписью` (при выполнении записи данных) или `ПередУдалением` (при выполнении удаления). Перед вызовом данных обработчиков событий список получателей предварительно очищается. Исходя из этого, вносить изменения в список получателей (если для объекта включена автоматическая регистрация изменений) можно только в обработчиках `ПередЗаписью` (и/или `ПередУдалением`). При этом следует помнить, что список получателей может содержать только ссылки на узлы, относящиеся к планам обмена, в состав которых входит соответствующий объект конфигурации.

В приведенном ниже примере обработчик `ПередЗаписью` исключает из списка получателей узел с кодом Оптовый плана обмена УдаленныеСклады (листинг 3.1).

Листинг 3.1. Пример обработчика события «ПередЗаписью»

```
Процедура ПередЗаписью()

    Узел = ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Оптовый");
    Объект.ОбменДанными.Получатели.Удалить(Узел);

КонецПроцедуры
```

Следует помнить, что значение свойства ОбменДанными не хранится в информационной базе. Можно сказать, что данное свойство заполняется некими значениями по умолчанию при создании объекта (в оперативной памяти). Одним из таких свойств, которое в дальнейшем определяет порядок регистрации изменений, является свойство Автозаполнение. То есть, несмотря на тот факт, что при настройке состава регистрации изменений авторегистрация для объекта была включена, при создании объекта (при создании, получении из кода, при открытии формы объекта) это свойство можно переопределить.

Пример реализации приведен в листинге 3.2.

Листинг 3.2. Пример изменения свойства «Автозаполнение»

```
Узел = ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Оптовый");
Объект.ОбменДанными.Получатели.Автозаполнение = Ложь;
Объект.ОбменДанными.Получатели.Добавить(Узел);
Объект.Записать();
```

В этом случае автозаполнение отключается и состав коллекции получателей можно определять в любом возможном месте (участке кода, отвечающего за работу с объектом). Но, чтобы конфигурация была логически понятна и легко читаема, лучше это делать все в тех же обработчиках событий ПередЗаписью и/или ПередУдалением.

Также следует напомнить, что в конфигурации может быть определено несколько планов обмена. В одном из них для объекта может быть включена автоматическая регистрация изменений, в другом она может быть отключена.

Рассмотрим пример. План обмена УдаленныеОфисы (автоматическая регистрация изменений для документа РасходнаяНакладная включена) имеет следующий состав непредопределенных узлов:

- Центральный склад.
- Офис на «Рублевке».

План обмена УдаленныеСклады (автоматическая регистрация изменений для документа РасходнаяНакладная отключена) имеет следующий состав непредопределенных узлов:

- Розничный склад.
- Оптовый склад.

Подобная настройка планов обмена может быть выполнена исходя из соображений, что документ РасходнаяНакладная должен присутствовать во всех офисах компании и при этом должен быть выгружен только на «свой» склад.

При создании документа РасходнаяНакладная (или изменении существующего) в таблице регистрации изменений данного документа появятся две записи для узлов:

- УдаленныеОфисы – Центральный склад.
- УдаленныеОфисы − Офис на «Рублевке».

Следует оговориться, что такое поведение системы наблюдается только в том случае, если не предпринимается никаких шагов по принудительной регистрации изменений. Кстати, свойство `Автозаполнение` в коллекции `Получатели` в данном примере установлено в значение `Истина`.

Ручная регистрация изменений

Если автоматическая регистрация изменений не производится, то перед вызовом обработчиков `ПередЗаписью` и `ПриУдалении` сброс и заполнение списка получателей не осуществляются. Исходя из этого, заполнение данного списка может производиться в любом фрагменте кода, как показано в листинге 3.3.

Листинг 3.3. Пример заполнения списка узлов

```
Узел = ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Оптовый");
Объект.ОбменДанными.Получатели.Добавить(Узел);
Объект.Записать();
```

Если автоматическая регистрация изменений для объекта отключена и в конфигурации нигде не встречаются строки кода, подобные тем, которые приведены выше (или будут рассматриваться ниже), экземпляры данного объекта никогда не попадут в таблицу регистрации изменений (для данного объекта она будет пустой).

Вернемся к примеру с несколькими планами обмена (условия – в предыдущем разделе).

Несмотря на тот факт, что в свойстве `Автозаполнение` коллекции `Получатели` (свойства `ОбменДанными`) установлено значение `Истина`, для непредопределенных узлов, определенных в плане обмена УдаленныеСклады, регистрация изменений не проводится. Но список получателей перед вызовом обработчиков событий `ПередЗаписью` и `ПередУдалением` очищается. Исходя из этого, для ручной регистрации изменений можно в модуле объекта (документа) определить следующий обработчик события (листинг 3.4).

Листинг 3.4. Пример обработчика события «ПередЗаписью»

```
Процедура ПередЗаписью(Отказ)

	Если ОбменДанными.Получатели.Автозаполнение Тогда
		ВключитьРегистрацию(ОбменДанными, Склад);

		// Проверить, изменился ли склад в уже существующем документе.
		Если Не ЭтоНовый() Тогда
			Запрос = Новый Запрос("
				|ВЫБРАТЬ
				|	Склад
				|ИЗ
				|	Документ.РасходнаяНакладная
				|ГДЕ
				|	Ссылка = &ТекСсылка");
			Запрос.УстановитьПараметр("ТекСсылка", Ссылка);

			Выборка = Запрос.Выполнить().Выбрать();
			Выборка.Следующий();
			Если Выборка.Склад <> Склад Тогда
				ВключитьРегистрацию(ОбменДанными, Выборка.Склад);
			КонецЕсли;
		КонецЕсли;
	КонецЕсли;

КонецПроцедуры
```

Процедура для регистрации изменений объекта в конкретном узле обмена будет выглядеть следующим образом (листинг 3.5).

Листинг 3.5. Процедура «ВключитьРегистрацию()»

```
Процедура ВключитьРегистрацию(ОбменДанными, Склад) Экспорт

	// Получить список узлов для конкретного склада.
	Запрос = Новый Запрос;
	Запрос.Текст = "ВЫБРАТЬ
			|	УдаленныеСклады.Ссылка
			|ИЗ
			|	ПланОбмена.УдаленныеСклады КАК УдаленныеСклады
			|ГДЕ
			|	(УдаленныеСклады.Склад = &Склад
			|		ИЛИ УдаленныеСклады.Склад = &ПустойСклад)
			|	И УдаленныеСклады.ЭтотУзел = ЛОЖЬ";

	Запрос.УстановитьПараметр("Склад", Склад);
	Запрос.УстановитьПараметр("ПустойСклад",Справочники.Склады.ПустаяСсылка());
	Выборка = Запрос.Выполнить().Выбрать();

	// Зарегистрировать изменения для выбранных узлов.
	Пока Выборка.Следующий() Цикл
		ОбменДанными.Получатели.Добавить(Выборка.Ссылка);
	КонецЦикла;

КонецПроцедуры
```

Обе рассмотренные процедуры одновременно решают две задачи:

- Регистрируют изменения для узла, у которого реквизит Склад совпадает со складом, указанным в документе.
- Производят проверку (для ранее существовавших документов), не изменился ли склад. Если склад изменился, то производится регистрация изменения для узла, значение реквизита Склад которого соответствует старому значению документа (хранимому на момент проверки в информационной базе). Далее в момент выгрузки изменений данная регистрация может быть «подменена» на объект `УдалениеОбъекта`, что приведет к удалению накладной, «не свойственной» узлу.

Принудительная регистрация изменений

В некоторых случаях может потребоваться принудительная регистрация изменений для какого-либо объекта, их списка (или всех объектов). Для этой цели может использоваться метод `ЗарегистрироватьИзменения()` объекта `ПланыОбменаМенеджер`. Данный метод позволяет выполнять регистрацию изменений одиночных элементов данных или целых групп для одного или нескольких узлов.

Первый параметр данного метода – ссылка на узел плана обмена или массив ссылок на узлы, для которых выполняется регистрация изменений. Если первый параметр представляет собой одиночную ссылку на узел, то второй параметр может быть опущен. При этом выполняется регистрация изменений всех элементов данных, которые на данный момент присутствуют в базе данных и изменения которых могут быть зарегистрированы для данного узла (листинг 3.6).

Листинг 3.6. Регистрация изменений данных для указанного узла

```
Узел = ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Новый");
ПланыОбмена.ЗарегистрироватьИзменения(Узел);
```

Данный вариант использования метода может быть полезен для организации начальной передачи данных вновь созданному узлу.

Если же первый параметр представляет собой массив ссылок на узлы, то второй параметр обязательно должен быть указан (листинг 3.7).

Листинг 3.7. Регистрация изменений элемента данных для указанных узлов

```
Узлы = Новый Массив(2);
Узлы[0] = ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Оптовый");
Узлы[1] = ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Розничный");
Данные = Справочники.Номенклатура.НайтиПоКоду("000000004");
ПланыОбмена.ЗарегистрироватьИзменения(Узлы, Данные);
```

Впрочем, второй параметр может присутствовать и в том случае, если первый параметр – одиночная ссылка на узел. В зависимости от способа задания второго параметра можно зарегистрировать изменения одного элемента данных или же всех данных, относящихся к одному объекту конфигурации. Кроме того, можно зарегистрировать удаление объекта (указав в качестве второго параметра объект типа `УдалениеОбъекта`).

Для регистрации изменений одного элемента в качестве второго параметра может быть указан сам элемент данных, ссылка на объект базы данных или объект конфигурации.

Пример регистрации изменений всех данных, относящихся к объекту конфигурации, приведен в листинге 3.8.

Листинг 3.8. Регистрация изменений данных, относящихся к объекту конфигурации

```
Узлы = Новый Массив(2);
Узлы[0] = ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Оптовый");
Узлы[1] = ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Розничный");
ПланыОбмена.ЗарегистрироватьИзменения(Узлы, Метаданные.Справочники.Номенклатура);
```

До сих пор ничего не говорилось об очистке таблицы регистрации изменений. По большому счету существуют две основные стратегии выполнения данного действия:

- Гарантированная доставка сообщений. В этом случае очистка таблиц регистрации изменений производится сразу же после того, как сообщение будет сформировано.
- Ожидание квитанции. В этом случае очистка таблиц регистрации изменений производится при приеме сообщения, исходя из номера последнего полученного «от нас» сообщения (этот номер передается как квитанция в заголовке сообщения).

Рассмотрим следующий пример. Считаем, что таблица регистрации изменений документа РасходнаяНакладная имеет следующее наполнение (табл. 3.10).

Таблица 3.10. Таблица регистрации изменений

ДокументСсылка.РасходнаяНакладная	Узел	Номер сообщения
Расходная накладная № 1 от…	Оптовый	3
Расходная накладная № 1 от…	Розничный	Null
Расходная накладная № 2 от…	Оптовый	1
Расходная накладная № 2 от…	Розничный	1
Расходная накладная № 3 от…	Оптовый	2
Расходная накладная № 3 от…	Розничный	1
Расходная накладная № 4 от…	Оптовый	3
Расходная накладная № 4 от…	Розничный	Null

Исходя из данных таблицы, можно сказать, что в узел Оптовый было отправлено три сообщения, в узел Розничный – одно.

Используется стратегия ожидания подтверждающей квитанции, при этом от участника обмена Оптовый приходит сообщение, в котором указано, что последнее полученное от «нас» сообщение имело номер 2. Исходя из этой информации, можно удалить изменения, отправленные в сообщениях 1 и 2 (поле НомерСообщения таблицы регистрации изменений).

Таблица примет следующий вид (табл. 3.11).

Таблица 3.11. Таблица регистрации изменений

ДокументСсылка.РасходнаяНакладная	Узел	Номер сообщения
Расходная накладная № 1 от…	Оптовый	3
Расходная накладная № 1 от…	Розничный	Null
Расходная накладная № 2 от…	Розничный	1
Расходная накладная № 3 от…	Розничный	1
Расходная накладная № 4 от…	Оптовый	3
Расходная накладная № 4 от…	Розничный	Null

В случае гарантированной доставки таблица регистрации изменений очищается сразу после отправки сообщения. Например, таблица регистрации изменений имела следующий вид (табл. 3.12).

Таблица 3.12. Таблица регистрации изменений

ДокументСсылка.РасходнаяНакладная	Узел	Номер сообщения
Расходная накладная № 1 от…	Оптовый	3
Расходная накладная № 1 от…	Розничный	Null
Расходная накладная № 2 от…	Оптовый	1
Расходная накладная № 3 от…	Оптовый	2
Расходная накладная № 4 от…	Оптовый	3
Расходная накладная № 4 от…	Розничный	Null

В узел Розничный формируется сообщение с номером 2. После того как сообщение сформировано, таблица регистрации будет иметь следующий вид (табл. 3.13).

Таблица 3.13. Таблица регистрации изменений

ДокументСсылка.РасходнаяНакладная	Узел	Номер сообщения
Расходная накладная № 1 от…	Оптовый	3
Расходная накладная № 1 от…	Розничный	2
Расходная накладная № 2 от…	Оптовый	1
Расходная накладная № 3 от…	Оптовый	2
Расходная накладная № 4 от…	Оптовый	3
Расходная накладная № 4 от…	Розничный	2

Далее производится вызов метода по удалению записей в таблице регистрации изменений (метод рассматривается чуть позже). Таблица регистрации изменений примет такой вид (табл. 3.14).

Таблица 3.14. Таблица регистрации изменений

ДокументСсылка.РасходнаяНакладная	Узел	Номер сообщения
Расходная накладная № 1 от…	Оптовый	3
Расходная накладная № 2 от…	Оптовый	1
Расходная накладная № 3 от…	Оптовый	2
Расходная накладная № 4 от…	Оптовый	3

Для удаления записей регистрации изменений у менеджера планов обмена имеется метод `УдалитьРегистрациюИзменений()`. В качестве первого параметра метода может использоваться ссылка на узел или массив таких ссылок. В качестве второго параметра может указываться ссылка на объект, `УдалениеОбъекта`, объект конфигурации, число (номер сообщения).

Для удаления записей таблицы регистрации изменений с номерами сообщения не больше переданного (в примере с номерами 1 и 2) может использоваться следующий фрагмент кода (листинг 3.9).

Листинг 3.9. Пример удаления записей таблицы регистрации

```
ЧтениеXML = Новый ЧтениеXML;
ЧтениеXML.ОткрытьФайл(ИмяФайла);

// Загрузить сообщение обмена из файла и начать чтение сообщения.
ЧтениеСообщения = ПланыОбмена.СоздатьЧтениеСообщения();
ЧтениеСообщения.НачатьЧтение(ЧтениеXML);

// Удалить регистрацию с номерами сообщения не больше принятого (2).
НомерПринятого = ЧтениеСообщения.НомерПринятого;
ПланыОбмена.УдалитьРегистрациюИзменений(Узел, НомерПринятого);
```

Пример удаления записей регистрации изменений для всех данных, которые зарегистрированы для узла, приведен в листинге 3.10.

Листинг 3.10. Пример удаления записей таблицы регистрации

```
Узел = ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Оптовый");
ПланыОбмена.УдалитьРегистрациюИзменений(Узел);
```

Можно удалить записи регистрации изменений конкретного элемента данных для одного или нескольких узлов (листинг 3.11).

Листинг 3.11. Пример удаления записей таблицы регистрации

```
Узлы = Новый Массив(2);
Узлы[0] = ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Оптовый");
Узлы[1] = ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Розничный");
Данные = Справочники.Номенклатура.НайтиПоКоду("000000004");
ПланыОбмена.УдалитьРегистрациюИзменений(Узлы, Данные);
```

Также можно удалить записи регистрации изменений всех данных, относящихся к объекту конфигурации, для одного или нескольких узлов (листинг 3.12).

Листинг 3.12. Пример удаления записей таблицы регистрации

```
Узлы = Новый Массив(2);
Узлы[0] = ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Оптовый");
Узлы[1] = ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Розничный");
ПланыОбмена.УдалитьРегистрациюИзменений(Узлы,      Метаданные.Справочники.Номенклатура);
```

Инфраструктура сообщений

Важнейшей составляющей инфраструктуры сообщений являются сами сообщения. Как уже отмечалось, сообщения передаются в рамках плана обмена от одного узла другому. То есть каждое сообщение точно ассоциировано с планом обмена, имеет одного отправителя и одного получателя.

Рассмотрим, что такое сообщение. Сообщение оформляется как документ XML, имеющий определенную структуру. В качестве примера рассмотрим следующее сообщение (листинг 3.13).

Листинг 3.13. Пример сообщения обмена

```
<v8msg:Message xmlns:v8msg="http://v8.1c.ru/messages">
        <v8msg:Header>
                <v8msg:ExchangePlan>УдаленныеСклады</v8msg:ExchangePlan>
                <v8msg:To>Оптовый</v8msg:To>
                <v8msg:From>ЦентрОфис</v8msg:From>
```

```
            <v8msg:MessageNo>20</v8msg:MessageNo>
            <v8msg:ReceivedNo>15</v8msg:ReceivedNo>
        </v8msg:Header>
        <v8msg:Body>
            <!— Тело сообщения -->
        </v8msg:Body>
</v8msg:Message>
```

Все сообщение находится внутри элемента XML с именем Message, относящимся к пространству имен http://v8.1c.ru/messages. Сообщение делится на заголовок Header и тело сообщения Body. Оба относятся к пространству имен http://v8.1c.ru/messages.

Структура заголовка жестко задана. Информация заголовка представлена в нескольких элементах XML, вложенных в элемент Header. Все элементы, вложенные в элемент Header, относятся к пространству имен http://v8.1c.ru/messages.

- Элемент с именем ExchangePlan содержит имя плана обмена, к которому относится сообщение.
- Элемент с именем From содержит код узла-отправителя.
- Элемент с именем To содержит код узла, для которого предназначено сообщение.
- Элемент с именем MessageNo содержит номер данного сообщения. Номер сообщения является положительным целым числом и присваивается узлом-отправителем. Номер каждого последующего сообщения равен номеру предыдущего отправленного сообщения плюс 1.
- Элемент с именем ReceivedNo содержит максимальный номер сообщения, которое узел-отправитель данного сообщения принял от узла-получателя данного сообщения. Данное значение включено в состав заголовка сообщения для подтверждения приема сообщений.

Тело сообщения содержится в элементе XML с именем Body, относящемся к пространству имен http://v8.1c.ru/messages. Данный элемент может иметь произвольное содержимое, определяемое прикладными потребностями. Инфраструктурой сообщений содержимое тела сообщения никак не регламентируется.

Сообщение для выгрузки данных может быть создано следующим образом (листинг 3.14).

Листинг 3.14. Пример сообщения для выгрузки данных

```
ЗаписьXML = Новый ЗаписьXML();
ЗаписьXML.ОткрытьФайл("c:\temp\выгрузка.xml");
ЗаписьXML.ЗаписатьОбъявлениеXML();
Узел = ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Оптовый");

// Создать объект "ЗаписьСообщенияОбмена".
ЗаписьСообщения = ПланыОбмена.СоздатьЗаписьСообщения();
ЗаписьСообщения.НачатьЗапись(ЗаписьXML, Узел);

// Записать тело сообщения.
// ...

ЗаписьСообщения.ЗакончитьЗапись();
ЗаписьXML.Закрыть();
```

Выгрузка данных сопровождается созданием объекта типа `ЗаписьСообщенияОбмена` (переменная `ЗаписьСообщения`). Сразу после создания этот объект еще «не знает» своего номера, получателя и т. п. Вся подобная информация указывается при выполнении метода `НачатьЗапись()`. В качестве первого параметра передается объект `ЗаписьXML`, в качестве второго – ссылка на узел плана обмена (ссылка на получателя).

При выполнении этого метода сообщению присваивается номер, определяемый как номер предыдущего отправленного сообщения, увеличенный на 1 (информация берется из узла-получателя). Производится запись в XML-документ заголовка сообщения, а также записывается начало элемента XML, соответствующего телу сообщения. Устанавливается блокировка на запись базы данных, соответствующая узлу плана обмена.

Завершить запись сообщения можно с помощью двух методов объекта `ЗаписьСообщенияОбмена`:

- `ЗакончитьЗапись()`,
- `ПрерватьЗапись()`.

Вызов метода `ПрерватьЗапись()` прерывает запись сообщения, при этом оно считается неотправленным (счетчик отправленных сообщений в итоге не увеличивается). Неявный вызов этого метода происходит также в случае повторного вызова метода `НачатьЗапись()` (без последующего вызова метода `ЗакончитьЗапись()`) либо при разрушении (окончании времени жизни переменной, связанной с объектом) объекта `ЗаписьСообщенияОбмена`.

Метод `ЗакончитьЗапись()` осуществляет нормальное завершение записи сообщения. В этом случае в сообщение записывается конец элемента XML (представляющего тело сообщения). В узел плана обмена записывается

номер сообщения обмена данными. С записи узла плана обмена снимается блокировка, и сообщение считается отправленным.

Для чтения записанного в файл сообщения обмена в базе узла-получателя может использоваться следующий фрагмент кода (листинг 3.15).

Листинг 3.15. Пример чтения сообщения обмена

```
ЧтениеXML = Новый ЧтениеXML;
ЧтениеXML.ОткрытьФайл("c:\temp\выгрузка.xml");

// Создать объект "ЧтениеСообщенияОбмена".
ЧтениеСообщения = ПланыОбмена.СоздатьЧтениеСообщения();
ЧтениеСообщения.НачатьЧтение(ЧтениеXML, ДопустимыйНомерСообщения.Больший);

// Прочитать тело сообщения.
// ...

ЧтениеСообщения.ЗакончитьЧтение();
ЧтениеXML.Закрыть();
```

После открытия XML-документа, содержащего сообщение, создается объект `ЧтениеСообщенияОбмена`. Чтение данных начинается с выполнения метода `НачатьЧтение()` объекта `ЧтениеСообщенияОбмена`. При этом производится чтение заголовка сообщения обмена данными, проверяются содержащиеся в заголовке данные. Если какие-либо данные указаны неправильно (задан неизвестный план обмена, указан узел, не входящий в план обмена, номер сообщения не соответствует ожидаемому), инициируется исключение.

При начале чтения сообщения устанавливается блокировка на запись узла плана обмена. Используя второй параметр метода, можно установить вариант ожидаемого номера сообщения:

- любой (фактически контроль номеров сообщения не производится);
- очередной (строго больший на единицу, чем предыдущий);
- больший.

Следует обратить внимание на тот факт, что в приведенном фрагменте кода нет ни указания на используемый план обмена (их в конфигурации может быть несколько), ни указания на узел-отправитель. Данная информация получается из заголовка сообщения (в случае невозможности определения, как и было сказано выше, возникает исключение).

Завершить чтение сообщения можно с использованием двух методов:

- `ЗакончитьЧтение()`;
- `ПрерватьЧтение()`.

Обращение к методу ПрерватьЧтение() вызывает немедленное прерывание чтения сообщения. Блокировка с записи узла плана обмена снимается. В соответствующий узел плана обмена не вносится никаких изменений (не изменяется номер последнего принятого сообщения). Этот же метод вызывается неявно при разрушении (окончании времени жизни переменной, связанной с данным объектом) объекта ЧтениеСообщенияОбмена или перед повторным вызовом метода НачатьЧтение().

Выполнение метода ЗакончитьЧтение() вызывает попытку нормального завершения чтения сообщения. При этом проверяется нормальное завершение сообщения (текущим элементом в объекте ЧтениеXML является конец элемента, реализующего само сообщение). Номер принятого сообщения помещается в реквизит НомерПринятого узла плана обмена. Блокировка записи узла плана обмена снимается, и сообщение считается принятым.

Распределенные информационные базы

Распределенная информационная база представляет собой иерархическую структуру, состоящую из отдельных информационных баз «1С:Предприятия» – узлов распределенной информационной базы, между которыми организован обмен данными с целью синхронизации конфигурации и данных.

В основе механизмов распределенных информационных баз лежат универсальные механизмы обмена данными, но они содержат некоторые дополнительные возможности, недоступные через универсальные механизмы.

Главное отличие распределенных информационных баз от универсальных механизмов обмена данными заключается в том, что универсальные механизмы позволяют выстраивать достаточно произвольные схемы обмена данными, в то время как распределенные информационные базы имеют более узкую специализацию.

Общие принципы

Распределенная информационная база – это совокупность информационных баз «1С:Предприятия» (узлов распределенной информационной базы), в которых поддерживается синхронизация конфигурации и данных. Распределенная информационная база имеет иерархическую структуру. У каждого узла распределенной информационной базы может быть один главный и произвольное число подчиненных узлов. «Самый главный узел», или узел, у которого нет главного узла, называется корневым узлом распределенной

информационной базы. Каждый из узлов может обмениваться данными только со своими «соседями», то есть со своими главным и подчиненными узлами (рис. 3.6).

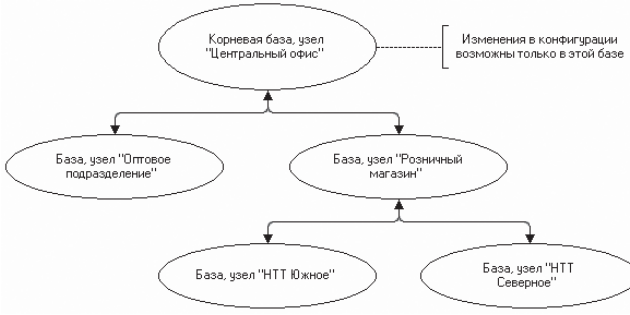


Рис. 3.6. Структура узлов распределенной информационной базы

Изменения конфигурации допускаются только в корневом узле распределенной информационной базы с последующим ее распространением по иерархии от корневого узла к его подчиненным и т. д. Таким образом, механизм управления распределенными информационными базами обеспечивает наличие во всех узлах распределенной информационной базы одной и той же конфигурации.

Изменение данных допускается в любом узле распределенной информационной базы. Синхронизация данных достигается путем распространения изменений данных, произведенных в одном узле, во все структуры распределенной информационной базы (рис. 3.7).

Рис. 3.7. Миграция данных в распределенной информационной базе

Если в рамках всей распределенной информационной базы поддерживается полная идентичность конфигурации, то полная идентичность данных не обязательна. Состав данных, изменения которых передаются в рамках распределенной информационной базы, может регулироваться как «по вертикали» (путем определения множества объектов конфигурации, данные которых участвуют в обмене), так и «по горизонтали» (путем задания условий на передачу и прием изменений на уровне отдельных элементов данных).

Планы обмена занимают центральное место в управлении распределенными информационными базами. Но для того, чтобы тот или иной план обмена мог использоваться для организации распределенной информационной базы, у него при конфигурировании должно быть установлено свойство `Распределенная информационная база`.

Данные в распределенной информационной базе переносятся с помощью сообщений, предоставляемых инфраструктурой сообщений. В отличие от универсальных механизмов обмена данными содержимое сообщений, передаваемых между узлами распределенной информационной базы, не может быть произвольным, а является регламентированным протоколом обмена, принятым для распределенной информационной базы.

Состав данных, изменениями которых будет производиться обмен в рамках распределенной информационной базы, определяется составом плана обмена. Вхождение объекта конфигурации в состав плана обмена показывает, что изменения данных, соответствующих объекту конфигурации, могут регистрироваться для узлов данного плана обмена. Но, в отличие от универсальных механизмов обмена данными, номенклатура данных, обмен которыми может производиться в рамках распределенной информационной базы, строго ограничена составом соответствующего плана обмена.

Для регистрации изменений данных в распределенной информационной базе задействована служба регистрации изменений. Элементы данных помещаются в сообщение с использованием механизмов XML-сериализации. Помимо изменений данных между узлами распределенной информационной базы передаются изменения конфигурации, а также некоторая дополнительная служебная информация. Регистрация изменений конфигурации и включение их в сообщение обмена в распределенной информационной базе осуществляются полностью автоматически и недоступны для пользователя и разработчика конфигураций.

В отличие от универсальных механизмов обмена данными формирование и прием сообщения обмена данными в распределенной информационной базе производятся «в одно действие», то есть все содержимое сообщения формируется путем вызова одного метода встроенного языка (листинг 3.16).

Листинг 3.16. Пример вызова метода «ЗаписатьИзменения()»

```
ПланыОбмена.ЗаписатьИзменения(ЗаписьСообщения, 0);
```

Считывание содержимого сообщения производится также путем вызова одного метода (листинг 3.17).

Листинг 3.17. Пример вызова метода «ПрочитатьИзменения()»

```
ПланыОбмена.ПрочитатьИзменения(ЧтениеСообщения);
```

Пример использования данных методов приведен в разделе «Запись и чтение сообщений обмена».

Для того чтобы управлять составом данных, помещаемых в сообщение, а также считываемых из сообщения и помещаемых в базу данных, на уровне отдельных элементов данных в модуле плана обмена могут быть определены обработчики событий:

- `ПриОтправкеДанныхПодчиненному`,
- `ПриОтправкеДанныхГлавному`,
- `ПриПолученииДанныхОтПодчиненного`,
- `ПриПолученииДанныхОтГлавного`.

Таким образом, в распределенной информационной базе практически полностью задействованы универсальные механизмы обмена данными, но имеются и некоторые дополнительные возможности, недоступные вне распределенной информационной базы.

Главный и подчиненный узлы

Как было указано выше, у каждого из узлов распределенной информационной базы может быть один главный и произвольное число подчиненных узлов. Для своего главного узла узел является подчиненным и, соответственно, для своих подчиненных – главным. Узел, у которого нет главного узла, является корневым узлом распределенной информационной базы. Корневой узел распределенной информационной базы – это единственное место, где разрешено вносить изменения в конфигурацию информационной базы.

Распределенная информационная база может быть построена на основе нескольких планов обмена, с установленным свойством `Распределенная информационная база`. Взаимодействие в каждой паре узлов «главный – подчиненный» производится в соответствии с одним из определенных

в конфигурации планов обмена. Никаких ограничений на использование того или иного плана обмена в том или ином узле распределенной информационной базы не накладывается.

Каждый из узлов распределенной информационной базы, как и в случае использования универсальных механизмов обмена данными, «знает» только своих «соседей», то есть свой главный и свои подчиненные узлы (рис. 3.8).

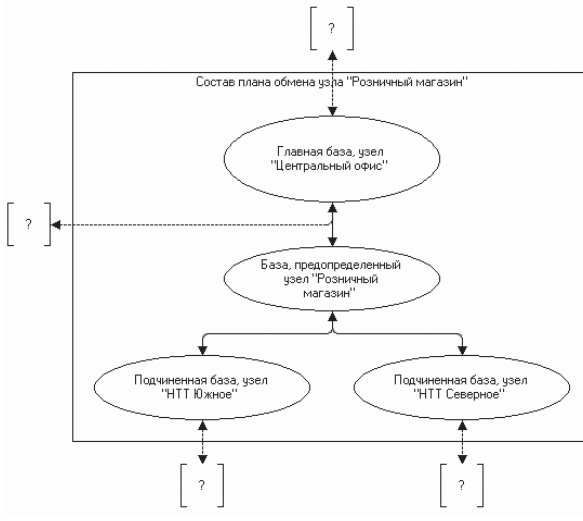


Рис. 3.8. Состав узлов распределенной информационной базы для узла «Розничный магазин»

Таким образом, полная схема распределенной информационной базы при наличии более чем двух уровней неизвестна никакому из узлов. Это является особенностью службы регистрации изменений: изменения регистрируются для всех непредопределенных узлов, если в каком-либо плане обмена указать все узлы распределенной базы, то это нарушит структуру подчинения баз данных.

Сообщение обмена данными в распределенной информационной базе

Для передачи изменений данных и конфигурации в распределенной информационной базе используются сообщения обмена данными, предоставляемые инфраструктурой сообщений. Если в случае применения универсальных механизмов обмена данными разработчик конфигурации сам определяет, что и как помещается в тело сообщения, то в случае распределенной

информационной базы структура и состав данных, помещаемых в тело сообщения, четко определены.

Рассмотрим структуру сообщения обмена данными, используемого в распределенной информационной базе. В качестве примера рассмотрим следующее сообщение (листинг 3.18).

Листинг 3.18. Пример структуры сообщения обмена

```
<v8msg:Message>
  <v8msg:Header>
    <v8msg:ExchangePlan>УдаленныеСклады</v8msg:ExchangePlan>
    <v8msg:To>Склад1</v8msg:To>
    <v8msg:From>Офис</v8msg:From>
    <v8msg:MessageNo>20</v8msg:MessageNo>
    <v8msg:ReceivedNo>15</v8msg:ReceivedNo>
  </v8msg:Header>
  <v8msg:Body>
    <v8de:Changes>
      <v8de:Signature>7b4d5320 f69c 4a7b 9273 ff56607fc8ab</v8de:Signature>
      <v8de:Config>
        <!--Измененные объекты конфигурации -->
        <v8de:Digest1>88d3f3a6ba3f4df03c7ec00f154837fc</v8de:Digest1>
        <v8de:Digest2>00cf636b02a488103a64c7a2cf81069e</v8de:Digest2>
      </v8de:Config>
      <v8de:ConfigurationExtensions>
        <ConfigurationExtensionDeletion>
          <!--идентификатор удаляемого расширения -->
        </ConfigurationExtensionDeletion>
        <v8de:ConfigurationExtension>
          <!--Данные расширения 1 -->
        </v8de:ConfigurationExtension>
        …
        <v8de:ConfigurationExtension>
          <!--Данные расширения N -->
        </v8de:ConfigurationExtension>
      </v8de:ConfigurationExtensions>
      <v8de:Nodes>
        <v8de:Node>
          <!--Данные главного узла -->
        </v8de:Node>
        <v8de:Node>
          <!--Данные подчиненного узла -->
        </v8de:Node>
      </v8de:Nodes>
      <v8de:Data>
        <!--Измененные элементы данных -->
      </v8de:Data>
    </v8de:Changes>
  </v8msg:Body>
</v8msg:Message>
```

Как видно из примера, все особенности сообщения обмена данными, используемого в распределенной информационной базе, сосредоточены в теле сообщения. Тело сообщения (элемент Body, относящийся к пространству имен http://v8.1c.ru/messages) содержит один-единственный элемент XML – Changes, относящийся к пространству имен http://v8.1c.ru/dataexchange. Внутри этого элемента сосредоточены все данные, передаваемые при обмене данными в распределенной информационной базе.

Элемент Changes может содержать следующие вложенные элементы:

- Элемент Signature содержит «подпись» плана обмена, в соответствии с которым получено сообщение.
- Элемент Config содержит изменения конфигурации, а также данные, идентифицирующие состояние конфигурации. Необязательные элементы Metadata, вложенные в Config, содержат изменения отдельных объектов конфигурации. Если изменения конфигурации не передаются в сообщении, то элементы Metadata отсутствуют. Такие элементы могут присутствовать только в сообщениях, передаваемых от главного узла подчиненному. Элементы Digest1 и Digest2 содержат цифровые подписи передаваемых в данном сообщении изменений конфигурации и всей конфигурации за вычетом изменений. Элементы Digest1 и Digest2 присутствуют во всех сообщениях: передаваемых от главного узла подчиненному и наоборот.
- Необязательный элемент ConfigurationExtensions содержит информацию о расширениях, которые могут передаваться в рамках распределенной информационной базы. В данном элементе содержится несколько вложенных элементов ConfigurationExtension, каждый из которых описывает одно расширение. Также в качестве подчиненного элемента может выступать элемент ConfigurationExtensionDeletion, который содержит идентификатор расширения, которое должно быть удалено в информационной базе-приемнике данного сообщения обмена. Подробнее о расширениях в распределенной информационной базе рассказывается в разделе «Обмен данными в распределенной информационной базе».
- Элемент Nodes может присутствовать только в сообщениях, передаваемых от главного узла подчиненному. Этот элемент содержит два вложенных элемента Node, первый из которых содержит данные главного узла (отправителя), а второй – подчиненного (получателя).
- И, наконец, элемент Data содержит измененные элементы данных, передаваемые в сообщении. Элементы данных помещаются в сообщение с помощью XML-сериализации.

Работа с распределенной информационной базой

В работе с распределенной информационной базой можно выделить следующие основные действия:

- создание начального образа подчиненного узла распределенной информационной базы;
- запись сообщения обмена данными для отправки в другой узел распределенной информационной базы;
- чтение сообщения обмена данными, отправленного из другого узла распределенной информационной базы.

Если для доступа к возможностям, предоставляемым универсальными механизмами обмена данными, без встроенного языка не обойтись, то перечисленные действия могут быть выполнены как из встроенного языка, так и интерактивно с помощью команд меню Еще формы списка плана обмена или иными средствами, определенными при конфигурировании.

Создание начального образа

Для того чтобы интерактивно создать начальный образ для узла плана обмена, необходимо открыть форму списка плана обмена или непосредственно форму узла плана обмена. Для выбранного (непредопределенного) узла плана обмена следует выполнить команду Создать начальный образ, которая доступна в командной панели формы, а также в меню Еще (рис. 3.9).

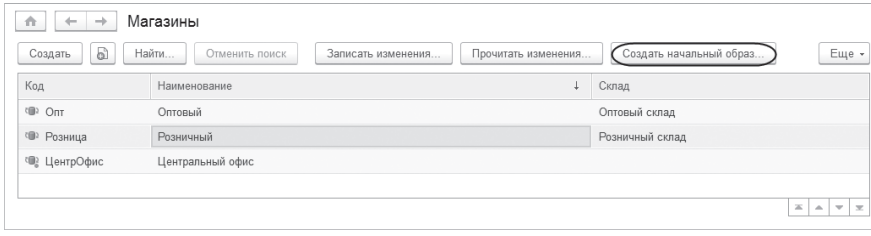


Рис. 3.9. Интерактивное создание начального образа

При этом на экране появится диалог создания начального образа, в котором будет предложено заполнить все необходимые параметры информационной базы – нового узла распределенной информационной базы. Структура диалога создания начального образа узла распределенной информационной базы аналогична структуре диалога создания новой информационной базы. После заполнения всех необходимых параметров платформа «1С:Предприятие» выполнит создание начального образа:

- Будет создана новая информационная база с указанными параметрами.
- Конфигурация распределенной информационной базы будет перенесена во вновь созданную информационную базу.
- Будет выполнен перенос данных текущей информационной базы во вновь созданную, при этом для каждого элемента данных будет вызван обработчик события узла плана обмена `ПриОтправкеДанныхПодчиненному` (т. е. будут перенесены только те данные, которые входят в состав соответствующего плана обмена, с учетом всех ограничений, фильтров, которые определены в обработчике события `ПриОтправкеДанныхПодчиненному`).
- В новом узле распределенной информационной базы будут заполнены данные узла ЭтотУзел плана обмена данными узла данной информационной базы, для которого создается начальный образ. Этот узел будет предопределенным для базы подчиненного узла.
- В новом узле распределенной информационной базы будет создан элемент плана обмена, соответствующий данной информационной базе. Его данные будут перенесены. Затем ссылка на этот элемент будет установлена в качестве значения главного узла в этой информационной базе.
- Будут удалены все записи о регистрации изменений для узла, начальный образ которого был только что создан.

Процесс создания начального образа при помощи метода `СоздатьНачальныйОбраз()` менеджера планов обмена аналогичен интерактивной процедуре – за исключением того, что все параметры новой информационной базы, а также узел, для которого необходимо создать начальный образ, должны быть переданы в качестве параметров при вызове указанной процедуры (листинг 3.19).

Листинг 3.19. Пример создания начального образа

```
Узел = ПланыОбмена.Магазины.НайтиПоКоду("Оптовый");
ПланыОбмена.СоздатьНачальныйОбраз(Узел, СтрокаСоединения);
```

В строковой переменной `СтрокаСоединения` указывается информационная база, в которой будет создан начальный образ подчиненного узла распределенной информационной базы. Информационная база для создания начального образа должна быть пустой или не должна существовать вовсе. Строка соединения представляет собой набор параметров, каждый из которых представляет собой фрагмент вида <Имя>=<Значение>, где имя <Имя> − имя параметра, а <Значение> − его значение. Фрагменты отделяются друг от друга символами «;». Если значение содержит пробельные символы, то оно должно быть заключено в двойные кавычки («).

Запись и чтение сообщений обмена

Если план обмена используется в реализации механизма распределенных информационных баз, то в форме списка узлов данного плана обмена для непредопределенного узла, в командной панели, а также в меню Еще доступны две команды по записи и чтению сообщения обмена: Записать изменения, Прочитать изменения (рис. 3.10).

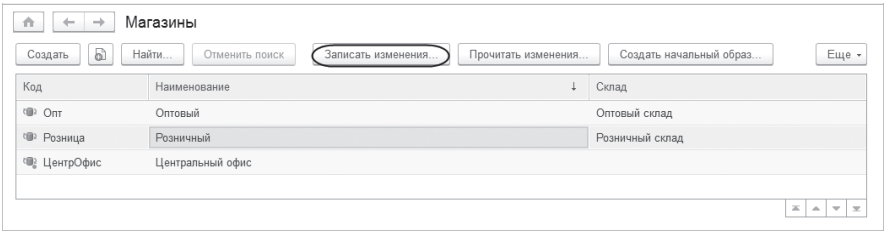


Рис. 3.10. Интерактивная запись и чтение изменений

При выборе команды (например, записи изменений) открывается диалог, в котором указывается количество элементов данных, обрабатываемых в одной транзакции, и устанавливается флажок Сжимать сообщение (XML-документ будет упакован в архив ZIP) – рис. 3.11.

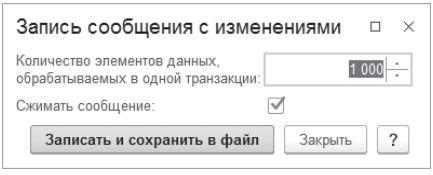


Рис. 3.11. Диалог записи сообщения обмена

После нажатия кнопки Записать и сохранить в файл нужно указать имя файла сообщения. Если файл был упакован в архив ZIP, то при чтении сообщения обмена из этого файла архив будет распакован.

К недостатку данного метода следует отнести тот факт, что в данном случае нет возможности отследить окончание как процесса выгрузки, так и процесса загрузки данных из сообщения обмена.

Аналогичное действие можно реализовать и самостоятельно, используя для записи следующий фрагмент кода (листинг 3.20).

Листинг 3.20. Пример записи сообщения обмена

```
ЗаписьXML.ОткрытьФайл("c:\temp\out.xml");
ЗаписьXML.ЗаписатьОбъявлениеXML();

// Создать новое сообщение.
Узел = ПланыОбмена.Магазины.НайтиПоКоду("Магазин");
ЗаписьСообщения = ПланыОбмена.СоздатьЗаписьСообщения();
ЗаписьСообщения.НачатьЗапись(ЗаписьXML, Узел);

// Записать изменения.
ПланыОбмена.ЗаписатьИзменения(ЗаписьСообщения, 1000);

ЗаписьСообщения.ЗакончитьЗапись();
ЗаписьXML.Закрыть();
```

Пример чтения сообщения обмена приведен в листинге 3.21.

Листинг 3.21. Пример чтения сообщения обмена

```
// Создать объект чтения XML.
ЧтениеXML = Новый ЧтениеXML;
ЧтениеXML.ОткрытьФайл("c:\temp\out.xml");

// Загрузить сообщение из файла.
ЧтениеСообщения = ПланыОбмена.СоздатьЧтениеСообщения();
ЧтениеСообщения.НачатьЧтение(ЧтениеXML);

// Прочитать изменения.
ПланыОбмена.ПрочитатьИзменения(ЧтениеСообщения,1000);

ЧтениеСообщения.ЗакончитьЧтение();
ЧтениеXML.Закрыть();
```

Данный подход позволяет отследить момент окончания процесса загрузки/выгрузки и при необходимости выполнить какие-либо действия.

При необходимости произвести упаковку полученного файла сообщений (или извлечение сообщения из архива) можно воспользоваться функциональностью платформы по работе с ZIP-архивами, рассмотренной в разделе «ZIP-архивы».

Подготовка конфигурации к работе в распределенной информационной базе

Конфигурация, разработанная для обычной информационной базы, в большинстве случаев требует адаптации для работы в распределенной информационной базе. В этом разделе мы рассмотрим основные методы адаптации конфигурации к работе в распределенной информационной базе.

Описанные методы не являются исчерпывающими, ими можно ограничиться, когда необходимо обеспечить простой обмен данными. Например, разделение на два офиса с полным дублированием информации в них.

Если же требуется реализация специфической схемы обмена, то может понадобиться реализация сложной архитектуры распределенного решения, влекущая за собой серьезную переработку конфигурации. Предположим, разрабатывается схема обмена, в которой данные из организаций холдинга должны передаваться в центральную информационную базу в виде сводных данных за определенный период. В этом случае требуется реализация в конфигурации соответствующих объектов для хранения детальной и сводной информации.

Одной из особенностей работы в модели распределенной информационной базы является возможность создания в различных узлах объектов (документов, элементов справочника, видов характеристик и т. д.), обладающих одинаковыми номерами (кодами). Это может привести к возникновению в одной информационной базе объектов с одинаковыми номерами (кодами) и привести к определенным проблемам.

Дело в том, что синхронизация объектов ведется по уникальному идентификатору. При создании двух и более одинаковых объектов в различных информационных базах можно определить у них одинаковые значения реквизитов, одинаковые наборы данных в табличных частях, но при этом значения уникальных идентификаторов (если механизм назначения уникальных идентификаторов не менялся разработчиком конфигурации) у них разные. В процессе загрузки такие объекты будут считаться разными. И в этих случаях для справочника могут создаться элементы с одинаковым кодом, для документов могут создаться экземпляры с одинаковым номером в определенном периоде и т. д.

Когда назначение номера (кода) выполняется системой автоматически, для подобных объектов необходимо ввести префиксацию, идентифицирующую место создания объекта. Для объектов, номера (коды) которых устанавливаются вручную, подобную префиксацию реализовывать не нужно (например, коды счетов назначаются пользователями вручную, добавление префикса для них является неправильным). Для объектов, требующих наличия префикса в номере (коде), необходимо:

- Установить тип номера (кода) в значение `Строка`.
- Добавить в конфигурацию объект конфигурации `Константа`, в котором будет установлено значение префикса номера (кода), однозначно идентифицирующее данный узел распределенной информационной базы (значение константы должно быть уникальным среди всех узлов распре-

деленной информационной базы, сама константа не должна быть включена в состав плана обмена, на базе которого развернута распределенная информационная база).

- Реализовать в модулях объектов обработчики событий, выполняемые при установке номера (кода) – ПриУстановкеНовогоНомера (ПриУстановкеНовогоКода). Реализация должна использовать в качестве значения префикса значение описанной ранее константы, листинг 3.22.

Листинг 3.22. Пример назначения префикса

```
Процедура ПриУстановкеНовогоКода(СтандартнаяОбработка, Префикс)

    Префикс = Константы.ПрефиксНомеров.Получить();

КонецПроцедуры
```

Работа конфигурации в распределенной информационной базе подразумевает обмен данными между узлами распределенной информационной базы. В процессе обмена порядок следования данных в сообщениях обмена не определен. При чтении данных могут возникать ситуации, при которых считанные данные ссылаются на несуществующие (возможно, еще не загруженные) данные. При записи считанных данных в информационную базу необходимо учесть возможность наличия неразрешенных ссылок в записываемых данных. Для этого в обработчиках событий ПередЗаписью, ПриЗаписи и ПередУдалением необходимо учитывать свойство Загрузка: при записи данных средствами механизма распределенной информационной базы данное свойство будет установлено в значение Истина. В указанных процедурах в режиме Загрузка не следует выполнять различные проверки, связанные с наличием тех или иных данных, участвующих в обмене (например, регистратор записываемого набора записей регистра накопления может быть еще не прочитан из сообщения обмена), не рекомендуется также выполнять изменение связанных данных.

В данном режиме рекомендуется дать возможность системе записать прочитанные данные. Контроль допустимости данных для данного узла распределенной информационной базы может быть осуществлен в соответствующих обработчиках, вызываемых при чтении сообщения обмена (ПриПолученииДанныхОтПодчиненного, ПриПолученииДанныхОтГлавного).

Особое внимание следует уделить регистрам сведений, имеющим независимый режим записи (свойство Режим записи установлено в значение Независимый). Для данных регистров гранулой обмена является набор записей с отбором по измерениям, с установленным свойством Основной

отбор. Необходимо проанализировать, будет ли соответствовать состав передаваемой информации логике работы конфигурации.

Например, в регистре сведений хранится информация о характеристиках товаров. В основной отбор включены измерения:

- Характеристика (идентифицирующее характеристику товара);
- Товар (определяющее собственно товар).

По логике работы конфигурации важно обеспечить целостность редактирования характеристик товара − состав характеристик товара должен редактироваться целиком. Считаем, что ситуация, когда в одном узле исправляется одна характеристика, а в другом – другая и через некоторое время (после обмена данными) в характеристиках происходит объединение исправлений, является недопустимой.

В данном случае (когда все измерения регистра сведений входят в основной отбор) гранулой обмена данными выступает одна запись регистра сведений. И это может привести к ситуации, при которой состав характеристик товара будет состоять из различных характеристик, полученных из разных узлов распределенной информационной базы (мы можем считать эту ситуацию недопустимой).

Для устранения подобных проблем следует внести коррективы в набор измерений, входящих в основной отбор регистра: исключить из основного отбора измерение, вносящее излишнюю гранулярность (характеристику из приведенного примера). Это позволит в процессе обмена передавать наборы записей, содержащие информацию, которая полностью соответствует логике работы конфигурации (все характеристики номенклатурной позиции являются логической единицей обмена).

Выполнение этих рекомендаций позволяет просто настроить обмен данными в рамках распределенной информационной базы. Дальнейшая адаптация конфигурации зависит от конкретной специфики обмена.

Пример реализации обмена данными в распределенной информационной базе

В этом разделе мы протестируем обмен данными сначала интерактивным, а затем программным образом, а также рассмотрим различные сценарии обмена данными в распределенной информационной базе.

ПОДРОБНЕЕ

Подробнее познакомиться с реализацией обмена данными в распределенной информационной базе можно в демонстрационной конфигурации «Примеры обмена», которая прилагается к книге.

В нашей демонстрационной конфигурации существует план обмена Магазины, на основе которого выполняется обмен данными в распределенной информационной базе. Для этого у плана обмена установлен флажок Распределенная информационная база (рис. 3.12).

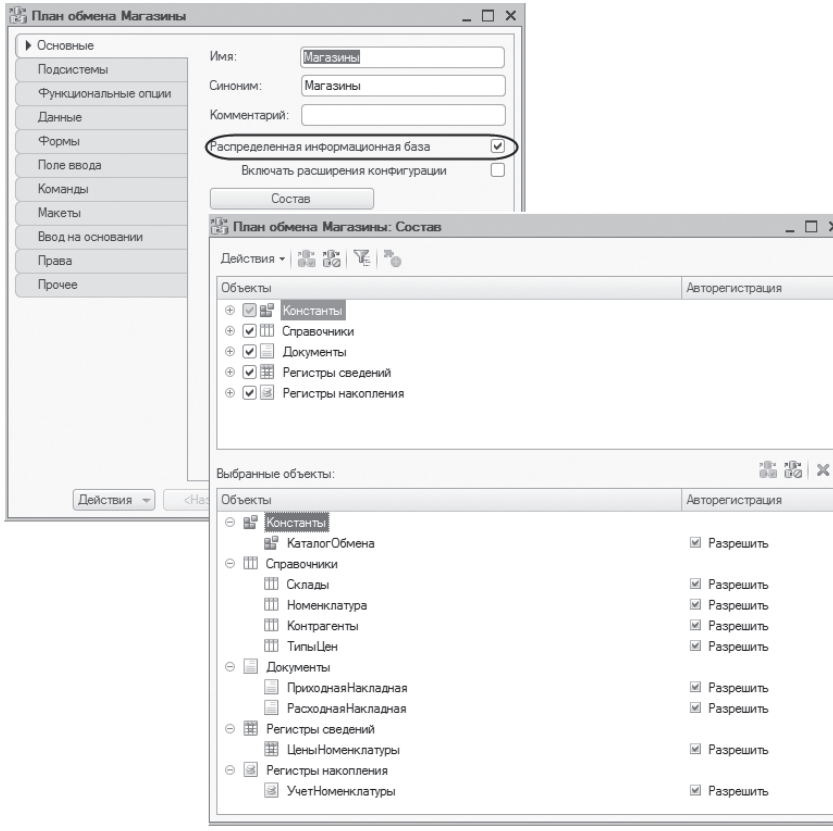


Рис. 3.12. План обмена «Магазины»

Состав плана обмена включает все объекты конфигурации за исключением константы ПрефиксНомеров, которая служит для установки префиксов кодов/номеров объектов, созданных в каждом узле обмена. Подробнее об этом рассказывается в разделе «Подготовка конфигурации к работе в распределенной информационной базе».

У плана обмена Магазины также существует реквизит Склад (тип СправочникСсылка.Склады) для хранения ссылки на склад, к которому относится конкретный узел обмена. Подробнее об этом будет рассказано ниже, в разделе «Распределение данных по подчиненным узлам».

Сначала создадим общий неглобальный модуль Обмен и поместим в нем экспортную функцию ПолучитьПрефиксНомера() – листинг 3.23.

Листинг 3.23. Функция «ПолучитьПрефиксНомера»

```
Функция ПолучитьПрефиксНомера() Экспорт

    Возврат Константы.ПрефиксНомеров.Получить();

КонецФункции
```

После этого в модули объектов всех справочников, участвующих в обмене, поместим процедуру ПриУстановкеНовогоКода() – листинг 3.24.

Листинг 3.24. Процедура «ПриУстановкеНовогоКода»

```
Процедура ПриУстановкеНовогоКода(СтандартнаяОбработка, Префикс)

    Префикс = Обмен.ПолучитьПрефиксНомера();

КонецПроцедуры
```

А в модули объектов всех документов, участвующих в обмене, поместим процедуру ПриУстановкеНовогоНомера() – листинг 3.25.

Листинг 3.25. Процедура «ПриУстановкеНовогоНомера»

```
Процедура ПриУстановкеНовогоНомера(СтандартнаяОбработка, Префикс)

    Префикс = Обмен.ПолучитьПрефиксНомера();

КонецПроцедуры
```

В результате созданные в базе каждого узла обмена объекты будут иметь префиксы, соответствующие значению константы, установленному при настройке этого узла.

Запустим «1С:Предприятие», введем некоторые тестовые данные и протестируем работу обмена в распределенной информационной базе.

В списке узлов плана обмена Магазины уже существует одна предопределенная запись, соответствующая нашей информационной базе. Это и будет центральный узел плана обмена. Установим его код (ЦентрОфис) и название (Центральный офис).

Затем добавим в список еще одну запись с кодом Опт и наименованием Оптовый. В поле Склад выберем значение Оптовый склад. Выделим этот узел, нажмем кнопку Создать начальный образ... и создадим подчиненный узел обмена путем создания начального образа из нашей центральной базы, как описано в разделе «Создание начального образа» (рис. 3.13).

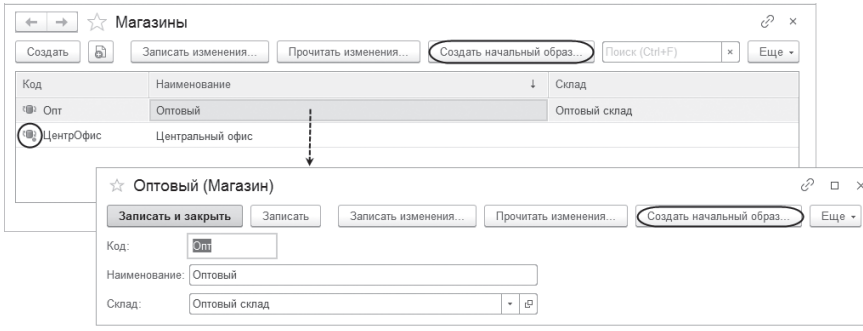


Рис. 3.13. Создание подчиненного узла «Оптовый» путем создания начального образа

При этом в базе подчиненного узла, в списке плана обмена Магазины, узел, для которого создавалась база (с кодом Опт), будет внесен платформой как предопределенный (помечен специальной пиктограммой в списке узлов), а узел центральной базы будет добавлен в качестве главного узла обмена (помечен специальной пиктограммой в списке узлов), рис. 3.14.

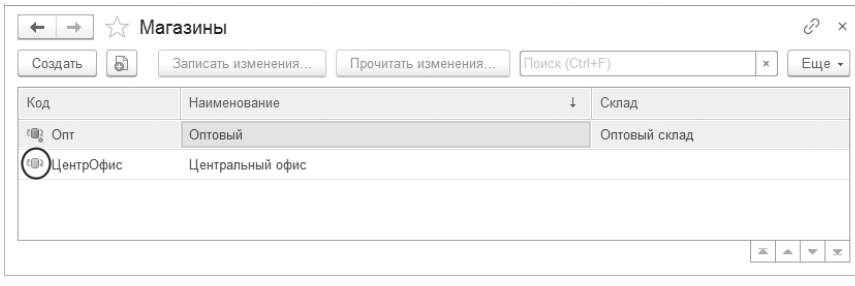


Рис. 3.14. Список узлов плана обмена для базы подчиненного узла

Для предотвращения конфликта записи объектов с одинаковыми номерами в базе каждого из узлов обмена нужно установить значение константы Префикс номеров. Подробнее об этом рассказывается в разделе «Подготовка конфигурации к работе в распределенной информационной базе». Для центральной базы установим значение этой константы как «ЦО» (рис. 3.15).

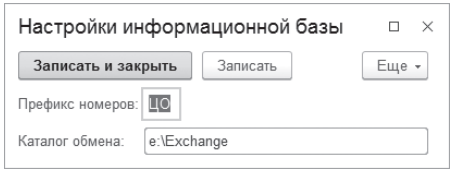


Рис. 3.15. Установка значения константы «ПрефиксНомеров»

После этого в базе центрального узла введем новый элемент справочника Номенклатура. Код этого элемента будет иметь префикс «ЦО» (рис. 3.16).

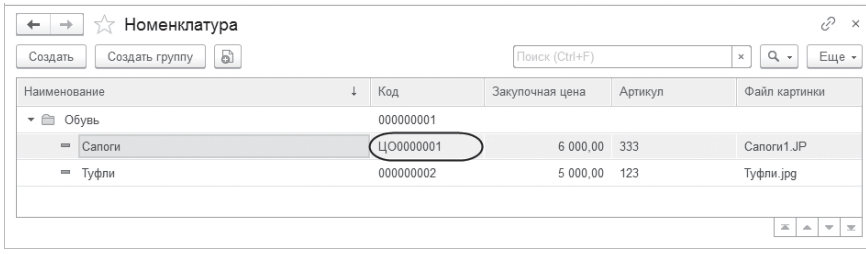


Рис. 3.16. Ввод новых данных в центральной базе

Теперь нам нужно передать эти изменения подчиненному узлу Оптовый. Для этого в центральной базе, в списке узлов плана обмена нужно выбрать подчиненный узел, которому будут передаваться данные (Оптовый), и нажать кнопку Записать изменения (рис. 3.17).

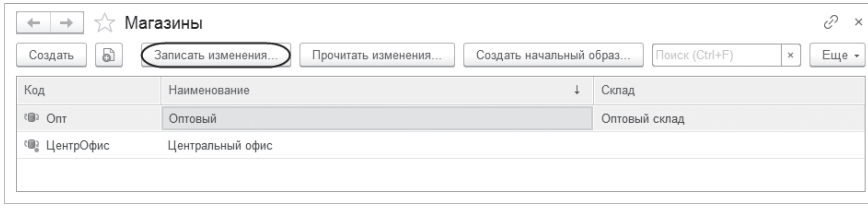


Рис. 3.17. Запись измененных данных из центральной базы в подчиненный узел «Оптовый»

Откроем базу подчиненного узла Оптовый. Там мы увидим все данные, перенесенные из центральной базы при создании начального образа. В списке узлов плана обмена Магазины выберем узел, из которого данные были выгружены (Центральный офис), нажмем кнопку Прочитать изменения и укажем файл архива, в который были записаны изменения от центрального узла

с автоматически сформированным именем Message_ЦентральныйОфис_Оптовый (рис. 3.18).

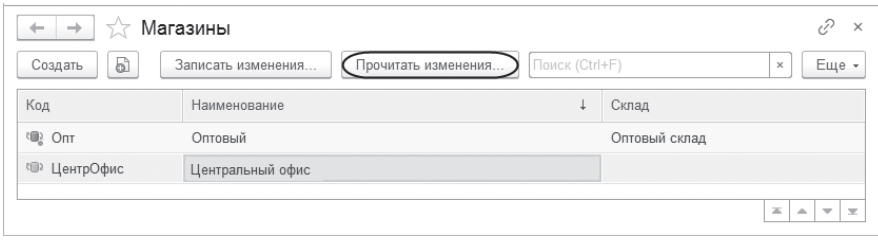


Рис. 3.18. Чтение измененных данных из центральной базы в подчиненный узел

Произойдет загрузка данных из центральной базы, являющейся корневым узлом, в подчиненный узел обмена. В справочнике Номенклатура мы увидим еще одну запись, код которой начинается с префикса «ЦО» (рис. 3.19).

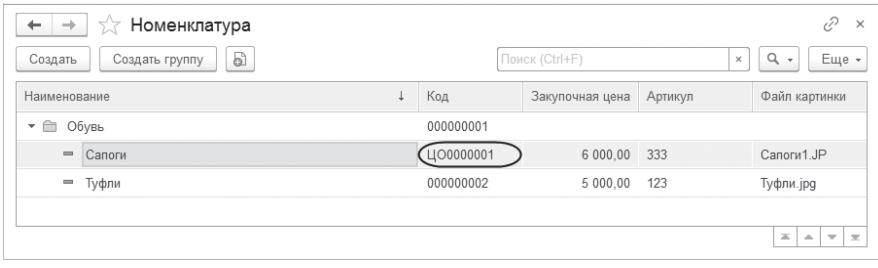


Рис. 3.19. Список справочника «Номенклатура» в подчиненном узле обмена «Оптовый»

Теперь установим значение константы Префикс номеров как «ОП». Добавим нового контрагента. Его код будет начинаться с префикса «ОП». В списке узлов плана обмена Магазины выберем узел, которому будут передаваться данные (Центральный офис) и нажмем кнопку Записать изменения. После этого откроем центральную базу, в списке узлов плана обмена Магазины выберем узел, из которого данные были выгружены (Оптовый), нажмем кнопку Прочитать изменения и укажем файл архива, в который были записаны изменения от подчиненного узла с автоматически сформированным именем Message_Оптовый_ЦентральныйОфис.

Таким образом, изменения данных можно производить в любом из узлов плана обмена и обмениваться данными в обе стороны. Но, в отличие от универсального обмена данными, в распределенных информационных базах обмениваться данными можно только между парами узлов «главный –

подчиненный», а не между подчиненными узлами. Изменения конфигурации возможны только в корневом узле и должны быть переданы по всем уровням иерархии.

Если изменить и конфигурацию, и данные и записать эти изменения, то они будут переданы в одном сообщении обмена. В этом случае прочитать сообщение обмена в подчиненном узле нужно дважды. Сначала будут получены изменения конфигурации. Затем нужно в режиме Конфигуратор обновить конфигурацию и выполнить чтение изменений еще раз. После этого будут получены измененные данные.

Сценарии обмена данными в распределенной информационной базе

Обработчики событий `ПриОтправкеДанныхПодчиненному`, `ПриОтправкеДанныхГлавному`, `ПриПолученииДанныхОтПодчиненного`, `ПриПолученииДанныхОтГлавного` в модуле плана обмена позволяют достаточно гибко управлять обменом данными в распределенной информационной базе. С использованием этих обработчиков может быть построено большое разнообразие сценариев обмена данными. В этом разделе в качестве примера будет рассмотрена организация нескольких сценариев.

Поведение по умолчанию

Данный сценарий является наиболее простым и соответствует поведению распределенной информационной базы по умолчанию. Для этого сценария характерно следующее:

- каждое изменение элемента данных, произведенное в любом из узлов распределенной информационной базы, стремится распространиться по всем узлам;
- разрешение коллизий производится на основании отношения узлов «главный – подчиненный».

Для реализации такого сценария все обработчики не должны изменять значения переданных им параметров, или же обработчики могут быть не определены вовсе.

Распределение данных по подчиненным узлам

Данный сценарий подразумевает, что для некоторых элементов данных, для которых он реализуется, выполняется следующее:

- вся совокупность элементов данных присутствует в главном узле;
- присутствие того или иного элемента данных в том или ином подчиненном узле определяется на основе сравнения значений некоторых

реквизитов элемента данных с реквизитами подчиненного узла плана обмена;

- разрешение коллизий производится на основании отношения узлов «главный – подчиненный».

Для реализации данного сценария нужно обеспечить, чтобы при записи сообщения обмена данными в главном узле в сообщение не попадали элементы данных, которые не должны присутствовать в подчиненном узле.

Кроме того, если значения реквизитов элемента данных могут быть изменены в подчиненном узле, то необходимо обеспечить, чтобы при получении сообщения обмена данными в главном узле производилась регистрация изменений для тех объектов, которых в соответствии со значениями их реквизитов в подчиненном узле быть не должно.

Для более детального рассмотрения примера предположим, что в качестве типа элементов данных, для которых реализуется сценарий, выступает документ РасходнаяНакладная. У данного документа имеется реквизит Склад типа `СправочникСсылка.Склады`. Обмен данными организован в соответствии с планом обмена Магазины. У этого плана обмена также определен реквизит Склад типа `СправочникСсылка.Склады`. В соответствии с этим планом обмена организована распределенная информационная база, в которой корневым узлом является центральный офис, а его подчиненными узлами – склады. У каждого из подчиненных узлов плана обмена значение реквизита Склад установлено так, чтобы обозначать, какому складу соответствует этот узел. Все документы РасходнаяНакладная должны присутствовать в корневом узле, а условием присутствия документов в подчиненных узлах является равенство значений реквизитов Склад в документе и узле плана обмена.

В этом случае, для того чтобы документы РасходнаяНакладная не попадали в те подчиненные узлы, куда они попадать не должны, обработчик события `ПриОтправкеДанныхПодчиненному` в модуле плана обмена должен иметь следующий вид (листинг 3.26).

Листинг 3.26. Процедура «ПриОтправкеДанныхПодчиненному()»

```
Процедура ПриОтправкеДанныхПодчиненному(ЭлементДанных, ОтправкаЭлемента)

    ТипДанных = ТипЗнч(ЭлементДанных);
    Если ТипДанных = Тип("ДокументОбъект.РасходнаяНакладная") Тогда
        Если ЭлементДанных.Склад <> Склад Тогда
            ОтправкаЭлемента = ОтправкаЭлементаДанных.Удалить;
        КонецЕсли;
    КонецЕсли;

КонецПроцедуры
```

В приведенном примере обработчика анализируется тип элемента данных, и если он равен ДокументОбъект.РасходнаяНакладная, то значение реквизита Склад документа сравнивается со значением реквизита Склад узла плана обмена. Если значения реквизитов равны, то значение параметра ОтправкаЭлемента можно не изменять (при вызове параметр имеет значение Авто). При этом в сообщение будет помещено XML-представление документа. Если же значения реквизитов не равны, то параметру ОтправкаЭлемента присваивается значение Удалить. В этом случае в сообщение будет помещено XML-представление объекта УдалениеОбъекта, проинициализированного ссылкой на соответствующий документ РасходнаяНакладная.

Может показаться странным, что в случае неравенства значений реквизитов Склад параметру ОтправкаЭлемента присваивается значение Удалить, а не Игнорировать, так как в случае значения Удалить XML-представление объекта УдалениеОбъекта будет помещаться в сообщения, отправляемые всем подчиненным узлам, кроме того узла, в который будет отправлен сам документ. Таким образом, в значительной части случаев УдалениеОбъекта будет отправлено тем узлам, где документа, который требуется удалить, никогда не было. Это действительно так, но в данном примере рассмотрен наиболее общий случай. Если же, например, известно, что значение реквизита Склад документа РасходнаяНакладная может быть установлено только при создании документа и в дальнейшем не может быть изменено, то параметру ОтправкаЭлемента в данном обработчике действительно могло бы быть присвоено значение Игнорировать.

Если же значение реквизита Склад документа РасходнаяНакладная может быть изменено в подчиненном узле, то в модуле плана обмена необходимо определить обработчик события ПриПолученииДанныхОтПодчиненного следующего вида (листинг 3.27).

Листинг 3.27. Процедура «ПриПолученииДанныхОтПодчиненного()»

```
Процедура ПриПолученииДанныхОтПодчиненного(ЭлементДанных, ПолучениеЭлемента, ОтправкаНазад)

    ТипДанных = ТипЗнч(ЭлементДанных);
    Если ТипДанных = Тип("ДокументОбъект.РасходнаяНакладная") Тогда
        Если ЭлементДанных.Склад <> Склад Тогда
            ПолучениеЭлемента = ПолучениеЭлементаДанных.Игнорировать;
            ОтправкаНазад = Истина;
        КонецЕсли;
    КонецЕсли;

КонецПроцедуры
```

В приведенном примере обработчика анализируется тип элемента данных, и если он равен ДокументОбъект.РасходнаяНакладная, то значение реквизита Склад документа сравнивается со значением реквизита Склад узла плана обмена. Если значения реквизитов равны, то значения параметров ПолучениеЭлемента и ОтправкаНазад можно не изменять, обеспечив этим поведение по умолчанию при приеме элемента данных. Если же значения реквизитов не равны, то параметру ОтправкаНазад присваивается значение Истина. Тем самым гарантируется, что изменения документа будут зарегистрированы и при отправке сообщения подчиненному узлу будет отправлено УдалениеОбъекта, если, конечно, реквизит Склад документа не будет изменен в главном узле так, что он окажется равен значению реквизита Склад соответствующего узла плана обмена.

Если же значение реквизита Склад документа РасходнаяНакладная не может быть изменено после создания документа, то обработчик ПриПолучении-ДанныхОтПодчиненного можно не определять.

Разрешение коллизий

На основе отношения «главный – подчиненный» в распределенной информационной базе организована типовая процедура разрешения коллизий, автоматически выполняемая при приеме сообщения. Считается, что изменение элемента данных, произведенное в главном узле, имеет высший приоритет по отношению к изменению, произведенному в подчиненном узле. Таким образом, если сообщение, пришедшее от подчиненного узла, содержит элемент данных, изменения которого зарегистрированы в базе главного узла для этого подчиненного узла, то никаких действий предпринято не будет, то есть этот элемент данных не будет помещен в базу данных и запись регистрации изменений не будет удалена.

Если сообщение, пришедшее от главного узла, содержит элемент данных, изменения которого зарегистрированы в базе подчиненного узла для главного узла, то элемент данных будет записан в базу данных, а запись регистрация изменения будет удалена.

В случае если данный сценарий не устраивает, можно реализовать прямо противоположный сценарий – при котором, например, принимаются изменения из нижестоящей базы.

Данный сценарий подразумевает, что для некоторых элементов данных, для которых он реализуется, выполняется следующее:

- каждое изменение элемента данных, произведенное в любом из узлов распределенной информационной базы, стремится распространиться по всем узлам;

- разрешение коллизий производится на основании отношения узлов «главный – подчиненный», но более высокий приоритет имеет подчиненный узел.

Для рассмотрения данного случая воспользуемся приведенным выше примером с документом РасходнаяНакладная и планом обмена Магазины.

В данном случае требуется определить обработчики событий `ПриПолученииДанныхОтПодчиненного` и `ПриПолученииДанныхОтГлавного` в модуле плана обмена. Обработчик `ПриПолученииДанныхОтПодчиненного` будет иметь следующий вид (листинг 3.28).

Листинг 3.28. Процедура «ПриПолученииДанныхОтПодчиненного()»

```
Процедура ПриПолученииДанныхОтПодчиненного(ЭлементДанных, ПолучениеЭлемента, ОтправкаНазад)

    ТипДанных = ТипЗнч(ЭлементДанных);
    Если ТипДанных = Тип("ДокументОбъект.РасходнаяНакладная") Тогда
        ПолучениеЭлемента = ПолучениеЭлементаДанных.Принять;
    КонецЕсли;

КонецПроцедуры
```

Приведенный обработчик весьма прост: проверяется тип элемента данных, и если элемент данных относится к интересующему нас типу, то параметру `ПолучениеЭлемента` присваивается значение `Принять`, что приводит к безусловному приему элемента данных, независимо от того, зарегистрированы его изменения или нет.

Обработчик события `ПриПолученииДанныхОтГлавного` выглядит следующим образом (листинг 3.29).

Листинг 3.29. Процедура «ПриПолученииДанныхОтГлавного()»

```
Процедура ПриПолученииДанныхОтГлавного(ЭлементДанных, ПолучениеЭлемента, ОтправкаНазад)

    ТипДанных = ТипЗнч(ЭлементДанных);
    Если ТипДанных = Тип("ДокументОбъект.РасходнаяНакладная") Тогда
        Если ПланыОбмена.ИзменениеЗарегистрировано(Ссылка, ЭлементДанных) Тогда
            ПолучениеЭлемента = ПолучениеЭлементаДанных.Игнорировать;
        КонецЕсли
    КонецЕсли;

КонецПроцедуры
```

Этот обработчик несколько сложнее. Если элемент данных относится к интересующему нас типу, то производится проверка – зарегистрированы ли изменения элемента данных для узла-отправителя сообщения. Если изменения зарегистрированы, то параметру `ПолучениеЭлемента` присваивается значение `Игнорировать`. В результате прочитанный элемент данных

не записывается в базу данных, а регистрация изменений сохраняется, что позволит поместить элемент данных в сообщение, отправляемое главному узлу.

Доработка примера обмена данными в распределенной информационной базе

В соответствии с вышесказанным доработаем наш пример.

Во-первых, до сих пор мы никак не отслеживали факт соответствия значения реквизита Склад передаваемых/принимаемых данных значению реквизита Склад узла обмена. Поэтому при получении данных в подчиненном узле Оптовый (в нашем примере – путем создания начального образа) из центральной базы туда попали данные, относящиеся розничному складу (например, приходные накладные), хотя по условию нашей задачи этих данных там быть не должно (рис. 3.20).

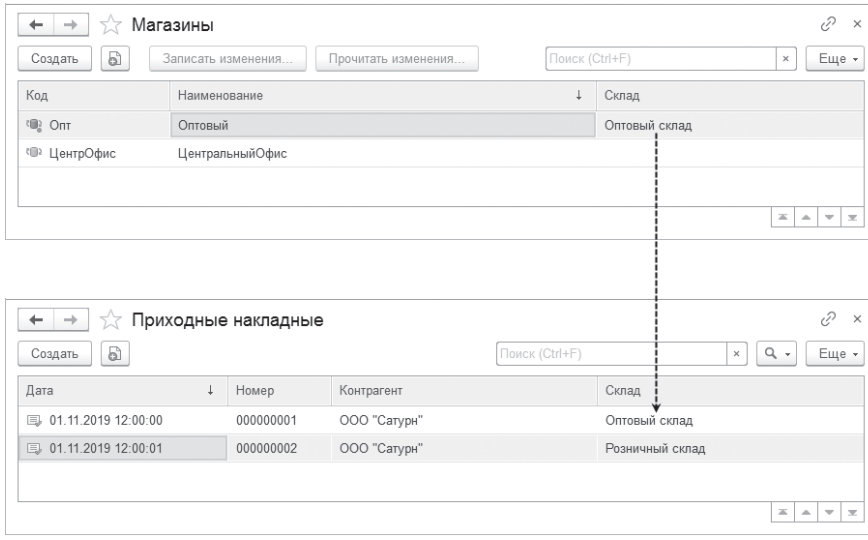


Рис. 3.20. Данные, полученные в узле обмена «Оптовый»

Во-вторых, мы реализуем обработку для выполнения программного обмена данными через установленный каталог обмена.

А также покажем нестандартный вариант разрешения коллизий, при котором изменения, сделанные в подчиненном узле, будут иметь более высокий приоритет над главным.

Распределение данных по подчиненным узлам

Чтобы устранить первую проблему несоответствия складов, поместим в модуле плана обмена Магазины обработчики событий ПриПолучении-ИданныхОтПодчиненного и ПриОтправкеДанныхПодчиненному.

Обработчик ПриОтправкеДанныхПодчиненному заполним следующим образом (листинг 3.30).

Листинг 3.30. Процедура «ПриОтправкеДанныхПодчиненному»

```
Процедура ПриОтправкеДанныхПодчиненному(ЭлементДанных, ОтправкаЭлемента)

    // Определить Склад, которому принадлежат отправляемые данные (если он есть).
    СкладДанных = ОпределитьСклад(ЭлементДанных);

    // Отправить данные, у которых Склад соответствует складу подчиненного узла плана обмена.
    Если СкладДанных <> Неопределено Тогда

        Если СкладДанных <> Склад Тогда

            ОтправкаЭлемента = ОтправкаЭлементаДанных.Удалить;

        КонецЕсли;

    КонецЕсли;

КонецПроцедуры
```

Это обработчик был подробно рассмотрен выше в листинге 3.26, за исключением функции ОпределитьСклад() для определения склада элемента данных обмена, который содержится в параметре ЭлементДанных. Код функции будет приведен в листинге 3.32.

Обработчик события ПриПолученииДанныхОтПодчиненного заполним следующим образом (листинг 3.31).

Листинг 3.31. Процедура «ПриПолученииДанныхОтПодчиненного»

```
Процедура ПриПолученииДанныхОтПодчиненного(ЭлементДанных, ПолучениеЭлемента, ОтправкаНазад)

    // Определить Склад, которому принадлежат получаемые данные.
    СкладДанных = ОпределитьСклад(ЭлементДанных);

    // Получить данные, у которых Склад соответствует складу подчиненного узла плана обмена.
    Если СкладДанных <> Неопределено Тогда

        Если СкладДанных <> Склад И Склад <> Справочники.Склады.ПустаяСсылка() Тогда

            ПолучениеЭлемента = ПолучениеЭлементаДанных.Игнорировать;
            ОтправкаНазад = Истина;

        КонецЕсли;

    КонецЕсли;

КонецПроцедуры
```

Это обработчик был подробно рассмотрен выше в листинге 3.27, поэтому мы не будем здесь еще раз это повторять.

Кроме того, поместим в модуле плана обмена функцию `ОпределитьСклад()`, которая будет возвращать ссылку на склад переданного элемента данных или возвращать `Неопределено`, если реквизит Склад у элемента данных отсутствует (листинг 3.32).

Листинг 3.32. Функция «ОпределитьСклад»

```
Функция ОпределитьСклад(ЭлементДанных)

    СкладДанных = Неопределено;
    Если ТипЗнч(ЭлементДанных) = Тип("СправочникОбъект.Склады") Тогда

        СкладДанных = ЭлементДанных.Ссылка;

    ИначеЕсли ТипЗнч(ЭлементДанных) = Тип("ДокументОбъект.ПриходнаяНакладная")
        Или ТипЗнч(ЭлементДанных) = Тип("ДокументОбъект.РасходнаяНакладная") Тогда

        СкладДанных = ЭлементДанных.Склад;

    ИначеЕсли ТипЗнч(ЭлементДанных) = Тип("РегистрНакопленияНаборЗаписей.УчетНоменклатуры") Тогда

        // Определить склад набора записей регистра накопления
        // УчетНоменклатуры по значению склада первой записи регистра.
        Если ЭлементДанных.Количество() > 0 Тогда

            СкладДанных = ЭлементДанных[0].Склад;

        КонецЕсли;

    КонецЕсли;

    Возврат СкладДанных;

КонецФункции
```

Кроме ссылки на реквизит Склад у документов ПриходнаяНакладная, РасходнаяНакладная и ссылки на измерение Склад у набора записей регистра накопления УчетНоменклатуры функция также возвращает ссылку на сам элемент справочника Склады, так как в узле обмена должен присутствовать только «свой» элемент справочника.

Теперь создадим еще раз начальный образ центральной базы для узла обмена Оптовый, как описано в разделе «Создание начального образа». При переносе элементов данных в начальный образ для каждого элемента данных вызывается обработчик события `ПриОтправкеДанныхПодчиненному`, поэтому данные, не соответствующие оптовому складу, не будут перенесены.

В результате в базу подчиненного узла помимо «общих» (не имеющих привязки к складу) данных будут перенесены только те данные, склад которых совпадает со складом узла обмена (рис. 3.21).

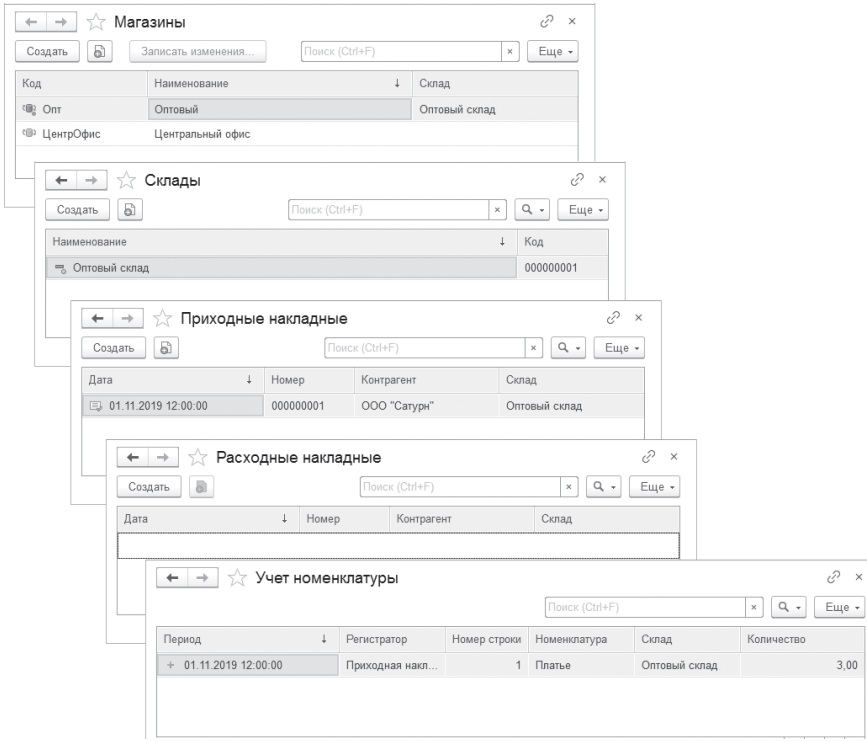


Рис. 3.21. Данные, перенесенные в узел обмена «Оптовый»

Если же в подчиненном узле ввести, например, расходную накладную с «неправильным» складом, то при получении изменений в главном узле в соответствии с обработчиком события `ПриПолученииДанныхОтПодчиненного` этот документ будет проигнорирован. То есть в базу данных он записан не будет, но изменения этого документа будут зарегистрированы для подчиненного узла. И при следующей отправке данных этому узлу в соответствии с обработчиком события `ПриОтправкеДанныхПодчиненному` ему будет послано удаление «неправильного» документа.

Обработка для выполнения обмена

Все описанные выше действия по обмену данными в распределенной информационной базе можно выполнить также программно. Покажем это на примере специальной обработки, с помощью которой для выбранного узла будут реализовываться те же самые действия, которые ранее мы выполняли интерактивно.

Итак, в конфигураторе центральной базы добавим обработку ОбменСМагазинами и откроем ее основную форму.

Добавим в форму реквизит Магазин типа `ПланОбменаСсылка.Магазины`, в котором будет содержаться ссылка на выбранный узел обмена. А также добавим в форму команды: ЗаписатьИзменения, ПрочитатьИзменения и СоздатьНачальныйОбраз. Перетащим эти команды и реквизит в окно элементов формы.

У всех трех кнопок, соответствующих командам, снимем флажок у свойства Доступность. Это делается для того, чтобы действия по созданию начального образа, записи и чтению изменений обмена становились возможны только в случае выбора в поле Магазин непредопределенного узла, который не совпадает с узлом текущей информационной базы.

Для того чтобы обеспечить такое поведение кнопок, поместим в модуле формы функцию `ПолучитьДоступностьПоУзлу()`, в которую передается ссылка на выбранный узел (листинг 3.33).

Листинг 3.33. Функция «ПолучитьДоступностьПоУзлу»

```
&НаСервереБезКонтекста
Функция ПолучитьДоступностьПоУзлу(Узел)

    ДоступностьКнопок = Ложь;

    // Определить наличие установленного непредопределенного узла.
    Если Узел <> ПланыОбмена.Магазины.ПустаяСсылка()
      И Узел <> ПланыОбмена.Магазины.ЭтотУзел()
      И Узел.ПолучитьОбъект() <> Неопределено Тогда

        ДоступностьКнопок = Истина;

    КонецЕсли;

    Возврат ДоступностьКнопок;

КонецФункции
```

Эта функция будет возвращать `Истина`, в случае если ссылка, содержащаяся в параметре `Узел`, не пустая и не равна ссылке на предопределенный узел, соответствующий текущей информационной базе. Для получения ссылки на текущий узел используется метод менеджера узла плана обмена `ЭтотУзел()`. В противном случае функция вернет `Ложь`.

После этого создадим обработчик события `ПриИзменении` для поля формы Магазин, в котором доступность кнопок будет зависеть от значения, возвращенного описанной ранее функцией (листинг 3.34).

Листинг 3.34. Обработчик события «ПриИзменении» поля «Магазин»

```
&НаКлиенте
Процедура МагазинПриИзменении(Элемент)

    ДоступностьКнопок = ПолучитьДоступностьПоУзлу(Магазин);

    // Установить доступность кнопок в зависимости от установленного узла обмена.
    Элементы.СоздатьНачальныйОбраз.Доступность = ДоступностьКнопок;
    Элементы.ЗаписатьИзменения.Доступность = ДоступностьКнопок;
    Элементы.ПрочитатьИзменения.Доступность = ДоступностьКнопок;

КонецПроцедуры
```

Теперь создадим обработчик команды СоздатьНачальныйОбраз и заполним следующим образом (листинг 3.35).

Листинг 3.35. Обработчик команды «СоздатьНачальныйОбраз»

```
&НаКлиенте
Процедура СоздатьНачальныйОбраз(Команда)

    Диалог = Новый ДиалогВыбораФайла(РежимДиалогаВыбораФайла.ВыборКаталога);
    Диалог.Заголовок = "Выбор каталога для создания начального образа";
    Диалог.Показать(Новый ОписаниеОповещения("ПослеВыбораКаталога", ЭтотОбъект));

КонецПроцедуры
```

В этом обработчике мы вызываем диалог выбора каталога, в который будет помещен начальный образ информационной базы, и показываем его пользователю с помощью немодального метода `Показать()`. В этот метод мы передаем описание оповещения, указывающее на экспортную процедуру `ПослеВыбораКаталога()`, которая будет вызвана после того, как пользователь выберет каталог (листинг 3.36).

Листинг 3.36. Процедура «ПослеВыбораКаталога»

```
&НаКлиенте
Процедура ПослеВыбораКаталога(ВыбранныеФайлы, Параметры) Экспорт

    Если ВыбранныеФайлы = Неопределено Тогда
        Возврат;
    КонецЕсли;

    Каталог = ВыбранныеФайлы[0];
    СоздатьНачальныйОбразНаСервере(Магазин, Каталог);
```

```
    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = "Создание начального образа узла завершено.";
    Сообщение.Сообщить();

КонецПроцедуры
```

В этом обработчике оповещения, в случае если каталог выбран, мы получаем путь к каталогу как первый элемент массива, содержащийся в параметре `ВыбранныеФайлы`, и вызываем процедуру `СоздатьНачальныйОбразНаСервере()`, в которую передаем этот каталог и выбранный узел обмена, содержащийся в реквизите Магазин (листинг 3.37).

Листинг 3.37. Процедура «СоздатьНачальныйОбразНаСервере»

```
&НаСервереБезКонтекста
Процедура СоздатьНачальныйОбразНаСервере(Узел, КаталогСоединения)

    ПланыОбмена.СоздатьНачальныйОбраз(Узел, "File =""" + КаталогСоединения + """");

КонецПроцедуры
```

В этой процедуре вызывается метод `СоздатьНачальныйОбраз()` объекта `ПланыОбменаМенджер`, с помощью которого и создается начальный образ для подчиненного узла распределенной информационной базы. В первом параметре мы передаем в этот метод ссылку на узел, для которого мы хотим создать начальный образ, а во втором – строку соединения с создаваемой информационной базой.

Теперь создадим обработчик команды ЗаписатьИзменения и заполним следующим образом (листинг 3.38).

Листинг 3.38. Обработчик команды «ЗаписатьИзменения»

```
&НаКлиенте
Процедура ЗаписатьИзменения(Команда)

    ЗаписатьИзмененияНаСервере(Магазин);

    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = "Запись изменений завершена.";
    Сообщение.Сообщить();

КонецПроцедуры
```

В этом обработчике мы вызываем процедуру `ЗаписатьИзмененияНаСервере()`. В эту процедуру мы передаем выбранный узел обмена, содержащийся в реквизите Магазин (листинг 3.39), и выводим сообщение пользователю об успешной записи изменений, зарегистрированных для этого узла.

Листинг 3.39. Процедура «ЗаписатьИзмененияНаСервере»

```
&НаСервереБезКонтекста
Процедура ЗаписатьИзмененияНаСервере(Узел)

	// Сформировать полное имя файла обмена.
	Каталог = Константы.КаталогОбмена.Получить();
	ИмяФайла = Каталог + ?(Прав(Каталог, 1) = "\","", "\") + "Message_"
		+ СокрЛП(ПланыОбмена.Магазины.ЭтотУзел().Код) + "_" +
		СокрЛП(Узел.Код) + ".xml";

	// Создать и проинициализировать объект ЗаписьXML.
	ЗаписьXML = Новый ЗаписьXML;
	ЗаписьXML.ОткрытьФайл(ИмяФайла);

	// Создать объект ЗаписьСообщенияОбмена и начать запись сообщения.
	ЗаписьСообщения = ПланыОбмена.СоздатьЗаписьСообщения();
	ЗаписьСообщения.НачатьЗапись(ЗаписьXML, Узел);

	// Записать содержимое тела сообщения обмена данными распределенной ИБ.
	ПланыОбмена.ЗаписатьИзменения(ЗаписьСообщения);

	// Закончить запись сообщения и запись XML.
	ЗаписьСообщения.ЗакончитьЗапись();
	ЗаписьXML.Закрыть();

КонецПроцедуры
```

В этой процедуре сначала мы формируем имя файла, в который будет записано сообщение обмена.

Для упрощения примера мы будем обмениваться сообщениями через заранее известный каталог обмена, путь к которому хранится в константе Каталог обмена. Значение этой константы (например, `e:\Exchange`) наряду с префиксом номеров можно задать в форме настроек информационной базы (см. рис. 3.15).

Имена файлов сообщений стандартизованы и имеют вид `Message_<КодУзлаОтправителя>_<КодУзлаПолучателя>.xml`. В качестве узла-отправителя сообщения обмена здесь выступает предопределенный узел информационной базы, полученный с помощью метода `ЭтотУзел()`, а в качестве узла-получателя – узел, ссылка на который содержится в параметре `Узел`.

Для доступа к файлу обмена мы создаем объект `ЗаписьXML`. Затем с помощью метода `СоздатьЗаписьСообщения()` объекта `ПланыОбменаМенджер` создаем объект `ЗаписьСообщенияОбмена`, с помощью которого будет выполняться запись сообщений обмена данными. При вызове метода этого объекта `НачатьЗапись()` во втором параметре мы указываем, для какого узла будет производиться выгрузка изменений.

После этого мы вызываем метод ЗаписатьИзменения() объекта ПланыОбменаМенджер, который и записывает изменения, предназначенные для передачи в выбранный узел, в сообщение обмена с ранее сформированным именем.

В заключение мы заканчиваем запись сообщения обмена и закрываем файл.

Теперь создадим обработчик команды ПрочитатьИзменения и заполним следующим образом (листинг 3.40).

Листинг 3.40. Обработчик команды «ПрочитатьИзменения»

```
&НаКлиенте
Процедура ПрочитатьИзменения(Команда)

    ПрочитатьИзмененияНаСервере(Магазин);

    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = "Чтение изменений завершено.";
    Сообщение.Сообщить();

КонецПроцедуры
```

В этом обработчике мы вызываем процедуру ПрочитатьИзменения-НаСервере(). В эту процедуру мы передаем выбранный узел обмена, содержащийся в реквизите Магазин (листинг 3.41), и выводим сообщение пользователю об успешном получении изменений, от этого узла.

Листинг 3.41. Процедура «ПрочитатьИзмененияНаСервере»

```
&НаСервереБезКонтекста
Процедура ПрочитатьИзмененияНаСервере(Узел)

    // Сформировать полное имя файла обмена.
    Каталог = Константы.КаталогОбмена.Получить();
    ИмяФайла = Каталог + ?(Прав(Каталог, 1) = "\", "", "\") + "Message_" + СокрЛП(Узел.Код) + "_" +
        СокрЛП(ПланыОбмена.Магазины.ЭтотУзел().Код) + ".xml";

    // Создать и проинициализировать объект ЧтениеXML.
    ЧтениеXML = Новый ЧтениеXML;
    ЧтениеXML.ОткрытьФайл(ИмяФайла);

    // Создать объект ЧтениеСообщенияОбмена и начать чтение сообщения.
    ЧтениеСообщения = ПланыОбмена.СоздатьЧтениеСообщения();
    ЧтениеСообщения.НачатьЧтение(ЧтениеXML);

    // Прочитать содержимое тела сообщения.
    ПланыОбмена.ПрочитатьИзменения(ЧтениеСообщения);

    // Закончить чтение сообщения и чтение XML.
    ЧтениеСообщения.ЗакончитьЧтение();
    ЧтениеXML.Закрыть();

КонецПроцедуры
```

В этой процедуре мы формируем имя файла, из которого будет прочитано сообщение обмена. Это делается по аналогии с процедурой `ЗаписатьИзмененияНаСервере()`, приведенной в листинге 3.39. Только здесь узел-отправитель и узел-получатель сообщения меняются местами.

Для доступа к файлу обмена мы создаем объект `ЧтениеXML`. Затем с помощью метода `СоздатьЧтениеСообщения()` объекта `ПланыОбменаМенеджер` создаем объект `ЧтениеСообщенияОбмена`, с помощью которого выполняется чтение сообщений обмена данными.

После этого мы вызываем метод `ПрочитатьИзменения()` объекта `ПланыОбменаМенеджер`, в котором и происходит чтение изменений, полученных от выбранного узла, из сообщения обмена с ранее сформированным именем.

В заключение мы заканчиваем чтение сообщения обмена и закрываем файл.

Проверим, как это работает. Запустим «1С:Предприятие» и в форме настроек информационной базы зададим путь к каталогу обмена (`e:\Exchange`), который будет храниться в константе Каталог обмена (см. рис. 3.15).

Затем откроем список узлов плана обмена Магазины. Добавим в этот список еще одну запись с кодом Розн и наименованием Розничный. В поле Склад выберем значение Розничный склад. Затем откроем обработку Обмен с магазинами, в поле Магазин выберем узел обмена Розничный и нажмем кнопку Создать начальный образ (рис. 3.22).

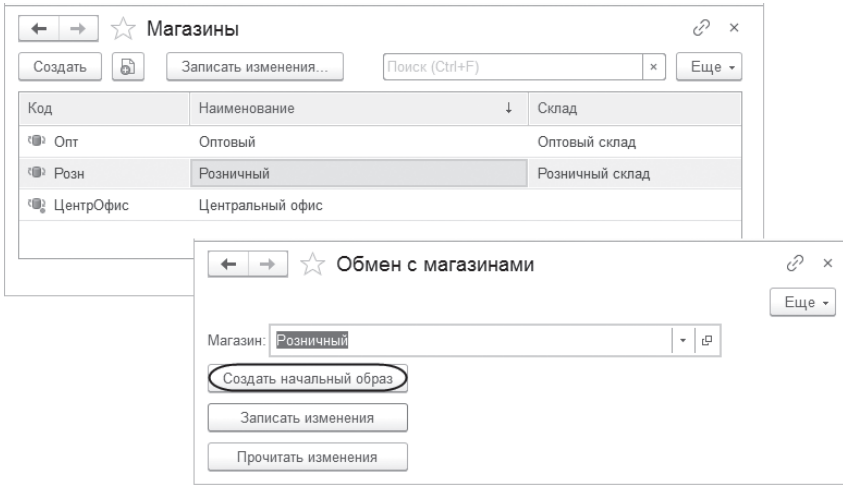


Рис. 3.22. Программное создание начального образа для узла «Розничный»

Таким образом, мы программно создадим подчиненный узел обмена путем создания начального образа из нашей центральной базы.

Откроем базу подчиненного узла Розничный. Там мы увидим все данные, перенесенные из центральной базы при создании начального образа. При программном обмене также будет поддерживаться стратегия распространения данных по подчиненным узлам, описанная ранее в разделе «Распределение данных по подчиненным узлам». В результате в базу розничного узла помимо «общих» данных (не привязанных к складу) попадут только те данные, склад которых совпадает со складом узла обмена. Поскольку такое поведение реализовано в модуле плана обмена Магазины, то оно «работает» независимо от способа обмена данными.

Теперь установим значение константы Префикс номеров как «РЗ» и зададим путь к каталогу обмена (`e:\Exchange`). Добавим новую номенклатуру. Ее код будет начинаться с префикса «РЗ». После этого откроем обработку Обмен с магазинами, в поле Магазин выберем узел, которому будут передаваться данные (Центральный офис), и нажмем кнопку Записать изменения (рис. 3.23).

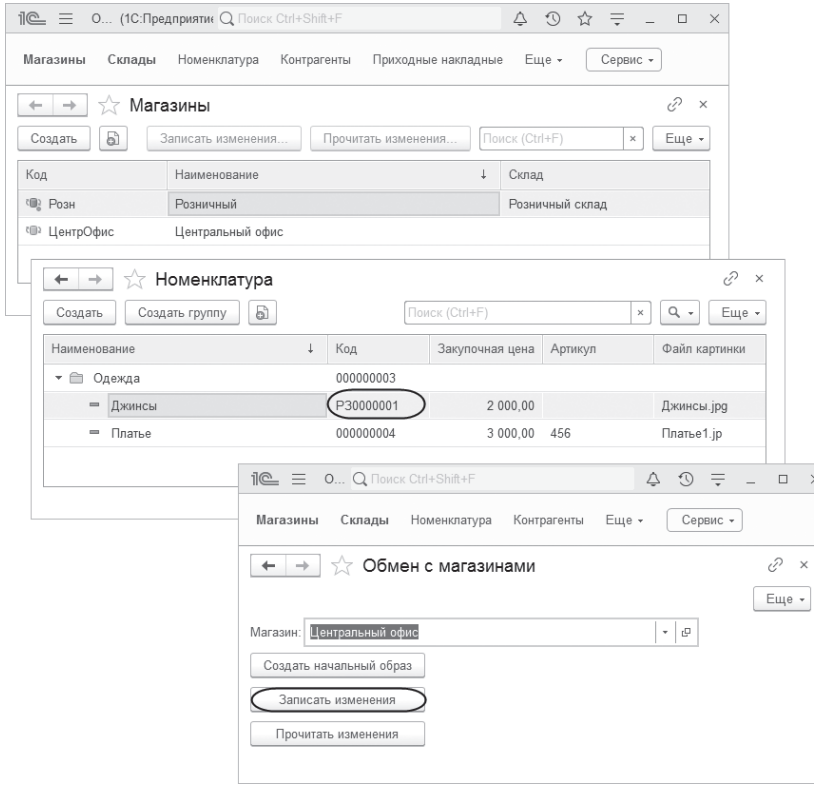


Рис. 3.23. Запись изменений в подчиненном узле для центральной базы

Чтобы получить эти изменения, в центральной базе откроем обработку Обмен с магазинами, в поле Магазин выберем узел, из которого данные были выгружены (Розничный), и нажмем кнопку Прочитать изменения. В результате новая номенклатура, добавленная в подчиненном узле, появится в центральной базе.

При этом при записи изменений для центральной базы в каталоге обмена появится файл сообщения обмена с именем Message_Розн_ЦентрОфис.xml. Из этого файла изменения от розничного узла будут прочитаны в центральную базу. Выбор файла сообщения пользователем не понадобится, поскольку имена файлов стандартизованы, а каталог обмена хранится в константе информационной базы.

Разрешение коллизий

До сих пор мы обменивались только вновь создаваемыми данными, поэтому у нас не возникало ситуаций, когда одни и те же данные изменялись бы сразу несколькими участниками обмена. На самом деле вполне возможна ситуация, когда, например, в один и тот же существующий документ вносятся изменения и в центральной базе, и в подчиненном узле. Такая ситуация называется «коллизией».

В результате, если ничего не предпринимать, после обмена данными в документе останутся только те изменения, которые были внесены в центральной базе. Изменения документа, сделанные в подчиненном узле, будут проигнорированы.

Это стандартный вариант разрешения коллизий, основанный на том, что главный узел обмена (в нашем примере это центральная база) имеет более высокий приоритет (его изменения более важны), чем подчиненный. Такой сценарий обмена данными в распределенной информационной базе заложен изначально и не требует никакого программирования.

Но бывают случаи, когда требуется реализовать какой-то особенный сценарий обмена. Например, вся нормативно-справочная информация ведется в центральной базе, а расходные и приходные накладные составляются в подчиненных узлах.

В этом случае коллизия должна быть разрешена ровно наоборот – когда изменения подчиненного узла имеют более высокий приоритет над главным. Но это касается только документов и их движений в регистрах. Обмен всеми остальными объектами происходит «по правилам».

Для решения данной задачи поместим в модуле плана обмена Магазины обработчик события `ПриПолученииДанныхОтГлавного` и дополним обработчик `ПриПолученииДанныхОтПодчиненного`.

Эти обработчики были подробно рассмотрены выше в листингах 3.28, 3.29, за исключением функции `ПринятьИзменения()` для определения типа элемента данных обмена, который содержится в параметре `ЭлементДанных`. Эта функция будет возвращать `Истина`, в случае если элемент данных имеет тип расходной или приходной накладных или связанных с ними регистров. В противном случае функция вернет `Ложь`. Поместим эту функцию в модуле плана обмена и заполним следующим образом (листинг 3.42).

Листинг 3.42. Функция «ПринятьИзменения»

```
Функция ПринятьИзменения(ЭлементДанных)

    Прием = Ложь;

    ТипДанных = ТипЗнч(ЭлементДанных);
    Если ТипЗнч(ЭлементДанных) = Тип("ДокументОбъект.ПриходнаяНакладная")
        Или ТипЗнч(ЭлементДанных) = Тип("ДокументОбъект.РасходнаяНакладная")
        Или ТипЗнч(ЭлементДанных) = Тип("РегистрНакопленияНаборЗаписей.УчетНоменклатуры") Тогда

        Прием = Истина;

    КонецЕсли;

    Возврат Прием;

КонецФункции
```

Обработчик `ПриПолученииДанныхОтПодчиненного` дополним следующим фрагментом, выделенным жирным шрифтом (листинг 3.43).

Листинг 3.43. Процедура «ПриПолученииДанныхОтПодчиненного()»

```
Процедура ПриПолученииДанныхОтПодчиненного(ЭлементДанных, ПолучениеЭлемента, ОтправкаНазад)

    Если ПринятьИзменения(ЭлементДанных) Тогда
        ПолучениеЭлемента = ПолучениеЭлементаДанных.Принять;
    КонецЕсли;

    // Определить Склад, которому принадлежат получаемые данные.
    СкладДанных = ОпределитьСклад(ЭлементДанных);

    // Получить данные, у которых Склад соответствует складу подчиненного узла плана обмена.
    …

КонецПроцедуры
```

В этом обработчике определяется, что от подчиненного узла будут безусловно приниматься все изменения интересующих нас документов и их движений. Для остальных элементов обмена никаких действий предпринято не будет, что равносильно стандартному варианту разрешения коллизий.

Обработчик события `ПриПолученииДанныхОтГлавного` заполним следующим образом (листинг 3.44).

Листинг 3.44. Процедура «ПриПолученииДанныхОтГлавного()»

```
Процедура ПриПолученииДанныхОтГлавного(ЭлементДанных, ПолучениеЭлемента, ОтправкаНазад)

    Если ПланыОбмена.ИзменениеЗарегистрировано(Ссылка, ЭлементДанных)
        И ПринятьИзменения(ЭлементДанных) Тогда

        ПолучениеЭлемента = ПолучениеЭлементаДанных.Игнорировать;

    КонецЕсли;

КонецПроцедуры
```

В этом обработчике определяется, что если в базе подчиненного узла зарегистрированы изменения интересующих нас документов и их движений для главного узла, то при получении данных от главного узла соответствующие изменения будут проигнорированы. В результате прочитанный элемент данных не записывается в базу подчиненного узла, а регистрация изменений сохраняется, что позволит поместить этот элемент данных в сообщение, отправляемое главному узлу.

Убедимся, что все работает правильно для всех узлов обмена, причем и программным, и интерактивным способом. Поскольку такой вариант разрешения коллизий реализован в модуле плана обмена Магазины, он будет «работать» независимо от способа обмена данными.

Особенности использования последовательности документов в распределенной информационной базе

Для отслеживания правильного порядка проведения документов в «1С:Предприятии» служит механизм последовательностей документов. Этот механизм позволяет отслеживать порядок проведения документов и производить восстановление этого порядка. Для того чтобы документ оказался в последовательности, он должен зарегистрироваться в последовательности, тогда механизм последовательностей будет учитывать его при своей работе.

При организации работы последовательности документов в распределенной информационной базе нужно учитывать, что участие документа в последовательности имеет смысл только в одном узле распределенной

информационной базы. Это может быть либо узел, в котором документ был создан, либо другой узел, но узел должен быть один. Нарушение данного принципа может привести к различным проблемам в процессе работы с системой – например, невозможности восстановления последовательности документов.

Таким образом, документ, участвующий в последовательности, должен регистрироваться в последовательности только в одном узле информационной базы. Для этого документ должен содержать информацию, по которой на момент записи документа можно сделать вывод, должен он регистрироваться в данном узле или нет. Для того чтобы документ мог это определить, все узлы информационной базы должны иметь уникальную идентификацию.

Например, документ может содержать реквизит, который содержит код узла плана обмена его информационной базы. Коды узлов информационных баз должны быть в этом случае уникальными. Этого можно добиться организационными методами. Основываясь на информации о принадлежности данному узлу распределенной информационной базы, документ должен при записи либо очистить набор записей регистрации в последовательности, либо, наоборот, его заполнить. Тем самым будет достигнута цель регистрации документа в последовательности только в собственном узле информационной базы.

Пример очистки наборов записей в последовательностях (фрагмент кода размещается в обработке проведения документа) приведен в листинге 3.45.

Листинг 3.45. Пример очистки набора записей регистрации документа в последовательности

```
Если Узел<>ПланыОбмена.Расходные.ЭтотУзел().Код Тогда
        Для Каждого НаборПоследовательности Из ПринадлежностьПоследовательностям Цикл
                НаборПоследовательности.Очистить();
        КонецЦикла;
КонецЕсли;
```

Узел – реквизит документа, содержащий код узла, в котором он должен регистрироваться в последовательностях.

Как уже было сказано, документ может участвовать в последовательности только в одном узле информационной базы. Поэтому сами последовательности документов не должны участвовать в обмене данных. Иначе записи регистрации документа будут переданы в другой узел информационной базы, тем самым нарушив принцип регистрации документа в последовательности только в одном узле информационной базы.

Универсальный механизм обмена данными

Универсальный механизм обмена данными может использоваться как вместе с механизмом распределенных информационных баз, так и по отдельности (для решения задач организации обмена данными информационных баз «1С:Предприятия 8» с различными программными системами).

В качестве программных систем, с которыми может быть организован универсальный обмен данными, могут выступать другие информационные базы «1С:Предприятия», в том числе с отличающимися конфигурациями, совершенно другие программные комплексы (в этом и состоит отличие от механизма распределенных информационных баз, в рамках которого организуется обмен между базами данных с идентичными конфигурациями).

К универсальному механизму обмена данными могут быть отнесены:

- средства чтения и записи документов XML,
- XML-сериализация,
- планы обмена.

Функциональность планов обмена была рассмотрена в предыдущих разделах данной главы (см. раздел «Планы обмена»). Возможности базовых средств чтения и записи XML-документов подробно рассматриваются в шестой главе, в разделе «XML-файлы».

Использование возможностей работы с XML-документами

До данного момента рассматривались только механизмы регистрации и удаления регистрации изменений. Рассмотрим пример кода, реализующий (в рамках универсального обмена) полный цикл выгрузки данных (включает в себя выборку данных, для которых были зарегистрированы изменения). При этом в качестве файлов «носителей данных» будут использоваться документы XML.

Следует отметить, что обмен данными может выполняться через файлы других форматов, но записать/прочитать данные с помощью объектов `ЧтениеСообщенияОбмена` и `ЗаписьСообщенияОбмена` без указания в качестве параметра объектов читающих и записывающих XML нельзя (листинг 3.46).

Листинг 3.46. Пример выгрузки данных

```
ЗаписьXML = Новый ЗаписьXML()
ЗаписьXML.ОткрытьФайл(ИмяФайлаСообщения);

Узел = ПланыОбмена.УдаленныеСклады.НайтиПоКоду(КодУзла);
ЗаписьСообщения = ПланыОбмена.СоздатьЗаписьСообщения();
ЗаписьСообщения.НачатьЗапись(ЗаписьXML, Узел);

Выборка = ПланыОбмена.ВыбратьИзменения(Узел, ЗаписьСообщения.НомерСообщения);
Пока Выборка.Следующий() Цикл
        Данные = Выборка.Получить();
        ЗаписатьXML(ЗаписьXML, Данные);
КонецЦикла;

ЗаписьСообщения.ЗакончитьЗапись();
```

Для получения выборки данных, для которых зарегистрированы изменения, используется метод `ВыбратьИзменения()`. Для обхода полученных данных используется метод `Следующий()` выборки. В процессе выборки изменений в записи таблиц регистрации изменений (которые раньше не выгружались и у которых, соответственно, в поле таблицы НомерСообщения находится значение `Null`) проставляется номер сообщения обмена данными, в котором должны передаваться изменения.

Представленный алгоритм выгрузки предполагает, что отправленные сообщения могут быть потеряны по тем или иным причинам. Если потерь сообщения не может быть в принципе (гарантированная доставка), то в конце алгоритма можно поместить следующую строку (листинг 3.47).

Листинг 3.47. Удаление регистрации изменений при гарантированной доставке

```
ПланыОбмена.УдалитьРегистрациюИзменений(Узел, НомерСообщения);
```

В этом случае из таблицы регистрации изменений будут удалены записи, относящиеся к узлу, ссылка на который передана в качестве первого параметра метода. Причем будут удалены не все записи, а только те, в которых номер сообщения равен или меньше того значения, которое передано в качестве второго параметра метода. Это те записи, в которых была произведена первая отправка изменений.

Фрагмент кода, в котором производится чтение сообщения, содержащего измененные данные, выглядит следующим образом (листинг 3.48).

Листинг 3.48. Пример чтения сообщения обмена

```
ЧтениеXML = Новый ЧтениеXML()
ЧтениеXML.ОткрытьФайл(ИмяФайлаСообщения);

ЧтениеСообщения = ПланыОбмена.СоздатьЧтениеСообщения();
ЧтениеСообщения.НачатьЧтение(ЧтениеXML);
ПланыОбмена.УдалитьРегистрациюИзменений(ЧтениеСообщения.Отправитель
                                        , ЧтениеСообщения.НомерПринятого);

Пока ВозможностьЧтенияXML(ЧтениеXML) Цикл
    Данные = ПрочитатьXML(ЧтениеXML);
    Данные.ОбменДанными.Отправитель = ЧтениеСообщения.Отправитель;
    Данные.ОбменДанными.Загрузка = Истина;
    Данные.Записать();
КонецЦикла;

ЧтениеСообщения.ЗакончитьЧтение();
```

В приведенных выше примерах чтения и записи сообщений не учитывалось, что при обмене данными может случиться так, что один и тот же элемент данных будет изменен одновременно в двух обменивающихся данными узлах. В этом случае непонятно, какое из изменений должно быть в конечном счете принято. Такая ситуация называется коллизией.

Один из способов разрешить коллизию – определить, какой из узлов является главным, а какой – подчиненным. При этом должно быть принято изменение, сделанное в главном узле, а изменение, сделанное в подчиненном узле, должно быть отвергнуто.

Для реализации этого при приеме сообщения перед записью данных необходимо установить, зарегистрировано ли изменение этих данных, и в зависимости от роли узла в данной паре «получатель – отправитель» принять решение: записывать или не записывать данные.

Ниже (листинг 3.49) приведен пример реализации стратегии «главный – подчиненный» при чтении сообщения. Предполагается, что для хранения роли узла в плане обмена был определен реквизит Главный, имеющий тип Булево.

Листинг 3.49. Пример реализации стратегии «главный – подчиненный»

```
ЧтениеXML = Новый ЧтениеXML()
ЧтениеXML.ОткрытьФайл(ИмяФайлаСообщения);

ЧтениеСообщения = ПланыОбмена.СоздатьЧтениеСообщения();
ЧтениеСообщения.НачатьЧтение(ЧтениеXML);
ПланыОбмена.УдалитьРегистрациюИзменений(ЧтениеСообщения.Отправитель
                                        , ЧтениеСообщения.НомерПринятого);

Отправитель = ЧтениеСообщения.Отправитель;
```

```
Главный = Отправитель.Главный;

Пока ВозможностьЧтенияXML(ЧтениеXML) Цикл
        Данные = ПрочитатьXML(ЧтениеXML);
        Если Главный ИЛИ (Не ПланыОбмена.ИзменениеЗарегистрировано(Отправитель, Данные)) Тогда
                Данные.ОбменДанными.Отправитель = ЧтениеСообщения.Отправитель;
                Данные.ОбменДанными.Загрузка = Истина;
                Данные.Записать();
        КонецЕсли;
КонецЦикла;

ЧтениеСообщения.ЗакончитьЧтение();
```

Следует обратить внимание на необходимость установки у загружаемого элемента данных свойства `Загрузка` в значение `Истина`. В обработчиках событий записи объекта рекомендуется отслеживать данный режим и не делать каких-либо проверок на корректность заполнения, присутствие каких-либо связанных данных. Это обусловлено тем фактом, что на момент загрузки данного объекта другие объекты могут быть еще не загружены.

Пример реализации универсального обмена

В этом примере мы не только реализуем обмен данными с помощью универсальных механизмов обмена стандартным образом, но и покажем, как организовать обмен данными с различной структурой, как реализовать стратегию распространения данных по конкретным узлам, покажем возможность разрешения коллизий, когда одни и те же данные изменяются одновременно в нескольких узлах обмена. Кроме того, рассмотрим возможность ручной регистрации изменений данных.

ПОДРОБНЕЕ

> Подробнее познакомиться с реализацией универсального обмена данными можно в демонстрационной конфигурации «Примеры обмена», которая прилагается к книге.

В нашей демонстрационной конфигурации существует план обмена УдаленныеСклады. У данного плана обмена имеются следующие реквизиты:

- Склад (тип `СправочникСсылка.Склады`) – ссылка на склад, представляющий данный узел;
- Главный (тип `Булево`) – для реализации стратегии «главный – подчиненный».

Состав плана обмена представлен на рисунке (рис. 3.24).

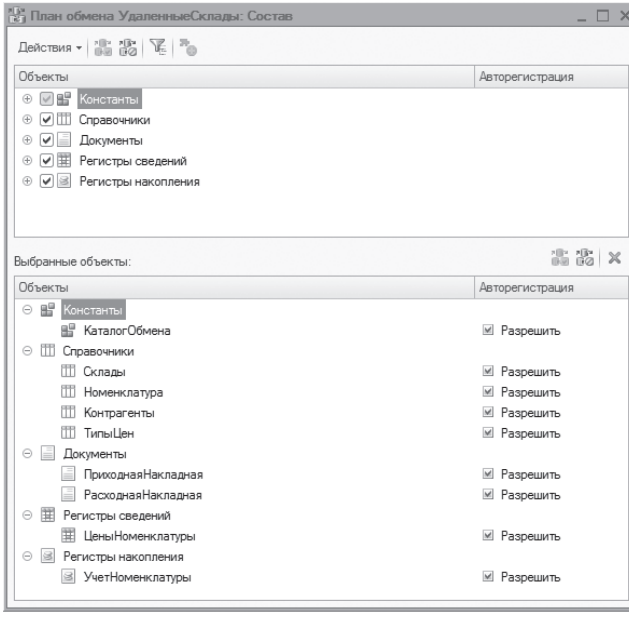


Рис. 3.24. Состав плана обмена «УдаленныеСклады»

Состав плана обмена включает все объекты конфигурации за исключением константы ПрефиксНомеров, которая служит для установки префиксов кодов/номеров объектов, созданных в каждом узле обмена. Подробнее об этом рассказывается в разделе «Подготовка конфигурации к работе в распределенной информационной базе».

Итак, сначала мы создадим форму узла (форму объекта) нашего плана обмена.

При создании нового узла обмена нам нужно указать, какой узел обмена будет главным для текущей информационной базы. Забегая вперед, поясним, что это свойство понадобится нам для разрешения коллизий при получении данных. Кроме того, нам нужно выполнить для узла своего рода начальную синхронизацию всех данных обмена.

Для этого добавим в форму команду ЗарегистрироватьИзменения и перетащим ее в окно элементов формы. Однако кнопка, связанная с этой командой, а также возможность установки флажка Главный должны быть недоступны для предопределенного узла, соответствующего данной информационной базе.

Для этого создадим обработчик события формы `ПриСозданииНаСервере` и заполним его следующим образом (листинг 3.50).

Листинг 3.50. Обработчик события формы «ПриСозданииНаСервере»

```
&НаСервере
Процедура ПриСозданииНаСервере(Отказ, СтандартнаяОбработка)

    Если Объект.ЭтотУзел = Истина Тогда
        Элементы.Главный.Доступность = Ложь;
        Элементы.ФормаЗарегистрироватьИзменения = Ложь;
    КонецЕсли;

КонецПроцедуры
```

В этом обработчике для определения предопределенного узла мы используем свойство узла плана обмена `ЭтотУзел`. Если оно принимает значение `Истина`, значит, это узел, соответствующий нашей информационной базе.

Создадим обработчик команды ЗарегистрироватьИзменения, из которого будет вызываться процедура `ЗарегистрироватьИзмененияНаСервере()`. В эту процедуру будет передаваться ссылка на узел обмена, для которого нужно зарегистрировать изменения (листинг 3.51).

Листинг 3.51. Обработчик команды «ЗарегистрироватьИзменения»

```
&НаКлиенте
Процедура ЗарегистрироватьИзменения(Команда)

    ЗарегистрироватьИзмененияНаСервере(Объект.Ссылка);

КонецПроцедуры
```

Процедуру `ЗарегистрироватьИзмененияНаСервере()` заполним следующим образом (листинг 3.52).

Листинг 3.52. Процедура «ЗарегистрироватьИзмененияНаСервере»

```
&НаСервереБезКонтекста
Процедура ЗарегистрироватьИзмененияНаСервере(Узел)

    ПланыОбмена.ЗарегистрироватьИзменения(Узел);

КонецПроцедуры
```

В этой процедуре мы обращаемся к механизму регистрации изменений, вызывая метод менеджера планов обмена – `ЗарегистрироватьИзменения()`. В этот метод передается ссылка на текущий узел плана обмена. В результате выполнения этой процедуры в информационной базе будут созданы записи регистрации изменений, предназначенные для пересылки в указанный узел, для всех объектов обмена, входящих в состав нашего плана обмена.

Стандартные процедуры обмена

Для выполнения обмена данными добавим в конфигурацию обработку ОбменСУдаленнымиСкладами и откроем ее основную форму.

Добавим в форму реквизит УзелОбмена типа `ПланОбменаСсылка.УдаленныеСклады`, в котором будет содержаться ссылка на выбранный узел обмена. А также добавим в форму команды ВыгрузитьДанные и ЗагрузитьДанные. Перетащим эти команды и реквизит в окно элементов формы.

У всех кнопок, соответствующих командам, снимем флажок у свойства Доступность. Это делается для того, чтобы действия по выполнению обмена становились возможны только в случае выбора в поле Узел обмена непредопределенного узла, который не совпадает с узлом текущей информационной базы.

Для того чтобы обеспечить такое поведение кнопок, поместим в модуле формы функцию `ПолучитьДоступностьПоУзлу()`, в которую передается ссылка на выбранный узел (листинг 3.53).

Листинг 3.53. Функция «ПолучитьДоступностьПоУзлу»

```
&НаСервереБезКонтекста
Функция ПолучитьДоступностьПоУзлу(Узел)

    ДоступностьКнопок = Ложь;

    // Определить наличие установленного непредопределенного узла.
    Если Узел <> ПланыОбмена.УдаленныеСклады.ПустаяСсылка()
      И Узел <> ПланыОбмена.УдаленныеСклады.ЭтотУзел()
      И Узел.ПолучитьОбъект() <> Неопределено Тогда
          ДоступностьКнопок = Истина;

    КонецЕсли;

    Возврат ДоступностьКнопок;

КонецФункции
```

Эта функция будет возвращать `Истина`, в случае если ссылка, содержащаяся в параметре `Узел`, не пустая и не равна ссылке на предопределенный узел, соответствующий текущей информационной базе. Для получения ссылки на текущий узел используется метод менеджера узла плана обмена `ЭтотУзел()`. В противном случае функция вернет `Ложь`.

После этого создадим обработчик события `ПриИзменении` для поля формы УзелОбмена, в котором доступность кнопок будет зависеть от значения, возвращенного описанной ранее функцией (листинг 3.54).

Листинг 3.54. Обработчик события «ПриИзменении» поля «УзелОбмена»

```
&НаКлиенте
Процедура УзелОбменаПриИзменении(Элемент)

    ДоступностьКнопок = ПолучитьДоступностьПоУзлу(УзелОбмена);

    // Установить доступность кнопок в зависимости от установленного узла обмена.
    Элементы.ЗагрузитьДанные.Доступность = ДоступностьКнопок;
    Элементы.ВыгрузитьДанные.Доступность = ДоступностьКнопок;

КонецПроцедуры
```

Теперь создадим обработчик команды ВыгрузитьДанные и заполним следующим образом (листинг 3.55).

Листинг 3.55. Обработчик команды «ВыгрузитьДанные»

```
&НаКлиенте
Процедура ВыполнитьВыгрузку(Команда)

    ВыполнитьВыгрузкуНаСервере(УзелОбмена);

КонецПроцедуры
```

В этом обработчике мы вызываем процедуру `ВыполнитьВыгрузкуНаСервере()`, в которую передаем ссылку на узел обмена, выбранный в поле обработки. В реквизите УзелОбмена будет содержаться ссылка на узел, для которого будет производиться запись изменений (листинг 3.56).

Листинг 3.56. Процедура «ВыполнитьВыгрузкуНаСервере»

```
&НаСервереБезКонтекста
Процедура ВыполнитьВыгрузкуНаСервере(Узел)

    // Получить объект узла обмена.
    УзелОбмена = Узел.ПолучитьОбъект();

    // Записать новое сообщение обмена.
    УзелОбмена.ЗаписатьСообщениеСИзменениями();

КонецПроцедуры
```

В этой процедуре мы получаем объект от ссылки на узел плана обмена и выполняем процедуру `ЗаписатьСообщенияСИзменениями()`, которую мы поместим в модуле плана обмена УдаленныеСклады (см. листинг 3.59).

Теперь создадим обработчик команды ЗагрузитьДанные и заполним следующим образом (листинг 3.57).

Листинг 3.57. Обработчик команды «ЗагрузитьДанные»

```
&НаКлиенте
Процедура ВыполнитьЗагрузку(Команда)

    ВыполнитьЗагрузкуНаСервере(УзелОбмена);

КонецПроцедуры
```

В этом обработчике мы вызываем процедуру `ВыполнитьЗагрузкуНаСервере()`, в которую передаем ссылку на узел обмена, выбранный в поле обработки. В реквизите УзелОбмена будет содержаться ссылка на узел, от которого будет производиться чтение изменений (листинг 3.58).

Листинг 3.58. Процедура «ВыполнитьЗагрузкуНаСервере»

```
&НаСервереБезКонтекста
Процедура ВыполнитьЗагрузкуНаСервере(Узел)

    // Получить объект узла обмена.
    УзелОбмена = Узел.ПолучитьОбъект();

    // Прочитать новое сообщение обмена.
    УзелОбмена.ПрочитатьСообщениеСИзменениями();

КонецПроцедуры
```

В этой процедуре мы получаем объект от ссылки на узел плана обмена и выполняем процедуру `ПрочитатьСообщенияСИзменениями()`, которую мы поместим в модуле плана обмена УдаленныеСклады (см. листинг 3.61).

Процедуру `ЗаписатьСообщенияСИзменениями()` в модуле плана обмена заполним следующим образом (листинг 3.59).

Листинг 3.59. Процедура для записи данных обмена

```
Процедура ЗаписатьСообщениеСИзменениями() Экспорт

    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = "-------- Выгрузка в узел " + Строка(ЭтотОбъект) + " ----------";
    Сообщение.Сообщить();

    // Получить имя файла обмена.
    ИмяФайла = ОбменСУдаленнымиСкладами.ПолучитьИмяФайлаОбмена(
                                ПланыОбмена.УдаленныеСклады.ЭтотУзел(), Ссылка);

    // Создать объект записи XML.
    // *** ЗаписьXML-документов.
    ЗаписьXML = Новый ЗаписьXML;
    ЗаписьXML.ОткрытьФайл(ИмяФайла);
    ЗаписьXML.ЗаписатьОбъявлениеXML();
```

```
    // *** Инфраструктура сообщений.
    ЗаписьСообщения = ПланыОбмена.СоздатьЗаписьСообщения();
    ЗаписьСообщения.НачатьЗапись(ЗаписьXML, Ссылка);
    // Записать соответствие пространств имен для сокращения размера файла сообщения
    ЗаписьXML.ЗаписатьСоответствиеПространстваИмен("xsd", "http://www.w3.org/2001/XMLSchema");
    ЗаписьXML.ЗаписатьСоответствиеПространстваИмен("xsi", "http://www.w3.org/2001/XMLSchema-instance");
    ЗаписьXML.ЗаписатьСоответствиеПространстваИмен("v8", "http://v8.1c.ru/data");

    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = " Номер сообщения: " + ЗаписьСообщения.НомерСообщения;
    Сообщение.Сообщить();

    // Получить выборку измененных данных.
    // *** Механизм регистрации изменений.
    ВыборкаИзменений = ПланыОбмена.ВыбратьИзменения(
                        ЗаписьСообщения.Получатель, ЗаписьСообщения.НомерСообщения);
    Пока ВыборкаИзменений.Следующий() Цикл
        // Записать данные в сообщение *** XML-сериализация.
        ЗаписатьXML(ЗаписьXML, ВыборкаИзменений.Получить());
    КонецЦикла;

    ЗаписьСообщения.ЗакончитьЗапись();
    ЗаписьXML.Закрыть();

    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = "-------- Конец выгрузки ----------";
    Сообщение.Сообщить();

КонецПроцедуры
```

В этой процедуре сначала мы сообщаем пользователю о начале выгрузки данных в узел обмена.

Затем с помощью функции `ПолучитьИмяФайлаОбмена()`, расположенной в общем модуле ОбменСУдаленнымиСкладами, мы формируем имя файла, в который будет записано сообщение обмена.

Для упрощения примера мы будем обмениваться сообщениями через заранее известный каталог обмена, путь к которому хранится в константе Каталог обмена. Значение этой константы (например, `e:\Exchange`), наряду с префиксом номеров, можно задать в форме настроек информационной базы (см. рис. 3.15).

Имена файлов сообщений стандартизованы и имеют вид: Message_<КодУзлаОтправителя>_<КодУзлаПолучателя>.xml. В качестве узла-отправителя сообщения обмена в функцию `ПолучитьИмяФайлаОбмена()` передается ссылка на предопределенный узел информационной базы, полученный с помощью метода `ЭтотУзел()`, а в качестве узла-получателя – ссылка на тот узел обмена, в модуле которого мы находимся (листинг 3.60).

Листинг 3.60. Функция «ПолучитьИмяФайлаОбмена()»

```
Функция ПолучитьИмяФайлаОбмена(УзелИсточник, УзелПриемник) Экспорт

    // Сформировать полное имя файла обмена.
    Каталог = Константы.КаталогОбмена.Получить();
    ИмяФайла = Каталог + ?(Прав(Каталог, 1) = "\","", "\") + "Message_" +
        СокрЛП(УзелИсточник.Код) + "_" + СокрЛП(УзелПриемник.Код) + ".xml";
    Возврат ИмяФайла;

КонецФункции
```

После этого мы обращаемся к механизмам записи XML-документов и создаем новый объект – `ЗаписьXML`. С помощью него открываем для записи XML-файл с полученным ранее именем и записываем в него объявление XML.

Затем мы обращаемся к механизмам инфраструктуры сообщений и создаем новый объект `ЗаписьСообщенияОбмена`, метод которого `НачатьЗапись()` позволяет, кроме всего прочего, создать очередной номер сообщения и записать заголовок сообщения в XML.

В качестве ссылки на узел обмена, который является получателем сообщения, мы используем значение стандартного реквизита `Ссылка` плана обмена УдаленныеСклады, в модуле которого мы находимся.

Для сокращения размера файла сообщения первоначально определяем соответствие используемым пространствам имен.

После этого, чтобы получить данные, которые необходимо сохранить в этом файле, мы обращаемся к механизму регистрации изменений и получаем выборку из записей регистрации изменений, предназначенных узлу-получателю. При формировании выборки методом менеджера планов обмена `ВыбратьИзменения()` мы передаем вторым параметром номер сообщения.

В цикле перебора измененных данных мы сериализуем эти данные в открытый XML-файл методом глобального контекста `ЗаписатьXML`.

После выхода из цикла мы заканчиваем запись методом `ЗакончитьЗапись()` объекта `ЗаписьСообщенияОбмена`. Затем закрываем файл с сообщением обмена методом `Закрыть()` объекта `ЗаписьXML` и выводим сообщение об окончании выгрузки данных.

Процедуру `ПрочитатьСообщенияСИзменениями()` в модуле плана обмена заполним следующим образом (листинг 3.61).

Листинг 3.61. Процедура для чтения данных обмена

```
Процедура ПрочитатьСообщениеСИзменениями() Экспорт

	// Получить имя файла обмена.
	ИмяФайла = ОбменСУдаленнымиСкладами.ПолучитьИмяФайлаОбмена(
	                                Ссылка, ПланыОбмена.УдаленныеСклады.ЭтотУзел());

	Файл = Новый Файл(ИмяФайла);
	Если Не Файл.Существует() Тогда
		Возврат;
	КонецЕсли;

	// Попытаться открыть файл.
	// *** Чтение документов XML.
	ЧтениеXML = Новый ЧтениеXML;
	Попытка
		ЧтениеXML.ОткрытьФайл(ИмяФайла);

	Исключение
		Сообщение = Новый СообщениеПользователю;
		Сообщение.Текст = "Невозможно открыть файл обмена данными.";
		Сообщение.Сообщить();

		Возврат;

	КонецПопытки;

	Сообщение = Новый СообщениеПользователю;
	Сообщение.Текст = "-------- Загрузка из " + Строка(ЭтотОбъект) + " ----------";
	Сообщение.Сообщить();
	Сообщение.Текст = " – Считывается файл " + ИмяФайла;
	Сообщение.Сообщить();

	// Загрузить данные из найденного файла.
	// *** Инфраструктура сообщений.
	ЧтениеСообщения = ПланыОбмена.СоздатьЧтениеСообщения();

	// Прочитать заголовок сообщения обмена данными – файла XML.
	ЧтениеСообщения.НачатьЧтение(ЧтениеXML);

	// Проверить, что сообщение предназначено для этого узла.
	Если ЧтениеСообщения.Отправитель <> Ссылка Тогда
		ВызватьИсключение "Неверный узел";
	КонецЕсли;

	// Удалить регистрацию изменений для узла-отправителя сообщения.
	// *** Служба регистрации изменений.
	ПланыОбмена.УдалитьРегистрациюИзменений(
	                                ЧтениеСообщения.Отправитель, ЧтениеСообщения.НомерПринятого);

	// Прочитать данные из сообщения в цикле. *** XML-сериализация.
	Пока ВозможностьЧтенияXML(ЧтениеXML) Цикл

		// Прочитать очередное значение.
```

```
            Данные = ПрочитатьXML(ЧтениеXML);

            // Записать полученные данные.
            Данные.ОбменДанными.Отправитель = ЧтениеСообщения.Отправитель;
            Данные.ОбменДанными.Загрузка = Истина;
            Данные.Записать();

        КонецЦикла;

        ЧтениеСообщения.ЗакончитьЧтение();
        ЧтениеXML.Закрыть();
        УдалитьФайлы(ИмяФайла);

        Сообщение = Новый СообщениеПользователю;
        Сообщение.Текст = "-------- Конец загрузки ----------";
        Сообщение.Сообщить();

КонецПроцедуры
```

В этой процедуре с помощью функции `ПолучитьИмяФайлаОбмена()`, расположенной в общем модуле ОбменСУдаленнымиСкладами (см. листинг 3.60), мы формируем имя файла, из которого будет прочитано сообщение обмена.

Имена файлов сообщений стандартизованы и имеют вид: Message_<КодУзлаОтправителя>_<КодУзлаПолучателя>.xml. В качестве узла-отправителя сообщения обмена в функцию `ПолучитьИмяФайлаОбмена()` передается ссылка на тот узел обмена, в модуле которого мы находимся, а в качестве узла-получателя – ссылка на предопределенный узел информационной базы.

Затем на основе имени файла мы создаем объект `Файл` и проверяем, существует ли он. Если файл обмена найден, то мы пытаемся открыть его для чтения с помощью объекта `ЧтениеXML`. В случае успеха выводим сообщение о начале загрузки данных из файла.

Затем мы обращаемся к механизмам инфраструктуры сообщений планов обмена и создаем объект `ЧтениеСообщенияОбмена`. Используя метод этого объекта `НачатьЧтение()`, мы считываем заголовок XML-сообщения, в котором содержится, в том числе, информация об отправителе сообщения.

После этого проверяем, является ли отправитель сообщения тем узлом плана обмена, для которого мы в данном вызове этой процедуры производим обмен данными. В качестве ссылки на узел обмена, который является отправителем сообщения, мы используем значение стандартного реквизита `Ссылка` плана обмена УдаленныеСклады, в модуле которого мы находимся.

Если все в порядке, то, перед тем как начать чтение данных, мы удаляем все записи регистрации изменений, которые были сделаны для этого узла и соответствовали номерам сообщений, меньшим или равным указанному

в обрабатываемом нами сообщении как номер принятого. Это делается затем, чтобы исключить дублирование данных, которые уже были ранее посланы этому узлу и им обработаны. Для этого мы обращаемся к службе регистрации изменений менеджера планов обмена и используем метод `УдалитьРегистрациюИзменений()`.

Затем в цикле мы выполняем чтение данных, содержащихся в сообщении обмена. При этом для каждого элемента в функции `ВозможностьЧтенияXML()` получается очередной тип данных XML и определяется, имеется ли соответствующий тип «1С:Предприятия».

Для чтения данных мы используем метод глобального контекста `ПрочитатьXML()`. В результате переменная `Данные` будет содержать объект «1С:Предприятия», полученный из сообщения обмена.

После этого мы записываем полученные данные методом `Записать()`. Перед записью полученного объекта мы устанавливаем у него в параметрах обмена данными узел отправителя, чтобы система при записи этого объекта в нашей базе данных не формировала записи регистрации изменений этого объекта для того узла, от которого мы его только что получили.

Кроме того, в параметрах обмена данными мы устанавливаем свойство `Загрузка`, информирующее систему о том, что запись объекта будет происходить в режиме обновления данных, полученных в результате обмена. Такое указание позволяет системе упростить процедуру записи объекта, отказавшись от ряда стандартных проверок и исключив изменения связанных данных, которые выполняются при обычной записи.

После того как все сообщение обмена будет нами обработано, мы заканчиваем чтение методом `ЗакончитьЧтение()` объекта `ЧтениеСообщенияОбмена`. Затем закрываем файл с сообщением обмена методом `Закрыть()` объекта `ЧтениеXML` и удаляем этот файл из каталога обмена. В заключение выводим сообщение об окончании загрузки данных из файла.

Стратегия распространения данных

В предыдущем разделе мы рассмотрели стандартные процедуры выгрузки/загрузки данных в узлы обмена. Но на самом деле обычно их требуется дополнить так, чтобы они решали еще ряд задач, возникающих при обмене данными. Например, при передаче данных обмена в другие узлы требуется реализовать стратегию распространения данных, при которой конкретным узлам должны отправляться только данные, соответствующие определенному условию.

В нашем примере мы реализуем стратегию распространения данных, согласно которой документы ПриходнаяНакладная и РасходнаяНакладная (и соответствующие им наборы записей регистра накопления УчетНоменклатуры) выгружаются только в том случае, если значение реквизита Склад документов (и наборов записей регистра накопления) совпадает по значению с одноименным реквизитом, определенным для узла-получателя.

Для этого дополним процедуру `ЗаписатьСообщениеСИзменениями()` в модуле плана обмена (см. листинг 3.59) следующим образом (листинг 3.62).

Листинг 3.62. Процедура «ЗаписатьСообщениеСИзменениями»

```
Процедура ЗаписатьСообщениеСИзменениями() Экспорт

    ...
    // Получить выборку измененных данных.
    // *** Механизм регистрации изменений.
    ВыборкаИзменений = ПланыОбмена.ВыбратьИзменения(
                        ЗаписьСообщения.Получатель, ЗаписьСообщения.НомерСообщения);
    Пока ВыборкаИзменений.Следующий() Цикл
        Данные = ВыборкаИзменений.Получить();

        // Проверить, нужен ли перенос данных. Если нет, то записать удаление этих данных.
        Если Не ОбменСУдаленнымиСкладами.НуженПереносДанных(
                    ЗаписьСообщения.Получатель, Данные) Тогда
            ОбменСУдаленнымиСкладами.УдалениеДанных(Данные);
        КонецЕсли;

        // Записать данные в сообщение *** XML-сериализация.
        ЗаписатьXML(ЗаписьXML, Данные);
    КонецЦикла;
    ...

КонецПроцедуры
```

Добавленный фрагмент выделен жирным шрифтом.

В цикле перебора измененных данных, перед тем как записать данные в XML-файл, мы проверяем, нужно ли выгружать эти данные в узел-получатель. Проверка выполняется в функции `НуженПереносДанных()`, расположенной в общем модуле ОбменСУдаленнымиСкладами (см. листинг 3.63). В случае если выгрузка не должна производиться (функция возвращает `Ложь`), вызывается процедура `УдалениеДанных()`, расположенная в общем модуле ОбменСУдаленнымиСкладами (см. листинг 3.64).

Листинг 3.63. Функция для реализации стратегии переноса данных

```
Функция НуженПереносДанных(Получатель, Данные) Экспорт

	Перенос = Истина;
	Склад = Получатель.Склад;

	Если ТипЗнч(Данные) = Тип("ДокументОбъект.ПриходнаяНакладная")
		ИЛИ   ТипЗнч(Данные) = Тип("ДокументОбъект.РасходнаяНакладная") Тогда

		// Проверить, совпадает ли склад документов со складом узла обмена.
		Если Не Склад.Пустая() И Данные.Склад <> Склад Тогда
			Перенос = Ложь;
		КонецЕсли;

	ИначеЕсли ТипЗнч(Данные) = Тип("РегистрНакопленияНаборЗаписей.УчетНоменклатуры") Тогда

		// Проверить, что у всех записей набора склад один и тот же.
		СкладНабора = Неопределено;
		Для Каждого Запись Из Данные Цикл
			Если СкладНабора <> Неопределено И Запись.Склад <> СкладНабора Тогда
				ВызватьИсключение "Склад должен быть одинаковый для всех записей";
			КонецЕсли;
			СкладНабора = Запись.Склад;
		КонецЦикла;

		// Проверить, совпадает ли склад набора записей со складом узла обмена.
		Если Не Склад.Пустая() И СкладНабора <> Склад Тогда
			Перенос = Ложь;
		КонецЕсли;
	КонецЕсли;

	Возврат Перенос;

КонецФункции
```

Функция `НуженПереносДанных()` реализует стратегию распространения данных, описанную выше. В параметре `Данные` в функцию передается полученный объект обмена, а в параметре `Получатель` – ссылка на узел-получатель сообщения.

Если тип объекта `ДокументОбъект.ПриходнаяНакладная` или `ДокументОбъект.РасходнаяНакладная` или `РегистрНакопленияНаборЗаписей.УчетНоменклатуры`, то проверяется следующее условие переноса данных. Если реквизит Склад для узла-получателя заполнен и не равен значению одноименного реквизита документов ПриходнаяНакладная или РасходнаяНакладная или регистра накопления УчетНоменклатуры, то функция возвращает `Ложь`. Во всех остальных случаях возвращается `Истина`.

Для набора записей регистра накопления производится проверка на то, что во всех записях используется один склад. Данная проверка может и не проводиться, если такая ситуация изначально не может встретиться.

В случае если данные (полученные при выборке изменений) не подлежат выгрузке в узел-получатель, производится вызов процедуры `УдалениеДанных()`. В этой процедуре для данных, имеющих объектную природу, создается объект `УдалениеОбъекта`. Этим достигается удаление в узле-приемнике «ранее неправильно отосланных данных». Наборы записей регистров очищаются (листинг 3.64).

Листинг 3.64. Функция для удаления данных при обмене

```
Процедура УдалениеДанных(Данные) Экспорт

    // Получить объект описания метаданного, соответствующий данным.
    ОбъектМетаданных = ?(ТипЗнч(Данные) = Тип("УдалениеОбъекта")
                        , Данные.Ссылка.Метаданные(), Данные.Метаданные());

    Если Метаданные.Справочники.Содержит(ОбъектМетаданных)
        ИЛИ Метаданные.Документы.Содержит(ОбъектМетаданных)
        ИЛИ Метаданные.ПланыСчетов.Содержит(ОбъектМетаданных)
        ИЛИ Метаданные.ПланыВидовРасчета.Содержит(ОбъектМетаданных)
        ИЛИ Метаданные.ПланыВидовХарактеристик.Содержит(ОбъектМетаданных) Тогда

        // Записать удаление объекта для объектных данных.
        Данные = Новый УдалениеОбъекта(Данные.Ссылка);

    ИначеЕсли Метаданные.РегистрыСведений.Содержит(ОбъектМетаданных)
        ИЛИ Метаданные.РегистрыНакопления.Содержит(ОбъектМетаданных)
        ИЛИ Метаданные.РегистрыБухгалтерии.Содержит(ОбъектМетаданных)
        ИЛИ Метаданные.РегистрыРасчета.Содержит(ОбъектМетаданных)
        ИЛИ Метаданные.РегистрыРасчета.Содержит(ОбъектМетаданных.Родитель)
        ИЛИ Метаданные.Последовательности.Содержит(ОбъектМетаданных) Тогда

        // Очистить данные.
        Данные.Очистить();

    КонецЕсли;

КонецПроцедуры
```

Обмен данными с разной структурой

Поскольку рассматриваемый нами механизм выгрузки относится к разряду универсальных, подразумевается, что обмен данными производится между конфигурациями, имеющими разную структуру.

Рассмотрим простой пример обмена данными с различной структурой. Например, в базе узла-получателя в иерархическом справочнике Номенклатура присутствует дополнительный реквизит по сравнению с базой узла-отправителя, и наоборот.

В этом случае запись всех «общих» для обоих узлов реквизитов справочника должна производиться в ручном режиме. Порядок следования реквизитов справочника в конфигурации может не совпадать с порядком их записи в XML, но последующее чтение реквизитов из XML должно производиться в том же порядке, в каком они были записаны. Все остальные данные с идентичной структурой сериализуются в/из XML стандартным образом.

Для этого дополним процедуру `ЗаписатьСообщениеСИзменениями()` в модуле плана обмена (см. листинг 3.59) следующим образом (листинг 3.65).

Листинг 3.65. Процедура «ЗаписатьСообщениеСИзменениями»

```
Процедура ЗаписатьСообщениеСИзменениями() Экспорт

    ...
    // Получить выборку измененных данных.
    // *** Механизм регистрации изменений.
    ВыборкаИзменений = ПланыОбмена.ВыбратьИзменения(
                        ЗаписьСообщения.Получатель, ЗаписьСообщения.НомерСообщения);
    Пока ВыборкаИзменений.Следующий() Цикл
        Данные = ВыборкаИзменений.Получить();

        // Проверить, нужен ли перенос данных. Если нет, то записать удаление этих данных.
        Если Не ОбменСУдаленнымиСкладами.НуженПереносДанных(
                        ЗаписьСообщения.Получатель, Данные) Тогда
            ОбменСУдаленнымиСкладами.УдалениеДанных(Данные);
        КонецЕсли;

        // Записать данные в сообщение *** XML-сериализация.
        ОбменСУдаленнымиСкладами.ЗаписатьДанные(ЗаписьXML, Данные);
    КонецЦикла;
    ...

КонецПроцедуры
```

Добавленный фрагмент выделен жирным шрифтом. В этом фрагменте данные, полученные в цикле перебора записей регистрации изменений, сериализуются в открытый XML-файл с помощью процедуры `ЗаписатьДанные()`, расположенной в общем модуле ОбменСУдаленнымиСкладами. В параметре `Данные` в процедуру передается полученный объект обмена (листинг 3.66).

Листинг 3.66. Процедура «ЗаписатьДанные»

```
Процедура ЗаписатьДанные(ЗаписьXML, Данные) Экспорт

    Удаление = ?(ТипЗнч(Данные) = Тип("УдалениеОбъекта"), Истина, Ложь);

    // Получить объект описания метаданного, соответствующий данным.
    ОбъектМетаданных = ?(Удаление, Данные.Ссылка.Метаданные(), Данные.Метаданные());
```

```
    Если Не Удаление И ОбъектМетаданных = Метаданные.Справочники.Номенклатура Тогда
        // Записать элемент справочника вручную.
        ЗаписатьXMLНоменклатура(ЗаписьXML, Данные);
    Иначе

        // Записать данные с помощью стандартного метода.
        ЗаписатьXML(ЗаписьXML, Данные);
    КонецЕсли

КонецПроцедуры
```

В процедуре производится проверка: если записывается не удаление данных и записываемый объект является элементом справочника Номенклатура, то вызывается процедура `ЗаписатьXMLНоменклатура()`, расположенная в общем модуле ОбменСУдаленнымиСкладами, для записи элемента справочника вручную (листинг 3.67). В противном случае запись производится с помощью стандартного метода `ЗаписатьXML()`.

Листинг 3.67. Процедура «ЗаписатьXMLНоменклатура»

```
Процедура ЗаписатьXMLНоменклатура(ЗаписьXML, Товар )Экспорт

    // Записать начало элемента XML.
    ЗаписьXML.ЗаписатьНачалоЭлемента("CatalogObject.Номенклатура.Вручную");

    // Ссылка
    ЗаписатьXML(ЗаписьXML, Товар.Ссылка, "Ref", НазначениеТипаXML.Явное);
    // ЭтоГруппа
    ЗаписатьXML(ЗаписьXML, Товар.ЭтоГруппа, "IsFolder", НазначениеТипаXML.Явное);
    // Родитель
    ЗаписатьXML(ЗаписьXML, Товар.Родитель, "Parent", НазначениеТипаXML.Явное);
    // Наименование
    ЗаписатьXML(ЗаписьXML, Товар.Наименование, "Description", НазначениеТипаXML.Явное);
    // Код
    ЗаписатьXML(ЗаписьXML, Товар.Код, "Code", НазначениеТипаXML.Явное);

    // Записать реквизиты, выгружаемые только для элемента справочника.
    Если Не Товар.ЭтоГруппа Тогда
        // ЗакупочнаяЦена
        ЗаписатьXML(ЗаписьXML, Товар.ЗакупочнаяЦена, "ЗакупочнаяЦена", НазначениеТипаXML.Явное);
        // ФайлКартинки
        ЗаписатьXML(ЗаписьXML, Товар.ФайлКартинки, "ФайлКартинки", НазначениеТипаXML.Явное);
        // Картинка
        ЗаписатьXML(ЗаписьXML, Товар.Картинка, "Картинка", НазначениеТипаXML.Явное);
    КонецЕсли;

    // Записать конец элемента.
    ЗаписьXML.ЗаписатьКонецЭлемента();

КонецПроцедуры
```

В процедуре ЗаписатьXMLНоменклатура() производится «ручное» формирование элемента CatalogObject.Номенклатура.Вручную.

Сначала в открытый XML-файл мы записываем начало элемента с заголовком, указывающим на ручное формирование элемента. Затем при помощи метода ЗаписатьXML() записываем значения «общих» реквизитов, которые хотим передать с помощью обмена. Третьим параметром передаем строковое имя записываемого реквизита и в качестве четвертого параметра указываем значение перечисления НазначениеТипаXML, определяющее необходимость явного указания типа данных XML. После этого мы завершаем запись элемента и записываем в XML-файл конец элемента.

В процедуру ПрочитатьСообщенияСИзменениями() в модуле плана обмена (см. листинг 3.61) внесем следующие изменения (листинг 3.68).

Листинг 3.68. Процедура «ПрочитатьСообщениеСИзменениями»

```
Процедура ПрочитатьСообщениеСИзменениями() Экспорт

    ...
    // Прочитать данные из сообщения в цикле. *** XML-сериализация.
    Пока ОбменСУдаленнымиСкладами.ВозможностьЧтенияДанных(ЧтениеXML) Цикл

        // Прочитать очередное значение.
        Данные = ОбменСУдаленнымиСкладами.ПрочитатьДанные(ЧтениеXML);

        // Записать полученные данные.
        Данные.ОбменДанными.Отправитель = ЧтениеСообщения.Отправитель;
        Данные.ОбменДанными.Загрузка = Истина;
        Данные.Записать();

    КонецЦикла;
    ...

КонецПроцедуры
```

Добавленный фрагмент выделен жирным шрифтом. В этом фрагменте в цикле мы выполняем чтение данных, содержащихся в сообщении обмена. При этом для каждого элемента в функции ВозможностьЧтенияДанных(), расположенной в общем модуле ОбменСУдаленнымиСкладами (см. листинг 3.69), проверяется возможность чтения данных. Если имя типа – CatalogObject.Товары.Вручную, возвращается значение Истина. В противном случае вызывается стандартный метод ВозможностьЧтенияXML(), с помощью которого из объекта ЧтениеXML получается очередной тип данных XML и определяется, имеется ли соответствующий тип «1С:Предприятия».

Листинг 3.69. Функция «ВозможностьЧтенияДанных»

```
Функция ВозможностьЧтенияДанных(ЧтениеXML)

    // Получить тип данных XML, который может быть считан в данный момент.
    ТипXML = ПолучитьXMLТип(ЧтениеXML);
    Если ТипXML = Неопределено Тогда
        Возврат Ложь;
    КонецЕсли;

    Если ТипXML.ИмяТипа = "CatalogObject.Номенклатура.Вручную" И ТипXML.URIПространстваИмен = ""
                                Тогда
        Возврат Истина;
    КонецЕсли;

    Возврат ВозможностьЧтенияXML(ЧтениеXML);

КонецФункции
```

В цикле чтения данных в процедуре `ПрочитатьСообщениеСИзменениями()` вызывается функция `ПрочитатьДанные()`, расположенная в общем модуле ОбменСУдаленнымиСкладами (см. листинг 3.70), в которой производится чтение данных из элемента. При этом если читается элемент `CatalogObject.Номенклатура.Вручную`, то вызывается функция `ПрочитатьXMLНоменклатура()`, расположенная в общем модуле ОбменСУдаленнымиСкладами (см. листинг 3.71) для «ручного» чтения данных, в противном случае чтение осуществляется с помощью стандартного метода `ПрочитатьXML`.

Листинг 3.70. Функция «ПрочитатьДанные»

```
Функция ПрочитатьДанные(ЧтениеXML)

    ТипXML = ПолучитьXMLТип(ЧтениеXML);
    Если ТипXML = Неопределено Тогда
        Возврат Неопределено;
    КонецЕсли;

    Если ТипXML.ИмяТипа = "CatalogObject.Номенклатура.Вручную" И ТипXML.URIПространстваИмен = ""
                        Тогда
        // Прочитать значение справочника Номенклатура.
        Возврат ПрочитатьXMLНоменклатура(ЧтениеXML);
    КонецЕсли;

    // Прочитать значение из объекта ЧтениеXML стандартным образом.
    Возврат ПрочитатьXML(ЧтениеXML);

КонецФункции
```

Листинг 3.71. Функция «ПрочитатьXMLНоменклатура»

```
Функция ПрочитатьXMLНоменклатура(ЧтениеXML)Экспорт

	Если ЧтениеXML.ТипУзла <> ТипУзлаXML.НачалоЭлемента Тогда
		ВызватьИсключение "Ошибка чтения XML";
	КонецЕсли;

	// Прочитать следующий узел XML-документа.
	ЧтениеXML.Прочитать();

	// Прочитать ссылку на элемент справочника Номенклатура.
	ТоварСсылка = ПрочитатьXML(ЧтениеXML);
	Если ТипЗнч(ТоварСсылка) <> Тип("СправочникСсылка.Номенклатура") Тогда
		ВызватьИсключение "Ошибка чтения XML";
	КонецЕсли;

	// Создать объект по полученной ссылке.
	Товар = ТоварСсылка.ПолучитьОбъект();

	// Прочитать признак группы.
	ЭтоГруппа = ПрочитатьXML(ЧтениеXML);
	Если Товар <> Неопределено Тогда
		Если Товар.ЭтоГруппа <> ЭтоГруппа Тогда
			ВызватьИсключение "Некорректные данные";
		КонецЕсли;

	Иначе
		// Создать элемент справочника Номенклатура.
		Если ЭтоГруппа = Истина Тогда
			Товар = Справочники.Номенклатура.СоздатьГруппу();
		Иначе
			Товар = Справочники.Номенклатура.СоздатьЭлемент();
		КонецЕсли;

		// Устанавить значение ссылки для нового объекта.
		Товар.УстановитьСсылкуНового(ТоварСсылка);
	КонецЕсли;

	// Родитель
	Товар.Родитель = ПрочитатьXML(ЧтениеXML);
	// Наименование
	Товар.Наименование = ПрочитатьXML(ЧтениеXML);
	// Код
	Товар.Код = ПрочитатьXML(ЧтениеXML);

	// Прочитать реквизиты, загружаемые только для элемента справочника.
	Если Не Товар.ЭтоГруппа Тогда
		// ЗакупочнаяЦена
		Товар.ЗакупочнаяЦена = ПрочитатьXML(ЧтениеXML);
		// ФайлКартинки
		Товар.ФайлКартинки = ПрочитатьXML(ЧтениеXML);
		// Картинка
		Товар.Картинка = ПрочитатьXML(ЧтениеXML);
	КонецЕсли;
```

```
    // Проверить, что текущим узлом является КонецЭлемента.
    Если ЧтениеXML.ТипУзла <> ТипУзлаXML.КонецЭлемента Тогда
        ВызватьИсключение "Ошибка чтения XML";
    КонецЕсли;

    // Прочитать следующий узел для завершение чтения элемента XML-документа.
    ЧтениеXML.Прочитать();
    Возврат Товар;

КонецФункции
```

В функции `ПрочитатьXMLНоменклатура()` производится последовательное считывание каждого узла элемента справочника Номенклатура, записанного вручную в сообщение обмена.

Сначала мы проверяем, что находимся в XML-файле на начале элемента, затем двигаемся на следующий узел методом `Прочитать()` объекта `ЧтениеXML`. Затем считываем ссылку на элемент справочника и пытаемся создать объект по полученной ссылке.

Если это сделать не удалось, значит, такого объекта еще не существует, и мы создаем или новую группу, или новый элемент справочника – в зависимости от значения реквизита ЭтоГруппа, считанного из сообщения обмена. Затем последовательно считываем (в том порядке, в котором записывали) и устанавливаем значения «общих» реквизитов, которые мы ранее вручную выгрузили в процедуре `ЗаписатьXMLНоменклатура()`, см. листинг 3.67.

В завершение проверяем: если считана вся информация о номенклатуре (достигнут конец элемента), то двигаемся на следующий элемент данных в сообщении обмена.

В результате функция возвращает данные о прочитанном элементе номенклатуры в процедуру `ПрочитатьСообщениеСИзменениями()`, в переменную `Данные` (см. листинг 3.68).

В случае стандартного чтения данных методом `ПрочитатьXML` переменная `Данные` будет содержать любой другой объект «1С:Предприятия», полученный из сообщения обмена.

Стратегия разрешения коллизий

Надо понимать, что при вводе новых объектов в различных узлах обмена ссылки на объекты базы данных будут разными и все новые объекты будут «доставлены по назначению». Но при обмене уже существующими объектами базы данных (с одинаковыми ссылками), которые был изменены сразу в нескольких узлах обмена после последнего сеанса обмена, может возник-

нуть коллизия, когда требуется определить, какие изменения принять, а какие – отклонить.

Чтобы разрешить эту коллизию, в нашем примере мы будем использовать наиболее простую стратегию «главный – подчиненный». Согласно этой стратегии изменения, полученные от узла обмена, отмеченного в списке узлов как главный, будут иметь приоритет над остальными.

То есть если для одного и того же объекта обмена данными изменения будут зарегистрированы и в главном, и в любом другом узле обмена, то изменения главного узла будут приняты, а остальные – отклонены.

В процедуру `ПрочитатьСообщенияСИзменениями()` в модуле плана обмена (см. листинг 3.61) внесем следующие изменения (листинг 3.72).

Листинг 3.72. Процедура «ПрочитатьСообщениеСИзменениями»

```
Процедура ПрочитатьСообщениеСИзменениями() Экспорт

    ...
    // Прочитать данные из сообщения в цикле. *** XML-сериализация.
    Пока ОбменСУдаленнымиСкладами.ВозможностьЧтенияДанных(ЧтениеXML) Цикл

        // Прочитать очередное значение.
        Данные = ОбменСУдаленнымиСкладами.ПрочитатьДанные(ЧтениеXML);

        // Проверить, нужно ли принимать сообщение от узла-отправителя.
        // Если нет, отклонить изменения.
        Если Не ОбменСУдаленнымиСкладами.ПринятьИзменения(
                        ЧтениеСообщения.Отправитель, Данные) Тогда
            Сообщение = Новый СообщениеПользователю;
            Сообщение.Текст = Строка(Данные.Метаданные())
                        + ": " + Строка(Данные) + " – Изменения отклонены";
            Сообщение.Сообщить();

            Продолжить;
        КонецЕсли;

        // Записать полученные данные.
        Данные.ОбменДанными.Отправитель = ЧтениеСообщения.Отправитель;
        Данные.ОбменДанными.Загрузка = Истина;
        Данные.Записать();

    КонецЦикла;

    КонецЦикла;
    ...

КонецПроцедуры
```

Добавленный фрагмент выделен жирным шрифтом. В этом фрагменте в цикле мы выполняем чтение данных, содержащихся в сообщении обмена.

Перед тем как записать данные в базу узла, в функции ПринятьИзменения(), расположенной в общем модуле ОбменСУдаленнымиСкладами (см. листинг 3.73), проверяется возможность принятия изменений от узла-отправителя сообщения. В параметре Данные в функцию передается полученный объект обмена, а в параметре Отправитель – ссылка на узел-отправитель сообщения.

Листинг 3.73. Функция для разрешения коллизии при обмене данными

```
Функция ПринятьИзменения(Отправитель, Данные)

    Прием = Истина;

    // Проверить зарегистрировано ли изменение данных для отправителя.
    Если ПланыОбмена.ИзменениеЗарегистрировано(Отправитель, Данные) Тогда

        // Разрешить получение данных только от главного узла.
        Если Не Отправитель.Главный Тогда
            Прием = Ложь;
        КонецЕсли;

    КонецЕсли;
    Возврат Прием;

КонецФункции
```

С помощью метода менеджера планов обмена ИзменениеЗарегистрировано() мы проверяем, зарегистрировано ли изменение объекта для узла-отправителя в нашей базе данных. Если да и если отправитель не является главным узлом, мы отклоняем полученные изменения (возвращается Ложь). Во всех остальных случаях изменения принимаются (возвращается Истина).

На самом деле алгоритм принятия изменений, реализованный в функции ПринятьИзменения(), может быть другим, гораздо более сложным и нестандартным. Но мы показали только самый простой и очевидный пример разрешения коллизии.

Тестирование работы универсального обмена данными

Запустим «1С:Предприятие», введем некоторые тестовые данные и протестируем работу универсальных механизмов обмена данными.

В списке узлов плана обмена Удаленные склады уже существует одна предопределенная запись, соответствующая нашей информационной базе. Это и будет центральный узел плана обмена. Установим его код (ЦентОфис) и название (Центральный офис). Затем добавим в список еще два узла обмена: Оптовый склад (код Опт) и Розничный склад (код Розница).

При создании каждого из этих узлов нужно указать соответствующий ему склад. Значение реквизита Склад будет определять стратегию распространения данных по узлам. Для узла Центральный офис склад будет содержать пустую ссылку, так как в центральный узел должны попадать все данные, независимо от их привязки к складу.

Отметка Главный будет отвечать за результат разрешения коллизий. Сейчас ни для какого узла обмена ставить эту отметку не будем, потому что главным для оптового и розничного узлов является Центральный офис, но он является предопределенным для той информационной базы, в которой мы находимся. Следовательно, его отметка в центральной базе невозможна.

Но вот что нужно сделать обязательно – так это после записи оптового и розничного узлов обмена нажать кнопку Зарегистрировать изменения и тем самым выполнить начальную синхронизацию этих узлов данными центральной базы (рис. 3.25).

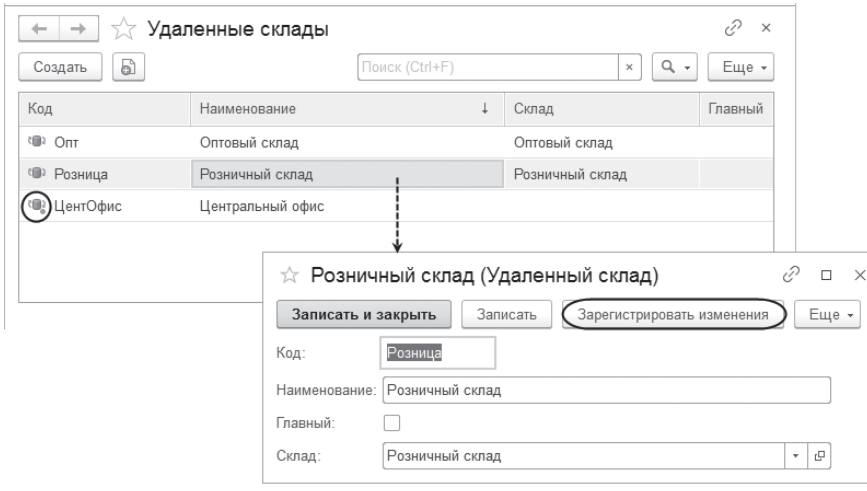


Рис. 3.25. Создание списка узлов плана обмена «Удаленные склады» в центральной базе

Затем откроем форму настроек информационной базы и установим значение константы Префикс номеров как «ЦО», а также зададим путь к каталогу обмена (`e:\Exchange`), который будет храниться в константе Каталог обмена (рис. 3.26).

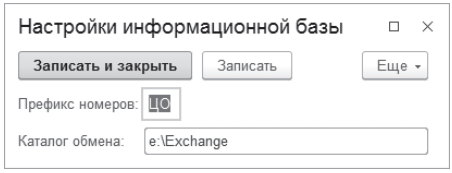


Рис. 3.26. Установка настроек информационной базы

После этого внесем в базу центрального узла новую номенклатуру, контрагента и приходные/расходные накладные так, чтобы было хотя бы по одному документу, относящемуся к каждому из складов.

Теперь нам нужно передать эти изменения узлу Оптовый склад. Для этого откроем обработку Обмен с удаленными складами, в списке узлов плана обмена выберем узел, которому будут передаваться данные (Оптовый склад), и нажмем кнопку Выгрузить данные (рис. 3.27).

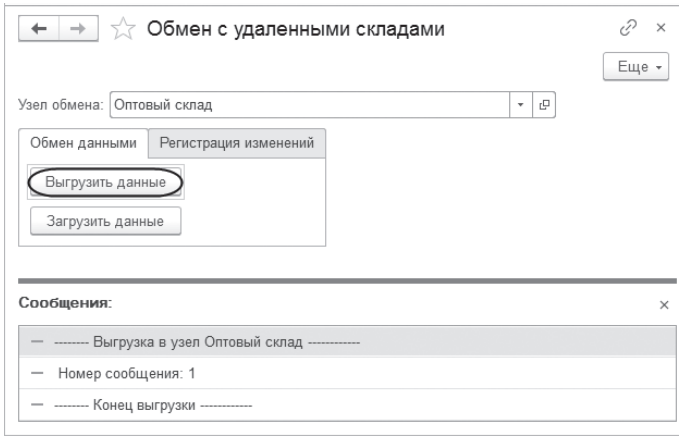


Рис. 3.27. Выгрузка данных из центральной базы в удаленный склад

При этом в каталоге обмена появится файл сообщения обмена с именем Message_ЦентОфис_Опт.xml. Из этого файла изменения будут в дальнейшем прочитаны в узел с кодом Опт.

Но сначала мы должны создать пустую базу этого узла из выгруженной ранее конфигурации. Создадим в этой базе такой же список узлов плана обмена Удаленные склады с такими же кодами узлов, что и в центральной базе. С той лишь разницей, что у узла Центральный офис мы установим флажок Главный, а предопределенным узлом здесь будет являться Оптовый склад (рис. 3.28).

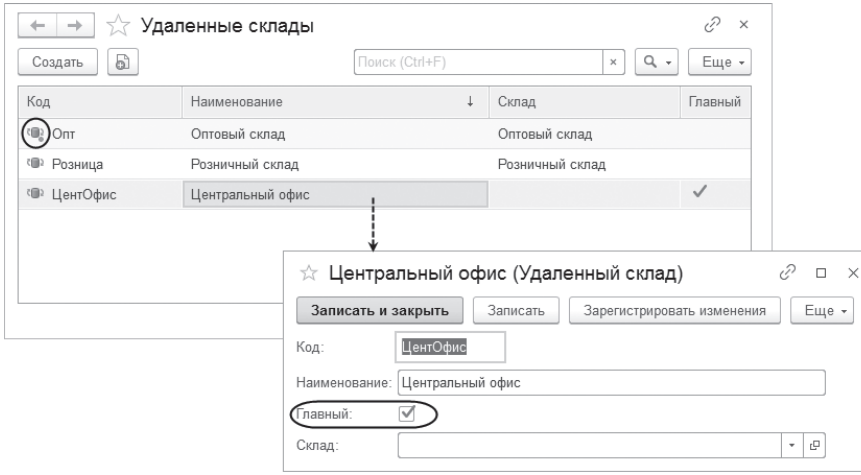


Рис. 3.28. Создание списка узлов плана обмена «Удаленные склады» в базе удаленного склада

Затем откроем форму настроек информационной базы и установим значение константы Префикс номеров как «ОС», а также зададим путь к каталогу обмена (`e:\Exchange`), который будет храниться в константе Каталог обмена.

Теперь откроем обработку Обмен с удаленными складами, в списке узлов плана обмена выберем узел, из которого данные были выгружены (Центральный офис), и нажмем кнопку Загрузить данные (рис. 3.29).

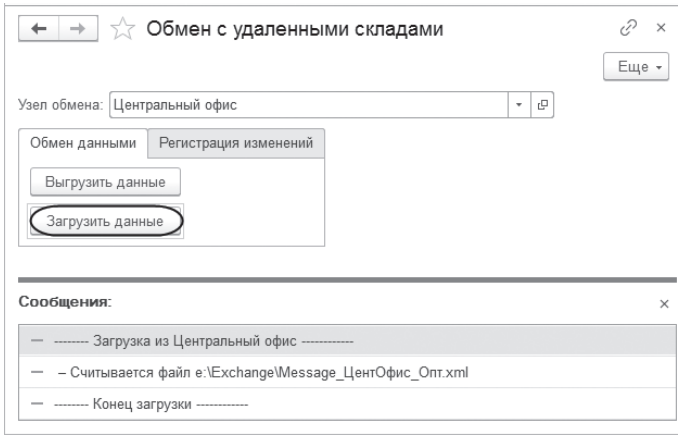


Рис. 3.29. Загрузка данных из центральной базы в удаленный склад

Мы видим, что все данные из центральной базы перенеслись в полном объеме, за исключением приходных и расходных накладных и их движений. Согласно стратегии распространения данных в узел Оптовый склад попали только документы с соответствующим складом.

Данные номенклатуры также перенеслись правильно – из центральной базы не перенесся артикул номенклатуры. Соответствующий реквизит номенклатуры мы не передавали, так как обменивались данными справочника в ручном режиме.

Теперь нам осталось проверить, как работает разрешение коллизий.

Добавим новую номенклатуру и изменим цену существующей номенклатуры, а также изменим дату существующей приходной накладной. После этого откроем обработку Обмен с удаленными складами, в списке узлов плана обмена выберем узел, которому будут передаваться данные (Центральный офис), и нажмем кнопку Выгрузить данные.

В центральной базе изменим те же данные, которые мы изменяли в базе оптового узла, затем откроем обработку Обмен с удаленными складами, в списке узлов плана обмена выберем узел, из которого данные были выгружены (Оптовый склад), и нажмем кнопку Загрузить данные (рис. 3.30).

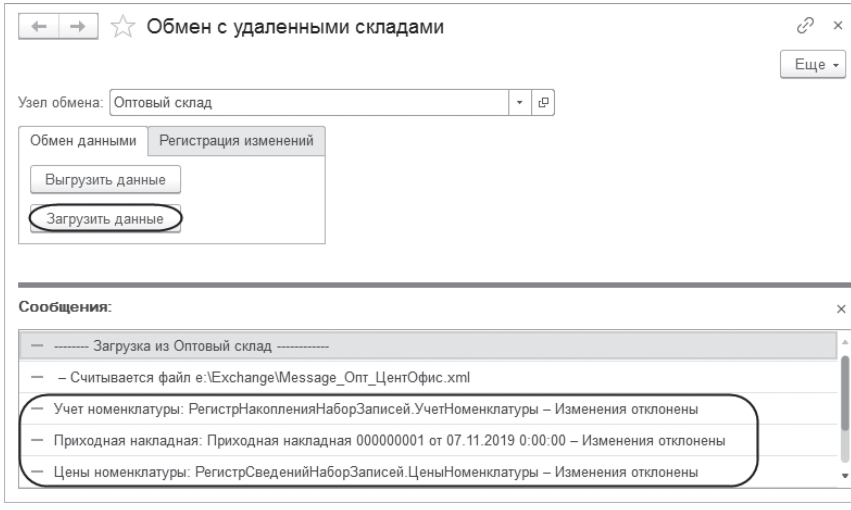


Рис. 3.30. Загрузка данных из удаленного склада в центральную базу

При чтении данных мы получим сообщения о том, что изменения данных, сделанные в оптовом узле, отклонены в пользу изменений центральной базы. Так произошло потому, что центральный офис является главным узлом по отношению к оптовому складу. А вот при передаче данных в другую сторону изменения центральной базы будут приняты и затрут изменения оптового узла.

Теперь проделаем те же действия для узла обмена Розничный склад и убедимся, что все работает правильно.

Протестируем также, как работает удаление данных, которых в узле быть не должно. В розничный склад приходят только накладные с соответствующим складом. Изменим склад документа на оптовый и отправим это изменение в центральный офис. Там оно будет принято, так как центральная база получает все данные независимо от склада. Но при следующей отправке изменений этого документа в розничный склад, во-первых, документ туда отправлен не будет, согласно стратегии распространения данных по узлам, а во-вторых, туда будет послано удаление «неправильного» документа. В результате он исчезнет из «чужого» узла и появится в оптовом складе при следующем сеансе обмена данными.

В заключение выполним обмен данными между розничным и оптовым складом. При использовании универсальных механизмов обмена это возможно. Все работает так же, как и в случае обмена с центральной базой. За исключением разрешения коллизий. Поскольку ни один из узлов обмена не является главным для другого, то изменения, сделанные в «своем» узле, в результате останутся, а изменения «чужого» узла будут отклонены.

Обмен предопределенными данными

До сих пор мы не заостряли внимание на том, что в обмене данными у нас участвует справочник Склады, содержащий предопределенные элементы с именами Основной и Розничный. Однако нужно иметь в виду, что при обмене предопределенными данными в нераспределенной информационной базе могут возникнуть проблемы.

Дело в том, что при создании справочников и других объектов конфигурации, которые могут содержать предопределенные элементы, свойство Обновление предопределенных данных для этих объектов стандартно устанавливается в значение Авто. Это приводит к тому, что при реструктуризации базы данных или при первом обращении к таблице, хранящей данных объекта конфигурации, создаются или обновляются элементы данных, связанных по имени с предопределенными элементами данных в конфигурации.

В нашем случае при открытии списка складов в базе оптового и розничного складов будут автоматически созданы элементы справочника Склады, связанные с предопределенными элементами справочника с именами Основной и Розничный. При этом свойство ИмяПредопределенныхДанных этих элементов справочника будет установлено в значение Основной и Розничный. Изменить эту связь можно только с помощью встроенного языка.

Но затем при загрузке данных из центральной базы в справочник Склады будут добавлены еще две записи, связанные с этими же предопределенными элементами. В результате элементы справочника продублируются из-за того, что данные из центральной базы будут восприняты как новые, так как у них будут другие ссылки.

В дальнейшем это приведет к ошибке, так как в системе не может быть двух объектов, связанных с одинаковым предопределенным элементом. Разработчик должен это отслеживать и не допускать такой ситуации. Например, можно сделать одну из баз ведущей в части обновления предопределенных данных, а в других – это обновление запретить.

Режим обновления предопределенных данных можно установить для конкретного объекта конфигурации с помощью метода встроенного языка УстановитьОбновлениеПредопределенныхДанных(). Либо можно установить свойство объекта конфигурации Обновление предопределенных данных в конфигураторе. А также это можно сделать для всей информационной базы в целом с помощью метода УстановитьОбновлениеПредопределенныхДанныхИнформационнойБазы().

Свойство объектов ОбновлениеПредопределенныхДанных может принимать следующие значения:

- Не обновлять автоматически – в этом случае платформа не выполняет создание или обновление элемента данных при создании или изменении предопределенных данных;
- Обновлять автоматически – в этом случае платформа автоматически создаст (или обновит существующие) элементы данных для новых (или измененных) предопределенных данных;
- Авто – платформа автоматически определяет необходимость выполнения обновления.

Возможность создания и обновления предопределенных данных определяется последовательным анализом режима обновления установленного в данных объекта конфигурации, или при конфигурировании этого объекта, или для всей информационной базы до тех пор, пока не будет обнаружено значение, отличное от значения Авто.

Сказанное касается только нераспределенных баз данных. Если это периферийный узел РИБ, то предопределенные данные не будут обновлены. Если проверка выполняется для центрального узла РИБ, обновление предопределенных данных будет выполнено.

Самое простое – это запретить обновление предопределенных данных справочника складов в конфигураторе. Для этого нужно открыть конфигурацию оптового или розничного склада и в окне редактирования свойств справочника Склады на закладке Прочее установить свойство Обновление предопределенных данных в значение Не обновлять автоматически (рис. 3.31).

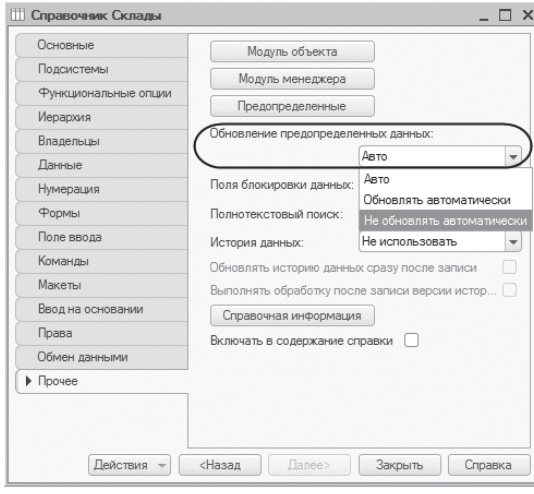


Рис. 3.31. Редактирование свойств справочника «Склады»

Но мы этого делать не будем, а рассмотрим более универсальный вариант решения этой проблемы – на уровне всей информационной базы в целом.

Добавим в конфигурацию константу ИспользоватьОбновлениеПредопределенныхДанных и поместим ее в общую форму констант НастройкиИнформационнойБазы. В модуле менеджера значения этой константы поместим обработчик события `ПриЗаписи` и заполним его следующим образом (листинг 3.74).

Листинг 3.74. Обработчик события «ПриЗаписи»

```
Процедура ПриЗаписи(Отказ)

    Если Значение = Истина Тогда
        УстановитьОбновлениеПредопределенныхДанныхИнформационнойБазы(
            ОбновлениеПредопределенныхДанных.ОбновлятьАвтоматически);
    Иначе
        УстановитьОбновлениеПредопределенныхДанныхИнформационнойБазы(
            ОбновлениеПредопределенныхДанных.НеОбновлятьАвтоматически);
    КонецЕсли;

КонецПроцедуры
```

Запустим «1С:Предприятие» и установим флажок использования обновления предопределенных данных в форме общих настроек центральной базы (рис. 3.32).

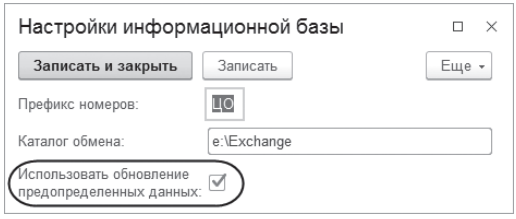


Рис. 3.32. Установка настроек информационной базы

Создадим снова базу оптового и/или розничного склада из конфигурации, откроем окно настроек и запишем этот флажок в снятом состоянии.

Затем откроем список складов. Он будет пустым, так как при записи настроек для всей информационной базы мы установили режим обновления предопределенных данных в значение Не обновлять автоматически.

Теперь выполним чтение начальных данных от центральной базы. После этого в списке складов появятся две записи, полученные из центральной базы, связанные с предопределенными элементами справочника с именами Основной и Розничный. И это правильно.

Как уже говорилось, автоматическое создание предопределенных данных в дочерних узлах распределенной информационной базы не производится. Но следует иметь в виду, что в случае отмены назначения главного узла (например, для выполнения каких-либо специальных действий) такая информационная база не будет считаться периферийной. В связи с этим при первом запуске такой информационной базы в режиме 1С:Предприятие режим создания или обновления предопределенных элементов может быть определен как Обновлять автоматически. Чтобы

избежать этого, можно воспользоваться параметром командной строки `/SetPredefinedDataUpdate` или использовать метод глобального контекста `УстановитьОбновлениеПредопределенныхДанныхИнформационнойБазы()`.

Ручная регистрация изменений

Бывают ситуации, когда автоматическая регистрация изменений некоторых объектов конфигурации не нужна, так как эти изменения требуется регистрировать только при соблюдении каких-то специфических условий. Или же необходимо зарегистрировать изменения объекта для какого-либо определенного перечня узлов, отличного от формируемого при авторегистрации.

В этих случаях можно воспользоваться двумя способами: определить обработчики события `ПередЗаписью` и `ПередУдалением` модуля объекта или программно отключить свойство `Автозаполнение`, очистить набор узлов-получателей и сформировать свой собственный список получателей.

Чтобы продемонстрировать эти возможности, внесем небольшие дополнения в пример, рассмотренный нами выше.

Откроем состав плана обмена УдаленныеСклады и запретим авторегистрацию для документа РасходнаяНакладная и регистра накопления УчетНоменклатуры (рис. 3.33).

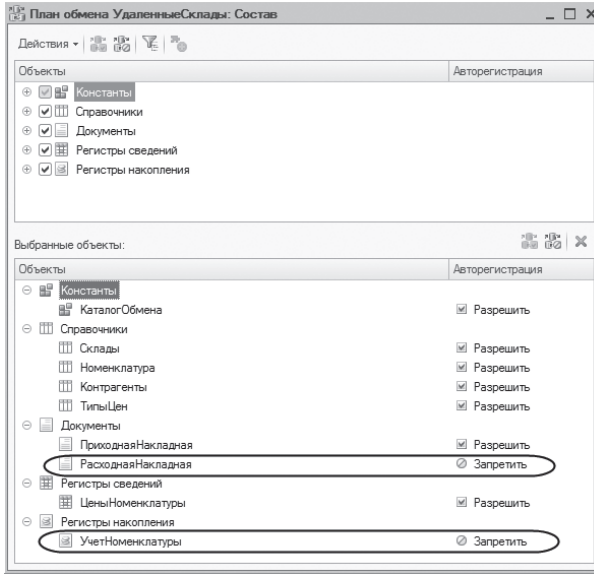


Рис. 3.33. Авторегистрация объектов, входящих в план обмена «УдаленныеСклады»

Таким образом, при изменении этих объектов список узлов-получателей свойства `ОбменДанными` автоматически заполняться не будет. Покажем, как зарегистрировать изменения вручную при соблюдении следующих условий:

- При обмене каждая накладная должна уйти в свой узел (реквизит Склад узла плана обмена должен совпасть со складом, выбранным в документе).
- Пользователям разрешено менять склад в документе. Если документ уже ушел в неправильный узел, он должен в нем удалиться и отправиться в правильный.

Формирование списка узлов-получателей в обработчиках события «ПриЗаписи»

Для реализации первой задачи в модуле объекта РасходнаяНакладная поместим обработчик события `ПередЗаписью` и заполним его следующим образом (листинг 3.75).

Листинг 3.75. Обработчик события «ПередЗаписью» документа «РасходнаяНакладная»

```
Процедура ПередЗаписью(Отказ)

    Если ОбменДанными.Получатели.Автозаполнение Тогда
        ОбменСУдаленнымиСкладами.ВключитьРегистрацию(ОбменДанными, Склад);

        // Проверить, изменился ли склад в существующем документе.
        Если Не ЭтоНовый() Тогда

            Запрос = Новый Запрос("
                |ВЫБРАТЬ
                |    Склад
                |ИЗ
                |    Документ.РасходнаяНакладная
                |ГДЕ
                |    Ссылка = &ТекСсылка");

            Запрос.УстановитьПараметр("ТекСсылка", Ссылка);
            Выборка = Запрос.Выполнить().Выбрать();
            Выборка.Следующий();
            Если Выборка.Склад <> Склад Тогда
                ОбменСУдаленнымиСкладами.ВключитьРегистрацию(
                                    ОбменДанными, Выборка.Склад);
            КонецЕсли;
        КонецЕсли;
    КонецЕсли;

    // Вывести диагностическое сообщение о регистрации изменений.
    ОбменСУдаленнымиСкладами.ВывестиРегистрацию(ЭтотОбъект);

КонецПроцедуры
```

В данном обработчике (в случае включения режима Автозаполнение) производится проверка соответствия выбранного склада в объекте и записанного в информационной базе. В случае расхождения документ регистрируется как для нового узла (в соответствии с измененным складом), так и для старого.

Затем с аналогичной целью поместим в модуле набора записей регистра накопления УчетНоменклатуры обработчик события ПриЗаписи и заполним его следующим образом (листинг 3.76).

Листинг 3.76. Обработчик события «ПередЗаписью» набора записей регистра «УчетНоменклатуры»

```
Процедура ПередЗаписью(Отказ)

    Если ОбменДанными.Получатели.Автозаполнение Тогда
        СкладНабора = Неопределено;

        Для Каждого Запись Из ЭтотОбъект Цикл
            Если СкладНабора = Неопределено
                ИЛИ
                Запись.Склад <> СкладНабора Тогда
                ОбменСУдаленнымиСкладами.ВключитьРегистрацию(ОбменДанными, Запись.Склад);
            КонецЕсли;
            СкладНабора = Запись.Склад;
        КонецЦикла;

        // Выбрать все различные склады регистра с отбором по регистратору.
        Запрос = Новый Запрос;
        Запрос.Текст = "ВЫБРАТЬ Различные
            |    Склад
            |ИЗ
            |    РегистрНакопления.УчетНоменклатуры
            |ГДЕ
            |    Регистратор = &ТекСсылка";
        Запрос.УстановитьПараметр("ТекСсылка", Отбор.Регистратор.Значение);
        Выборка = Запрос.Выполнить().Выбрать();
        Пока Выборка.Следующий() Цикл

            // Включить регистрацию изменений набора записей для каждого склада.
            ОбменСУдаленнымиСкладами.ВключитьРегистрацию(ОбменДанными, Выборка.Склад);

        КонецЦикла;
    КонецЕсли;

    // Вывести диагностическую информацию по набору записей.
    ОбменСУдаленнымиСкладами.ВывестиРегистрацию(ЭтотОбъект);

КонецПроцедуры
```

Обе эти процедуры используют процедуры `ВключитьРегистрацию()` и `ВывестиРегистрацию()`. Эти процедуры мы поместим в общем модуле ОбменСУдаленнымиСкладами и заполним следующим образом (листинг 3.77, 3.78).

Листинг 3.77. Процедура «ВывестиРегистрацию()»

```
Процедура ВывестиРегистрацию(Данные) Экспорт

	ОбменДанными = Данные.ОбменДанными;
	Сообщение = Новый СообщениеПользователю();
	Сообщение.Текст = " - Регистрация " + Данные.Метаданные() + ": " + Строка(Данные);
	Сообщение.Сообщить();

	// Вывести список узлов, куда будет выполняться регистрация объекта.
	Для Каждого Получатель Из ОбменДанными.Получатели Цикл
		Если Не ОбменДанными.Отправитель = Получатель Тогда
			Сообщение.Текст = " - " + Получатель;
			Сообщение.Сообщить();
		КонецЕсли;
	КонецЦикла;

КонецПроцедуры
```

Процедура `ВывестиРегистрацию()` выводит в окно сообщений данные об объекте, регистрация которого произошла, и узлах-получателях, для которых эта регистрация произошла.

Листинг 3.78. Процедура «ВключитьРегистрацию()»

```
Процедура ВключитьРегистрацию(ОбменДанными, Склад) Экспорт

	// Получить список узлов для конкретного склада.
	Запрос = Новый Запрос;
	Запрос.Текст = "ВЫБРАТЬ
		|	УдаленныеСклады.Ссылка
		|ИЗ
		|	ПланОбмена.УдаленныеСклады КАК УдаленныеСклады
		|
		|ГДЕ
		|	(УдаленныеСклады.Склад = &Склад ИЛИ
		|	УдаленныеСклады.Склад = &ПустойСклад)
		|	И УдаленныеСклады.ЭтотУзел = ЛОЖЬ";

	Запрос.УстановитьПараметр("Склад", Склад);
	Запрос.УстановитьПараметр("ПустойСклад", Справочники.Склады.ПустаяСсылка());
	Выборка = Запрос.Выполнить().Выбрать();

	// Зарегистрировать изменения объекта для выбранных узлов.
	Пока Выборка.Следующий() Цикл
		ОбменДанными.Получатели.Добавить(Выборка.Ссылка);
	КонецЦикла;

КонецПроцедуры
```

Задача процедуры ВключитьРегистрацию() – определить всех возможных получателей регистрируемого изменения. В этот список попадут те узлы, у которых в реквизите Склад содержится ссылка на интересующий нас склад либо пустая ссылка (что расширяет функциональность решения). И, естественно, этот получатель не должен быть предопределенным узлом данного плана обмена.

Регистрация изменений для узла в обработке для выполнения обмена

До сих пор мы выполняли первоначальную регистрацию изменений для узла с помощью метода менеджера планов обмена ЗарегистрироватьИзменения() в форме узла обмена, сразу же после его создания.

Теперь нам нужно изменить процедуру регистрации изменений для узла с учетом ручной регистрации объектов. Будем выполнять полную регистрацию изменений для выбранного узла в обработке для выполнения обмена ОбменСУдаленнымиСкладами. А также нам понадобится возможность удаления всех записей регистрации изменений для узла.

Для этого добавим в форму обработки команды УдалитьРегистрациюИзменений и ПолнаяРегистрацияИзменений и перетащим эти команды в окно элементов формы. У кнопок, соответствующих командам, снимем флажок у свойства Доступность. Их доступность будет включаться при выборе непредопределенного узла обмена в форме обработки.

Для этого обработчик события ПриИзменении для поля формы УзелОбмена дополним следующим образом (листинг 3.79).

Листинг 3.79. Обработчик события «ПриИзменении» поля «УзелОбмена»

```
&НаКлиенте
Процедура УзелОбменаПриИзменении(Элемент)

    ДоступностьКнопок = ПолучитьДоступностьПоУзлу(УзелОбмена);

    // Установить доступность кнопок в зависимости от установленного узла обмена.
    Элементы.ЗагрузитьДанные.Доступность = ДоступностьКнопок;
    Элементы.ВыгрузитьДанные.Доступность = ДоступностьКнопок;
    Элементы.УдалитьРегистрациюИзменений.Доступность = ДоступностьКнопок;
    Элементы.ПолнаяРегистрацияИзменений.Доступность = ДоступностьКнопок;

КонецПроцедуры
```

Теперь создадим обработчик команды УдалитьРегистрациюИзменений и заполним следующим образом (листинг 3.80).

Листинг 3.80. Обработчик команды «УдалитьРегистрациюИзменений»

```
&НаКлиенте
Процедура УдалитьРегистрациюИзменений(Команда)

    ОбменСУдаленнымиСкладами.ПолноеУдалениеРегистрацииДанныхПоУзлу(УзелОбмена);

КонецПроцедуры
```

В этом обработчике мы вызываем процедуру `ПолноеУдалениеРегистрацииДанныхПоУзлу()`, в которую передаем ссылку на узел обмена, для которого должна произойти очистка таблиц регистрации изменений. Эту процедуру мы поместим в общем модуле ОбменСУдаленнымиСкладами и заполним следующим образом (листинг 3.81).

Листинг 3.81. Удаление регистрации изменений для узла

```
Процедура ПолноеУдалениеРегистрацииДанныхПоУзлу(Узел) Экспорт

    Сообщение = Новый СообщениеПользователю();
    Сообщение.Текст = "Удаление регистрации всех данных для узла " + Узел;
    Сообщение.Сообщить();
    ПланыОбмена.УдалитьРегистрациюИзменений(Узел);

КонецПроцедуры
```

Теперь создадим обработчик команды ПолнаяРегистрацияИзменений и заполним следующим образом (листинг 3.82).

Листинг 3.82. Обработчик команды «ПолнаяРегистрацияИзменений»

```
&НаКлиенте
Процедура ПолнаяРегистрацияИзменений(Команда)

    ОбменСУдаленнымиСкладами.ПолнаяРегистрацияДанныхПоУзлу(УзелОбмена, Ложь);

КонецПроцедуры
```

В этом обработчике мы вызываем процедуру `ПолнаяРегистрацияДанныхПоУзлу()`, в которую передаем ссылку на узел обмена, для которого необходимо выполнить регистрацию изменений. Эту процедуру мы поместим в общем модуле ОбменСУдаленнымиСкладами и заполним следующим образом (листинг 3.83).

Листинг 3.83. Полная регистрация изменений для узла

```
Процедура ПолнаяРегистрацияДанныхПоУзлу(Узел, РегистрироватьВсе) Экспорт

    // Зарегистрировать изменения всех данных для узла, в случае если
    // значения параметра РегистрироватьВсе = Истина или
    // значение склада в узле не установлено.
    Сообщение = Новый СообщениеПользователю();

    Если РегистрироватьВсе Или Узел.Склад.Пустая() Тогда
        Сообщение.Текст = "Регистрация всех данных по узлу " + Узел;
        Сообщение.Сообщить();
        ПланыОбмена.ЗарегистрироватьИзменения(Узел);
    Иначе
        Сообщение.Текст = "Регистрация данных по узлу " + Узел + " ----------";
        Сообщение.Сообщить();
        ПланОбмена = Узел.Метаданные();
        Состав = ПланОбмена.Состав;

        Для Каждого ЭлементСостава Из Состав Цикл
            Сообщение.Текст = "   Регистрация данных " + ЭлементСостава.Метаданные
                              + " по узлу " + Узел;
            Сообщение.Сообщить();

            Если ЭлементСостава.Метаданные = Метаданные.Документы.РасходнаяНакладная Тогда

                // Зарегистрировать изменения документов РасходнаяНакладная для узла,
                // в случае если
                // склад узла совпадает со складом накладной.
                // Выбрать документы РасходнаяНакладная с отбором по складу узла.
                Запрос = Новый Запрос("ВЫБРАТЬ
                    |    Ссылка
                    |ИЗ
                    |    Документ.РасходнаяНакладная
                    |ГДЕ
                    |    Склад = &Склад");

                Запрос.УстановитьПараметр("Склад", Узел.Склад);
                Выборка = Запрос.Выполнить().Выбрать();
                Пока Выборка.Следующий() Цикл
                    // Зарегистрировать изменение документа для узла.
                    ПланыОбмена.ЗарегистрироватьИзменения(Узел, Выборка.Ссылка);
                КонецЦикла;

            ИначеЕсли ЭлементСостава.Метаданные
                    = Метаданные.РегистрыНакопления.УчетНоменклатуры Тогда

                // Выбрать все различные регистраторы регистра.
                Запрос = Новый Запрос("ВЫБРАТЬ РАЗЛИЧНЫЕ
                    |    Регистратор
                    |ИЗ
                    |    РегистрНакопления.УчетНоменклатуры
                    |ГДЕ
                    |    Склад = &Склад");
```

```
                    Запрос.УстановитьПараметр("Склад", Узел.Склад);
                    Выборка = Запрос.Выполнить().Выбрать();
                    Пока Выборка.Следующий() Цикл
                        // Создать набор записей регистра с отбором по регистратору.
                        НаборЗаписей = РегистрыНакопления
                            .УчетНоменклатуры.СоздатьНаборЗаписей();
                        НаборЗаписей.Отбор.Регистратор.Установить(Выборка.Регистратор);

                        // Зарегистрировать изменения набора записей для узла.
                        ПланыОбмена.ЗарегистрироватьИзменения(Узел, НаборЗаписей);
                    КонецЦикла;

                Иначе

                    // Зарегистрировать изменения всех данных объекта для узла.
                    ПланыОбмена.ЗарегистрироватьИзменения(Узел, ЭлементСостава.Метаданные);
                КонецЕсли;

            КонецЦикла;

        КонецЕсли;

КонецПроцедуры
```

Во втором параметре `РегистрироватьВсе` в процедуру `ПолнаяРегистрацияДанныхПоУзлу()` передается значение типа `Булево`. Если это значение `Истина`, то для данного узла будет выполняться регистрация изменения всех объектов (невзирая на принадлежность данного узла к какому-либо определенному складу). Такой же вариант регистрации будет осуществлен, если в качестве первого параметра будет передан узел, у которого в реквизите Склад содержится пустая ссылка.

Полная регистрация всех объектов производится с помощью метода менеджера планов обмена `ЗарегистрироватьИзменения()` без указания второго параметра (его значение по умолчанию `Неопределено`). При этом на отключенную авторегистрацию (для документов РасходнаяНакладная и наборов записей регистра накопления УчетНоменклатуры) внимания не обращается.

Если в качестве второго параметра передается значение `Ложь`, расходные накладные и записи регистра накопления регистрируются как измененные только в случае совпадения склада со складом, определенным в узле плана обмена. Все остальные данные регистрируются обычным образом.

Проверим, как это работает. В центральной базе в обработке Обмен с удаленными складами выберем узел обмена Розничный склад. Сначала удалим всю регистрацию изменений для этого узла обмена с помощью кнопки Удалить регистрацию изменений, а потом снова выполним полную регистрацию изменений по этому узлу с помощью кнопки Полная регистрация изменений. Затем нажмем кнопку Выгрузить данные и запишем все изменения, зарегистрированные в центральной базе для розничного склада в файл выгрузки.

После этого создадим пустую базу розничного склада из выгруженной ранее конфигурации. В этой базе создадим такой же список узлов плана обмена Удаленные склады с такими же кодами узлов, что и в центральной базе. Установим префикс номеров и каталог обмена в форме настроек информационной базы так же, как мы это делали раньше. В обработке Обмен с удаленными складами выберем узел обмена Центральный офис, нажмем кнопку Загрузить данные и получим все изменения, переданные от центральной базы.

Убедимся, что ручная регистрация изменений работает правильно. При этом изменения расходных накладных и движений регистра накопления просто не регистрируются для узлов обмена с несоответствующим складом. То есть они отфильтровываются не на уровне записи в сообщение обмена, а сразу – на уровне формирования таблиц регистрации изменений.

Регистрация в произвольные узлы

В нашем примере обмена данными для документов ПриходнаяНакладная была включена автоматическая регистрация изменений (см. рис. 3.33). Но иногда требуется зарегистрировать изменения для какого-либо определенного перечня узлов, отличного от формируемого при авторегистрации.

В этом случае можно программно отключить свойство `Автозаполнение`, очистить набор узлов-получателей и сформировать свой собственный список получателей.

Для этого добавим в форму обработки еще одну команду ЗарегистрироватьПриходные и перетащим ее в окно элементов формы. У соответствующей кнопки будем устанавливать доступность при выборе непредопределенного узла обмена в форме обработки. Это делается в обработчике события `ПриИзменении` поля формы УзелОбмена, как это было показано в предыдущем разделе (см. листинг 3.79).

Теперь создадим обработчик команды ЗарегистрироватьПриходные и заполним следующим образом (листинг 3.84).

Листинг 3.84. Обработчик команды «ЗарегистрироватьПриходные»

```
&НаКлиенте
Процедура ЗарегистрироватьПриходные(Команда)

    УзлыПолучатели = Новый Массив;
    УзлыПолучатели.Добавить(УзелОбмена);

    ЗарегистрироватьПриходныеНаСервере(УзлыПолучатели);

КонецПроцедуры
```

В этом обработчике мы вызываем процедуру ЗарегистрироватьПриходныеНаСервере(), в которую передаем массив узлов, для которых требуется зарегистрировать приходные накладные. Для упрощения примера мы будем добавлять в этот массив только одно значение – ссылку на узел обмена, выбранный в форме обработки (листинг 3.85).

Листинг 3.85. Процедура «ЗарегистрироватьПриходныеНаСервере»

```
&НаСервереБезКонтекста
Процедура ЗарегистрироватьПриходныеНаСервере(Узлы)

	Выборка = Документы.ПриходнаяНакладная.Выбрать();
	Пока Выборка.Следующий() Цикл
		Объект = Выборка.ПолучитьОбъект();

		// Добавить набор узлов в коллекцию получателей
		Объект.ОбменДанными.Получатели.АвтоЗаполнение = Ложь;
		ЗаполнитьНаборУзлов(Объект.ОбменДанными.Получатели, Узлы);

		Объект.Записать();

	КонецЦикла;

КонецПроцедуры
```

В теле процедуры производится выборка всех документов ПриходнаяНакладная. У каждого полученного объекта отключается автозаполнение, и набор получателей формируется самостоятельно с помощью процедуры ЗаполнитьНаборУзлов(), в которую передается коллекция узлов-получателей, содержащаяся в свойстве объекта ОбменДанными, и массив узлов, для которых требуется зарегистрировать приходные накладные (листинг 3.86).

Листинг 3.86. Процедура «ЗаполнитьНаборУзлов()»

```
Процедура ЗаполнитьНаборУзлов(Набор, Список)

	Набор.Очистить();
	Для каждого Элемент Из Список Цикл
		Набор.Добавить(Элемент);
	КонецЦикла;

КонецПроцедуры
```

В этой процедуре коллекция узлов-получателей, для которых регистрируются изменения объекта, очищается. И в цикле обхода элементов массива, полученного в параметре Список, эта коллекция заполняется заново новыми значениями.

При такой регистрации будут зарегистрированы изменения всех приходных накладных, невзирая на соответствие склада документа и склада узла обмена. Но при выгрузке данных в узел «неправильные» накладные будут отсеяны согласно стратегии распространения данных, используемой в этом примере обмена данными.

Использование транзакций при организации обмена

При реализации обмена, как посредством универсального механизма обмена, так и с использованием механизма распределенных информационных баз, одна из задач, которые требуют решения, – это обеспечение целостности и согласованности данных.

Данная задача при использовании универсального обмена может решаться с использованием транзакций (листинг 3.87).

Листинг 3.87. Пример использования транзакций

```
НачатьТранзакцию();

// обработка данных
...

ЗафиксироватьТранзакцию();
```

При выгрузке данных в механизме распределенных информационных баз используется метод `ЗаписатьИзменения()` менеджера планов обмена (листинг 3.88).

Листинг 3.88. Пример использования метода «ЗаписатьИзменения()»

```
ПланыОбмена.ЗаписатьИзменения(ЗаписьСообщения, 0);
```

Второй параметр метода указывает количество элементов, записываемых в одной транзакции. Значение `0` указывает на то, что запись всех измененных данных производится в одной транзакции.

Такой подход (когда вся выгрузка или загрузка осуществляется в одной транзакции) обладает как определенными преимуществами, так и определенными недостатками.

Преимущества такого подхода заключаются в том, что, например, в файловом варианте работы действия, сгруппированные в одну транзакцию, выполняются значительно быстрее (до определенного предела). Кроме того, при

выгрузке данных использование одной транзакции позволяет избежать несогласованности выгружаемых данных (например, когда после выгрузки документа, но до выгрузки наборов записей регистров произошло перепроведение документа).

Минусом такого подхода является снижение параллельности работы пользователей, так как выгружаемые данные будут заблокированы до окончания транзакции.

Кроме того, необходимо учитывать объем изменений, выполняемых в одной транзакции. Например, в файловом варианте все изменения, произведенные транзакцией, накапливаются в оперативной памяти, что при записи больших объемов данных может привести к исчерпыванию свободной памяти.

В клиент-серверном варианте такой опасности нет, но все-таки записывать большие объемы данных в одной транзакции не рекомендуется (из-за проблем параллельности блокировки тоже требуют ресурса сервера баз данных и т. п.).

При обмене большими порциями данных имеет смысл использовать несколько транзакций при загрузке или выгрузке данных.

Например, в алгоритме загрузки данных можно использовать следующий фрагмент кода, позволяющий разбивать процесс загрузки на несколько транзакций по 1000 элементов данных (в данном случае величина 1000 – условная) – листинг 3.89.

Листинг 3.89. Пример использования нескольких транзакций при выгрузке данных

```
Счетчик = 0;
НачатьТранзакцию();

// обработка данных
...

Если Счетчик = 1000 Тогда
        Счетчик = 0;
        ЗафиксироватьТранзакцию();
        НачатьТранзакцию();
КонецЕсли;

// обработка данных
...
ЗафиксироватьТранзакцию();
```

В случаях использования механизма распределенных информационных баз количество объектов указывается вторым параметром метода `ЗаписатьИзменения()` (листинг 3.90).

Листинг 3.90. Пример указания количества элементов, записываемых в одной транзакции

```
ПланыОбмена.ЗаписатьИзменения(ЗаписьСообщения, 1000);
```

Следует отметить, что зачастую в каждом конкретном случае необходимо искать компромисс между скоростью и параллельностью работы пользователей.

Методика включения в сообщение обмена дополнительной информации

В некоторых случаях обмен данными должен сопровождаться передачей в сообщении обмена служебной дополнительной информации. Передачу подобной информации можно осуществлять непосредственно в теле сообщения обмена. Дополнительную информацию можно размещать в теле сообщения как при реализации универсального обмена, так и в случае работы с распределенными информационными базами.

При универсальном обмене это делается довольно просто. При выгрузке данных (в любой момент: в начале сообщения, в середине, в конце) можно записать в тело сообщения специализированный элемент XML (с нужным наполнением) (листинг 3.91).

Листинг 3.91. Пример записи служебной информации

```
ЗаписьXML.ЗаписатьНачалоЭлемента("info");
ЗаписьXML.ЗаписатьАтрибут("содержание", "Дополнительная информация");
ЗаписьXML.ЗаписатьТекст("передаваемая информация");
ЗаписьXML.ЗаписатьКонецЭлемента();
```

При чтении необходимо предусмотреть возможность чтения данного специализированного узла (листинг 3.92).

Листинг 3.92. Пример чтения служебной информации

```
Если ЧтениеXML.Имя="info" Тогда

    // Прочитать текст xml элемента.
    ЧтениеXML.Прочитать();

    // Прочитать конец элемента info.
    ЧтениеXML.Прочитать();

    // Спозиционироваться на начале элемента v8de:Changes
    ЧтениеXML.Прочитать();

КонецЕсли;
```

При необходимости указания дополнительных данных при работе с распределенной информационной базой можно использовать следующий подход (листинг 3.93).

Листинг 3.93. Пример передачи дополнительной информации в сообщении распределенной информационной базы

```
// Создать объект записи XML.
ЗаписьXML = Новый ЗаписьXML;
ЗаписьXML.ОткрытьФайл("c:\out.xml");
ЗаписьXML.ЗаписатьОбъявлениеXML();

// Создать новое сообщение.
Узел = ПланыОбмена.Магазины.НайтиПоКоду("Магазин");
ЗаписьСообщения = ПланыОбмена.СоздатьЗаписьСообщения();
ЗаписьСообщения.НачатьЗапись(ЗаписьXML, Узел);

// Записать дополнительную информацию.
ЗаписьXML.ЗаписатьНачалоЭлемента("info");
ЗаписьXML.ЗаписатьАтрибут("содержание", "Код");
ВключитьСтроку = "
        |Событие = Документы.Событие.СоздатьДокумент();
        |Событие.Дата = ТекущаяДата();
        |Событие.ОписаниеСобытия = ""Сформировать внутренние заказы"";
        |// заполнение других реквизитов документа
        |Событие.Записать(); ";
ЗаписьXML.ЗаписатьТекст(ВключитьСтроку);
ЗаписьXML.ЗаписатьКонецЭлемента();

// Записать изменения.
ПланыОбмена.ЗаписатьИзменения(ЗаписьСообщения, 0);

ЗаписьСообщения.ЗакончитьЗапись();
ЗаписьXML.Закрыть();
```

В результате выполнения получаем следующее сообщение (приведен начальный фрагмент) – листинг 3.94.

Листинг 3.94. Фрагмент сообщения обмена

```
<v8msg:Message xmlns:v8msg="http://v8.1c.ru/messages">
        <v8msg:Header>
                <v8msg:ExchangePlan>Магазины</v8msg:ExchangePlan>
                <v8msg:To>Магазин</v8msg:To>
                <v8msg:From>ЦентрОфис</v8msg:From>
                <v8msg:MessageNo>1</v8msg:MessageNo>
                <v8msg:ReceivedNo>0</v8msg:ReceivedNo>
        </v8msg:Header>
        <v8msg:Body
                <info содержание="Код">
                        Событие=Документы.Событие.СоздатьДокумент();
                        Событие.Дата=ТекущаяДата();
                        Событие.ОписаниеСобытия="Сформировать внутренние заказы";
```

```
            // заполнение других реквизитов документа
            Событие.Записать(); ";
        </info>
        <v8de:Changes
            xmlns:v8="http://v8.1c.ru/data"
            xmlns:v8de="http://v8.1c.ru/dataexchange/2005/02"
            xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                <v8de:Signature>
                    b16e45e5-1a9b-48b3-811f-286a45f6dd50
                </v8de:Signature>
```

Чтение сообщения обмена, содержащего дополнительную информацию, может быть выполнено так, как показано в листинге 3.95.

Листинг 3.95. Пример чтения сообщения обмена

```
// Создать объект чтения XML.
ЧтениеXML = Новый ЧтениеXML;
ЧтениеXML.ОткрытьФайл("c:\out.xml");

// Загрузить сообщение из найденного файла.
ЧтениеСообщения = ПланыОбмена.СоздатьЧтениеСообщения();
ЧтениеСообщения.НачатьЧтение(ЧтениеXML);

// Прочитать текст элемента xml.
ЧтениеXML.Прочитать();

// Прочитать конец элемента info.
ЧтениеXML.Прочитать();

// Спозиционироваться на начале элемента v8de:Changes.
ЧтениеXML.Прочитать();

// Прочитать изменения.
ПланыОбмена.ПрочитатьИзменения(ЧтениеСообщения);

ЧтениеСообщения.ЗакончитьЧтение();
ЧтениеXML.Закрыть();
```

Обратите внимание, что после начала чтения сообщения объект `ЧтениеXML` уже позиционирован на начале элемента info (можно сразу производить выборку атрибутов элемента).

Следует отметить, что алгоритм как записи, так и чтения (в обоих вариантах обмена) зависит от структуры элемента, специализирующегося на «переносе» дополнительной информации.

Организация одностороннего обмена

В некоторых случаях нет необходимости организовывать полноценный обмен данными между двумя узлами − достаточно передавать данные в одном направлении (подобная задача может возникнуть как при организации универсального обмена, так и в контексте распределенной информационной базы). Однако для правильного функционирования механизмов обмена и инфраструктуры сообщений, в частности, необходимо получение ответных сообщений, содержащих квитанции о доставке данных.

Все механизмы реализации одностороннего обмена можно разбить на две большие группы:

- без ответных квитанций;
- с ответными сообщениями, содержащими только квитанции.

Для реализации одностороннего обмена без необходимости получения ответных сообщений можно воспользоваться схемой обмена с гарантированной доставкой. Суть этой схемы в следующем: после формирования сообщения обмена производится удаление регистрации изменений. Этим действием подтверждается, что данные (записанные в только что сформированное сообщение) будут гарантированно доставлены адресату (листинг 3.96).

Листинг 3.96. Пример удаления регистрации изменений

```
Узел = ПланыОбмена.Магазины.НайтиПоКоду("Магазин");
ПланыОбмена.УдалитьРегистрациюИзменений(Узел);
```

В случае если данный механизм используется в распределенных базах данных, следует помнить, что метод `УдалитьРегистрациюИзменений()` не очищает изменения, связанные со структурой конфигурации. То есть если конфигурация изменяется (и используется распределенная информационная база), отказаться от передачи ответных квитанций не получится.

Если рассматривать схемы, связанные с передачей ответного сообщения (содержащего только квитанцию), то можно выделить следующие варианты решения:

- Отсутствие регистрации изменений данных в информационной базе, выступающей в роли узла-приемника, − сообщение обмена содержит только квитанцию о доставке (данный вариант доступен только при реализации универсального обмена).
- Фильтрация выгружаемых данных из информационной базы, выступающей в роли узла-приемника, − сообщение обмена содержит только

квитанцию о доставке (рекомендуется в случае использования распределенных информационных баз, так как остается необходимость получать изменения конфигурации) – листинг 3.97.

- Данные просто не включаются в сообщение (можно использовать в распределенных информационных базах, для уведомления о приеме изменений конфигурации, в случае если в источнике используется принудительная очистка таблиц регистрации изменений) – листинг 3.98. В результате в файл обмена включается только заголовок сообщения (листинг 3.99).
- Фильтрация принимаемых данных от информационной базы, выступающей в роли узла-приемника, − при приеме отрабатывается только квитанция о доставке.

Последний вариант является наименее предпочтительным, так как предполагает передачу фактически ненужных данных.

Листинг 3.97. Процедура «ПриОтправкеДанныхПодчиненному()»

```
// В зависимости от направления можно использовать либо
// событие ПриОтправкеДанныхПодчиненному, либо ПриОтправкеДанныхГлавному
Процедура ПриОтправкеДанныхПодчиненному(ЭлементДанных, ОтправкаЭлемента)

    ОтправкаЭлемента = ОтправкаЭлементаДанных.Игнорировать;

КонецПроцедуры
```

Листинг 3.98. Пример записи сообщения обмена

```
ЗаписьXML = Новый ЗаписьXML;
ЗаписьXML.ОткрытьФайл("c:\out.xml");
ЗаписьXML.ЗаписатьОбъявлениеXML();

// Создать новое сообщение.
Узел=ПланыОбмена.Магазины.НайтиПоКоду("Магазин");
ЗаписьСообщения = ПланыОбмена.СоздатьЗаписьСообщения();
ЗаписьСообщения.НачатьЗапись(ЗаписьXML, Узел);
ЗаписьСообщения.ЗакончитьЗапись();
ЗаписьXML.Закрыть();
```

Листинг 3.99. Фрагмент сообщения обмена

```
<v8msg:Message xmlns:v8msg="http://v8.1c.ru/messages">
    <v8msg:Header>
        <v8msg:ExchangePlan>Магазины</v8msg:ExchangePlan>
        <v8msg:To>Магазин</v8msg:To>
        <v8msg:From>ЦентрОфис</v8msg:From>
        <v8msg:MessageNo>6</v8msg:MessageNo>
        <v8msg:ReceivedNo>0</v8msg:ReceivedNo>
    </v8msg:Header>
    <v8msg:Body/>
</v8msg:Message>
```

Примеры реализации автоматического обмена данными

В процессе использования механизмов обмена данными часто возникает необходимость выполнять процедуру обмена автоматически (например, каждую ночь в определенные часы). В данном разделе мы рассмотрим несколько возможных вариантов организации автоматического обмена данными.

Использование регламентных заданий

Автоматический обмен может быть реализован при помощи регламентных заданий, которые выполняются платформой «1С:Предприятие» по определенному расписанию. Допустим, каждый день в начале работы системы центральной базе необходимо обмениваться данными с узлом плана обмена Оптовый.

Для этого в конфигурации центральной базы создадим объект `РегламентноеЗадание` с именем АвтоматическийОбменДанными. Установим его свойство Предопределенное и зададим расписание его выполнения – Выполнять: каждый день; с 8:00 один раз в день (рис. 3.34).

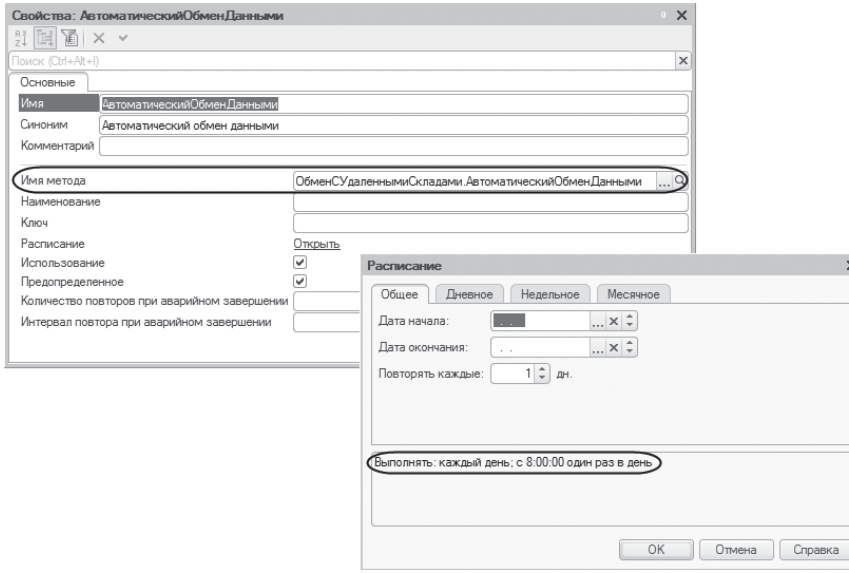


Рис. 3.34. Свойства и расписание регламентного задания

В свойстве Имя метода укажем процедуру общего неглобального модуля, которая будет вызываться при выполнении этого регламентного задания. Процедуру АвтоматическийОбменДанными() поместим в общем модуле ОбменСУдаленнымиСкладами и заполним следующим образом (листинг 3.100).

Листинг 3.100. Процедура «АвтоматическийОбменДанными()»

```
Процедура АвтоматическийОбменДанными() Экспорт

    Узел = ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Опт");
    // Получить объект узла обмена.
    УзелОбмена = Узел.ПолучитьОбъект();
    // Прочитать новое сообщение обмена.
    УзелОбмена.ПрочитатьСообщениеСИзменениями();
    // Записать новое сообщение обмена.
    УзелОбмена.ЗаписатьСообщениеСИзменениями();

КонецПроцедуры
```

В процедуре выполняется поиск узла обмена Оптовый (с кодом Опт) и производится обмен данными с этим узлом в обе стороны с помощью методов плана обмена УдаленныеСклады ПрочитатьСообщениеСИзменениями() и ЗаписатьСообщениеСИзменениями(). Эти методы были подробно рассмотрены в разделе «Пример реализации универсального обмена».

Таким образом, в начале рабочего дня, при соединении с центральной информационной базой, при наступлении указанного времени будет запускаться регламентное задание, организующее автоматический обмен данными с узлом плана обмена Оптовый.

Использование объекта «COMСоединение»

Автоматический обмен может быть реализован при помощи внешней программы, использующей возможности объекта COMСоединение платформы «1С:Предприятие». Данный метод может быть использован, когда изменение конфигурации (для внедрения кода поддержки автоматического обмена) по каким-либо причинам невозможно или нежелательно.

Для примера напишем программу на языке Visual Basic для выполнения обмена, аналогичного описанному в разделе «Использование регламентных заданий» (листинг 3.101).

Листинг 3.101. Пример процедуры обмена

```
Dim connector = CreateObject("V82.COMConnector")
Dim connection = connector.connect("file=d:\DemoExchange")
Dim nodeRef = connection.ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Опт")
If (Not nodeRef.Пустая()) Then
        Dim node = nodeRef.ПолучитьОбъект()
        node.ПрочитатьСообщениеСИзменениями()
        node.ЗаписатьСообщениеСИзменениями()
End If
```

В данном примере используются те же процедуры узлов плана обмена УдаленныеСклады, что и в реализации обмена с использованием командной строки.

Полученный исполняемый модуль может быть поставлен в очередь планировщика. Пример на языке JavaScript приведен в листинге 3.102.

Листинг 3.102. Пример постановки задания в очередь

```
<%@ Language=javascript %>
<%
        entConn = new ActiveXObject("v82.ComConnector");
        conn = entConn.connect("file=d:\DemoExchange");
        nodeRef = conn.ПланыОбмена.УдаленныеСклады.НайтиПоКоду("Опт");
        if (nodeRef.Пустая() == false)
        {
                node = nodeRef.ПолучитьОбъект();
                node.ПрочитатьСообщениеСИзменениями();
                node.ЗаписатьСообщениеСИзменениями();
        }
%>
```

Данный код можно размещать в документах *.asp, *.aspx.

Использование планов обмена в расширении конфигурации

При появлении расширений конфигурации планы обмена вошли в состав объектов, которые расширяют данные. Планы обмена наряду с другими объектами конфигурации можно заимствовать в расширение. В расширении есть возможность добавлять в заимствованные планы обмена реквизиты, табличные части и реквизиты табличных частей, формы, команды и макеты.

А также можно расширять состав заимствованного плана обмена как собственными объектами расширения, так и объектами из расширяемой конфигурации (которые ранее не входили в состав этого плана обмена).

Но сузить состав передаваемой планом обмена информации нельзя, то есть нельзя исключить объекты расширяемой конфигурации из состава плана обмена.

Кроме того, в расширении имеется возможность создать собственный план обмена. Однако такой план обмена не может участвовать в распределенной информационной базе.

Также следует помнить, что в нераспределенной информационной базе платформа не предоставляет инструментов для автоматической синхронизации состава расширений между узлами обмена данными. Эту синхронизацию необходимо выполнять отдельно.

Универсальный способ обмена данными

Чтобы продемонстрировать возможности использования планов обмена в расширении, доработаем немного наш пример, рассмотренный в разделе «Пример реализации универсального обмена».

Добавим в нашу конфигурацию расширение, реализующее создание и описание характеристик номенклатуры. Затем позаимствуем в расширение план обмена, с помощью которого в расширяемой конфигурации выполняется обмен данными универсальным образом. Включим в его состав новые объекты, добавленные расширением, а также объекты из расширяемой конфигурации, не включенные ранее в состав этого плана обмена. Кроме того, добавим в заимствованный план обмена новый реквизит и отобразим его в форме узла.

Чтобы реализовать поставленные задачи, загрузим в нашу конфигурацию (с диска, прилагающегося к книге) или создадим с нуля расширение для описания характеристик номенклатуры. Это расширение содержит специальным образом связанные между собой объекты: план видов характеристик Расш1_ВидыХарактеристик (для хранения видов характеристик номенклатуры), справочник Расш1_ЗначенияХарактеристик (для хранения дополнительных значений видов характеристик) и регистр сведений Расш1_ДополнительныеХарактеристики (для хранения значений характеристик номенклатуры).

На самом деле, какую функциональность реализует расширение, значения не имеет. Это просто один из примеров расширения, на котором мы хотим показать работу с планами обмена.

Предположим, что до этого с помощью плана обмена УдаленныеСклады не передавалась информация о ценах номенклатуры, то есть справочник ТипыЦен и регистр сведений ЦеныНоменклатуры не входили в состав плана обмена расширяемой конфигурации.

Теперь мы хотим расширить состав плана обмена и включить в него информацию о ценах, а также о дополнительных характеристиках номенклатуры.

Для этого позаимствуем в расширение план обмена УдаленныеСклады. Объекты расширяемой конфигурации, которые нужно включить в состав плана обмена, тоже надо заимствовать. Поэтому позаимствуем из расширяемой конфигурации справочник ТипыЦен и регистр сведений ЦеныНоменклатуры.

Установим признак модифицированности у свойства Состав заимствованного плана обмена, нажмем на ссылку Открыть и откроем его состав. Установим признак модифицированности у тех элементов состава плана обмена, о которых мы говорили (хранящих информацию о ценах и характеристиках номенклатуры), и включим соответствующие объекты в состав расширяющего плана обмена (рис. 3.35).

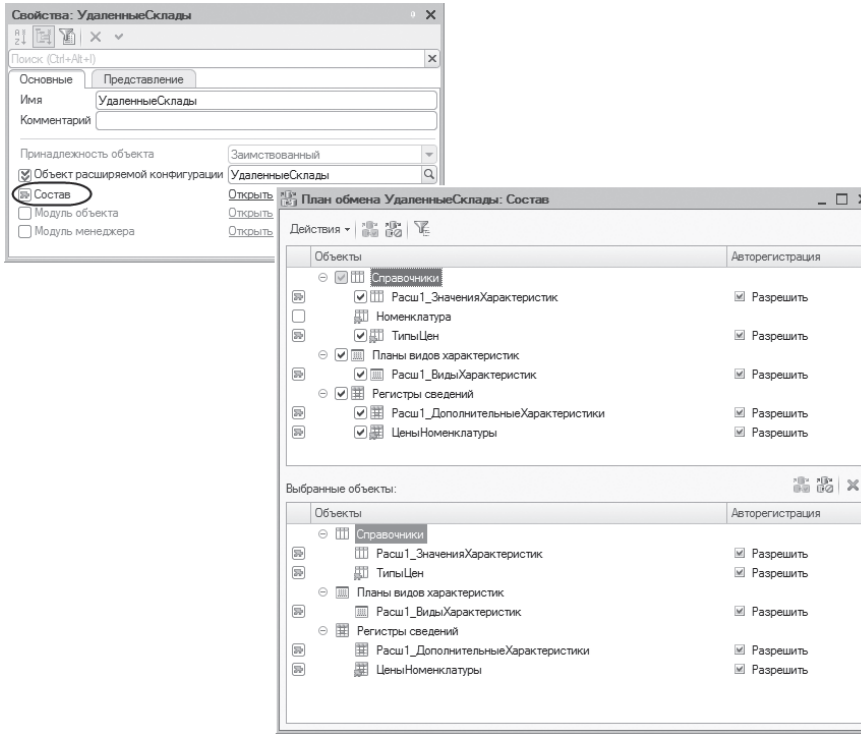


Рис. 3.35. Состав расширяющего плана обмена «УдаленныеСклады»

После этого в модули новых объектов (плана видов характеристик Расш1_ВидыХарактеристик и справочника Расш1_ЗначенияХарактеристик), участвующих в обмене, поместим процедуру ПриУстановкеНовогоКода() – листинг 3.103.

Листинг 3.103. Процедура «ПриУстановкеНовогоКода»

```
Процедура ПриУстановкеНовогоКода(СтандартнаяОбработка, Префикс)

    Префикс = Обмен.ПолучитьПрефиксНомера();

КонецПроцедуры
```

Как правило, планы обмена заимствуются не только для того, чтобы расширить их состав, но и затем, чтобы дополнить их функциональность. Например, нам нужно добавить новый реквизит плана обмена для хранения информации о способе обмена данными и отобразить его в форме узла.

Добавим в заимствованный план обмена реквизит Расш1_СпособОбмена. Чтобы отобразить его в форме узла, позаимствуем эту форму из расширяемой конфигурации. Откроем форму в расширении, выделим основной реквизит формы Объект и нажмем кнопку Добавить в расширение в командной панели окна реквизитов формы. После этого перетащим новый реквизит плана обмена в окно элементов формы (рис. 3.36).

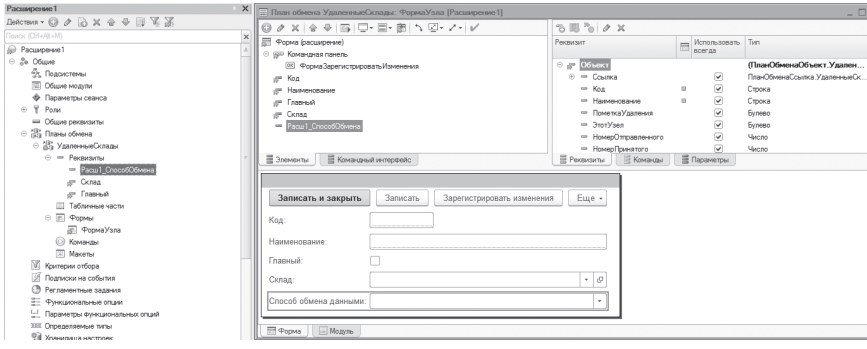


Рис. 3.36. Расширяющая форма узла плана обмена «УдаленныеСклады» в конфигураторе

Обновим конфигурацию базы данных, примем все изменения в окне реструктуризации информации и запустим «1С:Предприятие». Форма узла обмена примет следующий вид (рис. 3.37).

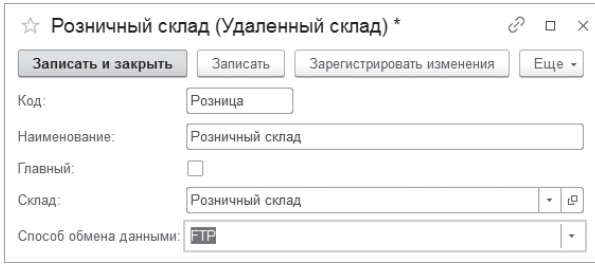


Рис. 3.37. Форма узла плана обмена «Удаленные склады» в режиме «1С:Предприятие»

Никакая дальнейшая обработка значения, установленного для нового реквизита, в нашем расширении не производится, поскольку это не входит в наши задачи.

Введем несколько видов характеристик номенклатуры в план видов характеристик Виды характеристик, значения видов характеристик – в справочник Значения характеристик и установим некоторые значения дополнительных характеристик номенклатуры в регистре сведений Дополнительные характеристики.

Сохраним в файл наше расширение, а также отдельно саму расширяемую конфигурацию и протестируем, как работает обмен данными.

В центральной базе в обработке Обмен с удаленными складами выберем узел обмена Розничный склад. Сначала удалим всю регистрацию изменений для этого узла обмена с помощью кнопки Удалить регистрацию изменений, а потом снова выполним полную регистрацию изменений по этому узлу с помощью кнопки Полная регистрация изменений. Затем нажмем кнопку Выгрузить данные и запишем все изменения, зарегистрированные в центральной базе для розничного склада, в файл выгрузки.

После этого создадим пустую базу розничного склада из выгруженной ранее расширяемой конфигурации. Затем в конфигураторе загрузим в нее ранее сохраненное расширение. Запустим «1С:Предприятие» и создадим в этой базе такой же список узлов плана обмена Удаленные склады с такими же кодами узлов, что и в центральной базе. Установим префикс номеров и каталог обмена в форме настроек информационной базы так же, как мы это делали раньше.

В обработке Обмен с удаленными складами выберем узел обмена Центральный офис, нажмем кнопку Загрузить данные и получим все изменения, переданные от центральной базы. Помимо ранее передаваемых данных тут будут данные о ценах и о характеристиках номенклатуры. Добавим новый вид характеристики (его код будет иметь префикс «РС») и значение этой характеристики для номенклатуры (рис. 3.38).

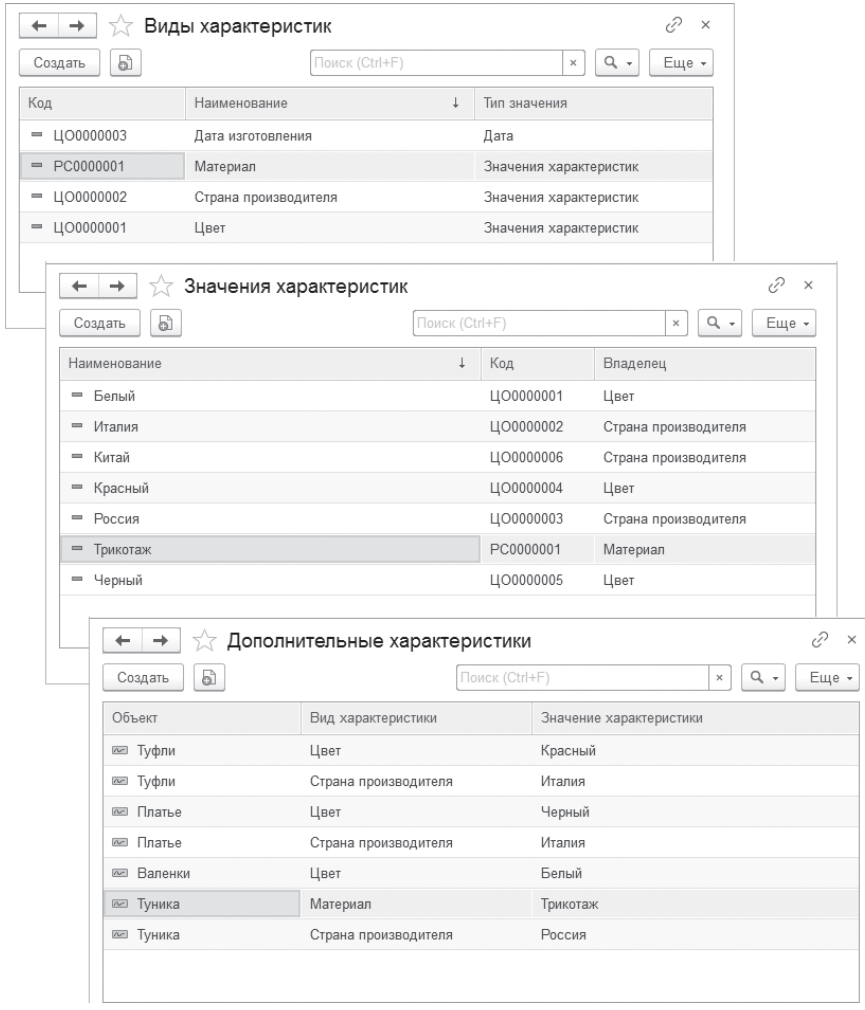


Рис. 3.38. Данные о характеристиках номенклатуры в узле обмена «Розничный склад»

Выгрузим эти изменения и прочитаем их в центральной базе. Все работает правильно. Все доработки, сделанные ранее в расширяемой конфигурации (стратегия распространения данных, разрешение коллизий, ручная регистрация данных и т. п.), также функционируют, поскольку в расширении мы их никак не изменяли.

Обмен данными в распределенной информационной базе

Расширение конфигурации можно использовать также и в распределенной информационной базе. В этом случае расширение будет передаваться между узлами распределенной информационной базы с помощью сообщений обмена, тем самым обеспечивая единый состав расширений в рамках распределенной системы.

Для указания того, что план обмена может передавать расширения, необходимо установить у этого плана обмена свойство Включать расширения конфигурации. Этот флажок можно установить только для планов обмена, которые участвуют в распределенной информационной базе.

В то же время каждое расширение, подключаемое к информационной базе, имеет свойство Используется в распределенной информационной базе. Установка данного свойства означает, что данное расширение будет мигрировать в подчиненные информационные базы.

Продемонстрируем вышесказанное на примере плана обмена Магазины, на основе которого выполняется обмен данными в распределенной информационной базе. У этого плана обмена установлен флажок Распределенная информационная база.

Чтобы наше расширение передавалось вместе с конфигурацией в подчиненные узлы, установим в свойствах расширения флажок Используется в распределенной информационной базе. А также в свойствах плана обмена Магазины установим флажок Включать расширения конфигурации (рис. 3.39).

Позаимствуем план обмена Магазины в расширение и дополним его состав объектами расширения, как было описано в предыдущем разделе (рис. 3.40).

Далее реализуем следующую задачу. В примере, демонстрирующем обмен данными в распределенной информационной базе (в разделе «Разрешение коллизий»), мы реализовали нестандартный вариант разрешения коллизий, при котором изменения подчиненного узла имели более высокий приоритет, чем главного. Это касалось только документов и их движений в регистрах. Обмен всеми остальными объектами происходил «по правилам».

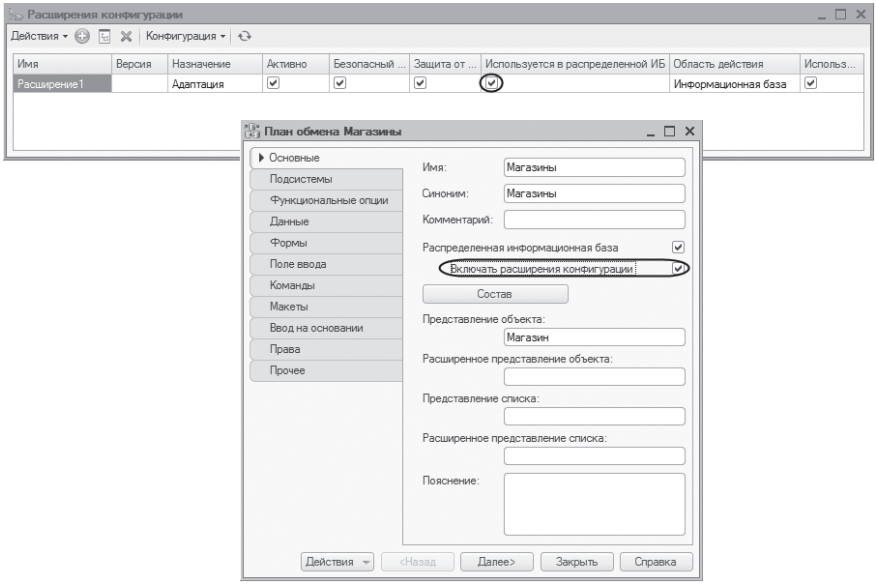


Рис. 3.39. Установка свойств расширения и плана обмена для использования в распределенной информационной базе

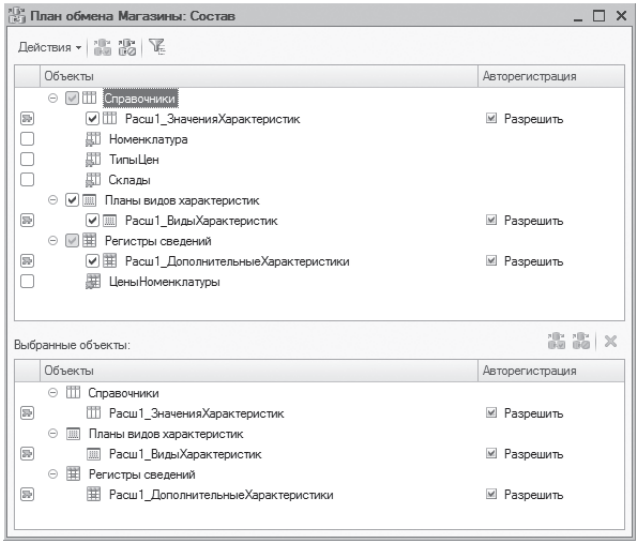


Рис. 3.40. Состав расширяющего плана обмена «Магазины»

Возможность принятия или отклонения изменений определялась в функции `ПринятьИзменения()`, помещенной в модуле плана обмена Магазины (листинг 3.104).

Листинг 3.104. Функция «ПринятьИзменения» в расширяемой конфигурации

```
Функция ПринятьИзменения(ЭлементДанных)

	Прием = Ложь;

	ТипДанных = ТипЗнч(ЭлементДанных);
	Если ТипЗнч(ЭлементДанных) = Тип("ДокументОбъект.ПриходнаяНакладная")
		Или ТипЗнч(ЭлементДанных) = Тип("ДокументОбъект.РасходнаяНакладная")
		Или ТипЗнч(ЭлементДанных) = Тип("РегистрНакопленияНаборЗаписей.УчетНоменклатуры") Тогда

		Прием = Истина;

	КонецЕсли;

	Возврат Прием;

КонецФункции
```

Теперь мы хотим, чтобы при обмене дополнительными характеристиками номенклатуры в случае возникновения коллизий также приоритет был бы у подчиненного узла. Для этого нам нужно создать расширяющий метод функции плана обмена и добавить туда нужную функциональность.

В расширяемом модуле плана обмена Магазины установим курсор на заголовок функции `ПринятьИзменения()`, вызовем команду контекстного меню Добавить в расширение и выберем тип перехвата Вместо. Шаблон расширяющей функции заполним следующим образом (листинг 3.105).

Листинг 3.105. Расширяющая функция «Расш1_ПринятьИзменения()»

```
&Вместо("ПринятьИзменения")
Функция Расш1_ПринятьИзменения(ЭлементДанных)

	Результат = ПродолжитьВызов(ЭлементДанных);

	Если ТипЗнч(ЭлементДанных) = Тип(
			"РегистрСведенийНаборЗаписей.Расш1_ДополнительныеХарактеристики") Тогда
		Результат = Истина;
	КонецЕсли;

	Возврат Результат;

КонецФункции
```

Сначала с помощью метода глобального контекста `ПродолжитьВызов()` производится вызов расширяемой функции. Это гарантирует, что вся прежняя функциональность разрешения коллизий при обмене данными будет сохранена.

В заключение надо не забыть снять флажок у свойства расширения Безопасный режим, так как расширение модулей не может выполняться в безопасном режиме.

Проверим, как это работает. Запустим «1С:Предприятие» и создадим начальный образ центральной базы для одного из подчиненных узлов плана обмена Магазины.

Затем откроем базу подчиненного узла в конфигураторе. Расширение также будет загружено в базу узла без возможности его изменения (рис. 3.41).

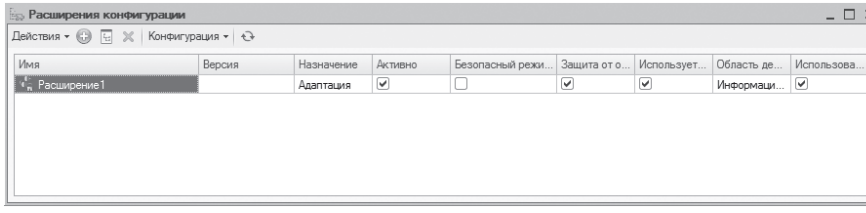


Рис. 3.41. Расширение, переданное в базу подчиненного узла

Протестируем обмен данными между центральным и подчиненным узлами плана обмена в обе стороны и убедимся, что наряду с остальными данными передается также и информация о характеристиках номенклатуры. Причем при возникновении коллизий изменения значений характеристик, установленные в подчиненном узле, имеют больший приоритет, так же как и изменения приходных и расходных накладных.

Глава 4.
Внешние компоненты

Технология создания внешних компонент разработана для решения специальных задач интеграции, в которых требуется тесное взаимодействие между системой «1С:Предприятие» и другими программами.

Процесс написания файла внешней компоненты описан в документации по созданию внешних компонент (https://its.1c.ru/db/metod8dev/content/3221/hdoc). В данном разделе мы не будем касаться их внутреннего устройства, рассмотрим только способы использования готовых внешних компонент в системе «1С:Предприятие».

Внешние компоненты могут быть созданы по двум технологиям:

- Native API − рекомендуемая технология;
- COM − старая технология, существовавшая в версиях «1С:Предприятие 8.1» и младше. Сейчас она поддерживается для совместимости со старыми компонентами.

Это позволяет создавать внешние компоненты:

- для ОС семейства Windows, ОС семейства Linux и macOS;
- веб-клиента, работающего в веб-браузерах Microsoft Internet Explorer, Mozilla Firefox, Google Chrome и Safari;
- мобильного приложения, работающего под управлением мобильных ОС iOS, Android и Windows.

В зависимости от контекста исполнения могут использоваться внешние компоненты, созданные только по «новой» технологии или по любой из этих технологий:

- толстый и тонкий клиенты могут использовать компоненты, созданные по обеим технологиям;
- веб-клиент может использовать любые компоненты Native API, а компоненты COM только в том случае, если веб-клиент работает в браузере под управлением ОС семейства Windows;
- на сервере и в мобильном приложении можно использовать только компоненты Native API.

Внешняя компонента представляет собой файл с расширением dll. Так как «1С:Предприятие» может работать на разных операционных системах с различной разрядностью, разработчикам внешних компонент рекомендуется создавать комплект из четырех файлов: для ОС Windows 32 разряда, ОС Windows 64 разряда, ОС Linux 32 разряда, ОС Linux 64 разряда. Эти файлы могут поставляться:

- в виде ZIP-архива (рекомендуемый способ для толстого клиента и сервера, а для тонкого клиента, веб-клиента и мобильного приложения это обязательный способ). Архив включает в себя четыре файла внешних компонент и файл-манифест, описывающий назначение каждой из компонент;
- в виде отдельных файлов.

Внешняя компонента (либо набор внешних компонент) может храниться:

- на диске (только в виде отдельных файлов);
- в макете «1С:Предприятия», содержащем двоичные данные (в виде отдельных файлов или ZIP-архива);
- в информационной базе в реквизите с типом `ХранилищеЗначения` (в виде отдельных файлов или ZIP-архива).

При работе с компонентами, написанными с использованием Native API, следует придерживаться следующей схемы работы:

- В интерфейсе реализуется команда, которая вызывает установку внешней компоненты на компьютер пользователя с помощью метода глобального контекста `УстановитьВнешнююКомпоненту()`. Данная операция обязательна для работы в тонком и веб-клиенте.
- Компонента подключается с помощью метода `ПодключитьВнешнююКомпоненту()`. При этом следует помнить, что попытка подключить неустановленную компоненту приведет к ошибке.
- Подключенная компонента используется согласно предоставленному программному интерфейсу.

СОВЕТ

Не следует объединять в один программный код установку и подключение внешней компоненты. Установка считается однократным событием, и повторная установка вызовет интерактивное исключение.

Если используется компонента, упакованная в zip-архив, то для указания параметров внешней компоненты (имя устанавливаемого файла, тип, архитектура, используемый веб-браузер) служит специальный файл-манифест, формируемый разработчиком компоненты.

Получаемые из конфигурации или информационной базы внешние компоненты сохраняются после получения и используются при последующем подключении без повторного получения (компонента, используемая на сервере, сохраняется только на время работы рабочего процесса сервера).

ПОДРОБНЕЕ

Более подробно про работу с внешними компонентами рассказано в документации «1С:Предприятия» в разделе «Глава 38. Внешние компоненты».

Ниже мы рассмотрим разные ситуации, в которых требуется подключать внешнюю компоненту.

Подключение внешней компоненты в тонком клиенте или в веб-клиенте (на примере Native API компоненты)

Для подключения внешней компоненты в тонком клиенте или веб-клиенте используются методы глобального контекста `ПодключитьВнешнююКомпоненту()` и `УстановитьВнешнююКомпоненту()` (причем метод `УстановитьВнешнююКомпоненту()` используется только тогда, когда внешняя компонента расположена не на диске).

Разберем примеры подключения внешней компоненты в зависимости от вариантов ее хранения.

Подключение внешней компоненты из файла на диске (отдельные файлы)

Данный вариант подключения возможен только в тонком клиенте (листинг 4.1).

Листинг 4.1. Пример подключения внешней компоненты

```
СисИнфо = Новый СистемнаяИнформация;
Если СисИнфо.ТипПлатформы = ТипПлатформы.Windows_x86 Тогда
        ПодключитьВнешнююКомпоненту("C:\AddInCPP.dll", "ПроизвольноеИмя", ТипВнешнейКомпоненты.Native);
ИначеЕсли СисИнфо.ТипПлатформы = ТипПлатформы.Windows_x86_64 Тогда
        ПодключитьВнешнююКомпоненту("AddInCPP64.dll", "ПроизвольноеИмя", ТипВнешнейКомпоненты.Native);
ИначеЕсли СисИнфо.ТипПлатформы = ТипПлатформы.Linux_x86 Тогда
        ПодключитьВнешнююКомпоненту("libAddInCPP.so", "ПроизвольноеИмя", ТипВнешнейКомпоненты.Native);
Иначе
    ПодключитьВнешнююКомпоненту("libAddInCPP64.so", "ПроизвольноеИмя", ТипВнешнейКомпоненты.Native);
КонецЕсли;
ОбъектКомпоненты = Новый ("AddIn.ПроизвольноеИмя.ComponentExtention");
ОбъектКомпоненты.ИмяСвойства = ...;
```

В рассмотренном примере (и далее) используются следующие входные данные:

- `ПроизвольноеИмя` – произвольное имя, задаваемое разработчиком «1С:Предприятия».
- `ComponentExtention` – расширение, реализуемое компонентой. Это расширение должно быть описано в документации к этой внешней компоненте.

Подключение внешней компоненты из макета (ZIP-архив)

Листинг 4.2. Пример подключения внешней компоненты

```
УстановитьВнешнююКомпоненту("Обработка.Компонента.Макет.AddIn");
ПодключитьВнешнююКомпоненту("Обработка.Компонента.Макет.AddIn", "ПроизвольноеИмя");
ОбъектКомпоненты = Новый ("AddIn.ПроизвольноеИмя.ComponentExtention");
ОбъектКомпоненты.ИмяСвойства = ...;
```

Подключение внешней компоненты из базы данных (ZIP-архив)

Листинг 4.3. Пример подключения внешней компоненты

```
Ссылка = ПолучитьНавигационнуюССылку("Справочники.ВнешниеКомпоненты.НашаКомпонента"
                                     , "КомпонентаВАрхиве");
УстановитьВнешнююКомпоненту(Ссылка);
ПодключитьВнешнююКомпоненту(Ссылка, "ПроизвольноеИмя");
ОбъектКомпоненты = Новый ("AddIn.ПроизвольноеИмя.ComponentExtention");
ОбъектКомпоненты.ИмяСвойства = ...;
```

Подключение внешней компоненты в толстом клиенте или на сервере (на примере Native API компоненты)

Для подключения внешней компоненты на сервере или в толстом клиенте используется метод глобального контекста `ПодключитьВнешнююКомпоненту()`.

Разберем примеры подключения внешней компоненты в зависимости от вариантов ее хранения.

Подключение внешней компоненты из файла на диске (отдельные файлы)

Листинг 4.4. Пример подключения внешней компоненты

```
СисИнфо = Новый СистемнаяИнформация;
Если СисИнфо.ТипПлатформы = ТипПлатформы.Windows_x86 Тогда
        ПодключитьВнешнююКомпоненту("C:\AddInCPP.dll", "ПроизвольноеИмя", ТипВнешнейКомпоненты.Native);
ИначеЕсли СисИнфо.ТипПлатформы = ТипПлатформы.Windows_x86_64 Тогда
        ...
КонецЕсли;
ОбъектКомпоненты = Новый ("AddIn.ПроизвольноеИмя.ComponentExtention");
ОбъектКомпоненты.ИмяСвойства = ...;
```

Подключение внешней компоненты из макета (ZIP-архив)

Листинг 4.5. Пример подключения внешней компоненты

```
ПодключитьВнешнююКомпоненту ("Обработка.Компонента.Макет.AddIn", "ПроизвольноеИмя");
ОбъектКомпоненты = Новый ("AddIn.ПроизвольноеИмя.ComponentExtention");
ОбъектКомпоненты.ИмяСвойства = ...;
```

Подключение внешней компоненты из макета (отдельные файлы)

Листинг 4.6. Пример подключения внешней компоненты

```
СисИнфо = Новый СистемнаяИнформация;
Если СисИнфо.ТипПлатформы = ТипПлатформы.Windows_x86 Тогда
    ПодключитьВнешнююКомпоненту("Обработка.Компонента.Макет.AddInWindows32"
                                , "ПроизвольноеИмя", ТипВнешнейКомпоненты.Native);
ИначеЕсли СисИнфо.ТипПлатформы = ТипПлатформы.Windows_x86_64 Тогда
    ПодключитьВнешнююКомпоненту("Обработка.Компонента.Макет.AddInWindows64"
                                , "ПроизвольноеИмя", ТипВнешнейКомпоненты.Native);
ИначеЕсли СисИнфо.ТипПлатформы = ТипПлатформы.Linux_x86 Тогда
```

```
    ПодключитьВнешнююКомпоненту("Обработка.Компонента.Макет.AddInLinux32"
                                 , "ПроизвольноеИмя", ТипВнешнейКомпоненты.Native);
Иначе
    ПодключитьВнешнююКомпоненту("Обработка.Компонента.Макет.AddInLinux64"
                                 , "ПроизвольноеИмя", ТипВнешнейКомпоненты.Native);
КонецЕсли;
ОбъектКомпоненты = Новый ("AddIn.ПроизвольноеИмя.ComponentExtention");
ОбъектКомпоненты.ИмяСвойства = ...;
```

Подключение внешней компоненты из базы данных (ZIP-архив)

Листинг 4.7. Пример подключения внешней компоненты

```
Ссылка = ПолучитьНавигационнуюССылку("Справочники.ВнешниеКомпоненты.НашаКомпонента"
                                     , "КомпонентаВАрхиве");
ПодключитьВнешнююКомпоненту (Ссылка, "ПроизвольноеИмя");
ОбъектКомпоненты = Новый ("AddIn.ПроизвольноеИмя.ComponentExtention");
ОбъектКомпоненты.ИмяСвойства = ...;
```

Подключение внешней компоненты из базы данных (отдельные файлы)

Листинг 4.8. Пример подключения внешней компоненты

```
Перем Ссылка;
СисИнфо = Новый СистемнаяИнформация;
Если СисИнфо.ТипПлатформы = ТипПлатформы.Windows_x86 Тогда
    Ссылка = ПолучитьНавигационнуюССылку(Справочники.ВнешниеКомпоненты.НайтиПоКоду("000000001"),
                                         "КомпонентаWindows32");
ИначеЕсли СисИнфо.ТипПлатформы = ТипПлатформы.Windows_x86_64 Тогда
    Ссылка = ПолучитьНавигационнуюССылку(Справочники.ВнешниеКомпоненты.НайтиПоКоду("000000001"),
                                         "КомпонентаWindows64");
ИначеЕсли СисИнфо.ТипПлатформы = ТипПлатформы.Linux_x86 Тогда
    Ссылка = ПолучитьНавигационнуюССылку(Справочники.ВнешниеКомпоненты.НайтиПоКоду("000000001"),
                                         "КомпонентаLinux32");
Иначе
    Ссылка = ПолучитьНавигационнуюССылку(Справочники.ВнешниеКомпоненты.НайтиПоКоду("000000001"),
                                         "КомпонентаLinux64");
КонецЕсли;
ПодключитьВнешнююКомпоненту(Ссылка, "ПроизвольноеИмя", ТипВнешнейКомпоненты.Native);
ОбъектКомпоненты = Новый("AddIn.ПроизвольноеИмя.ComponentExtension");
ОбъектКомпоненты.ИмяСвойства = ...;
```

Глава 5. Взаимодействие с приложением системы «1С:Предприятие»

Automation

Технология Automation предназначена для программного использования объектов, чаще всего не имеющих визуального представления (исключения составляют, например, объекты Microsoft Office Word.Application и Excel. Application, при этом они отображаются в отдельном окне, отличном от окна «1С:Предприятия»).

Обычно задачи, которые решаются с помощью этой технологии, – это запись и чтение данных в/из специфических (для других приложений) форматов, запись и чтение в какие-либо базы данных.

«1С:Предприятие» может выступать как в роли Automation Server, так и в роли Automation Client.

Automation Server

Основное назначение Automation-сервера «1С:Предприятия» − управление приложением системы «1С:Предприятие» из других приложений и выполнение действий, аналогичных интерактивным действиям.

Automation-сервер «1С:Предприятия» предоставляет доступ ко всем свойствам и методам своего глобального контекста, имеет дополнительные

свойства и методы для выполнения действий, специфичных для работы в режиме Automation.

Для запуска системы «1С:Предприятие» в качестве Automation-сервера из внешнего приложения выполняется следующая последовательность действий:

- создается COM-объект с идентификатором V83.Application (толстый клиент) или V83c.Application (тонкий клиент);
- выполняется инициализация системы «1С:Предприятие» методом `Connect()`;
- вызываются свойства и методы системы «1С:Предприятие» как Automation-сервера.

Поставим задачу загрузить данные из листа программы Microsoft Excel в базу данных «1С:Предприятия».

В листе данные расположены следующим образом (рис. 5.1).

	A	B	C	D	E	F	G	H
1	Клиент	Сорокина И.В.						
2	Номер	777						
3	Дата	26.12.2019						
4								
5	**Номер**	**Наименование**			**Количество**	**Цена**	**Сумма**	
6	1	Тостер			10	2000	20000	
7	2	Кофемашина			5	5000	25000	
8	#							
9								
10								

Рис. 5.1. Состав листа Excel

Признаком конца табличной части является наличие в колонке «Номер» символа «#».

Пример кода макроса на Visual Basic, с помощью которого выполняется загрузка данных, приведен в листинге 5.1.

Листинг 5.1. Пример чтения данных из листа Excel в толстом клиенте

```
Sub OLE()
        Dim trade As Object
        Dim Элемент As Object

        Set trade = CreateObject("V83.Application")
        trade.Connect("File=""c:\InfoBases\Trade"";Usr=""Director"";")

        Rem создаем документ
        Set Документ = trade.Документы.РасходнаяНакладная.СоздатьДокумент()

        Rem получаем данные из листа
        Set Клиент = trade.Справочники.Клиенты.НайтиПоНаименованию(Application.Cells(1, 2).Value)
```

```
    НомерДокумента = Application.Cells(2, 2).Value
    Дата = Application.Cells(3, 2).Value

    Rem записываем полученные данные в документ
    Документ.Клиент = Клиент
    Документ.Дата = Дата
    Документ.Номер = НомерДокумента

    Номер = 6   'Первая строка табличной части
    НомерСтроки = Application.Cells(Номер, 1).Value

    Rem в последней строке табличной части будет символ #
    While НомерСтроки <> "#"

        Rem получаем данные из листа
        Set Номенклатура = trade.Справочники.Товары.НайтиПоНаименованию(
                            Application.Cells(Номер, 2).Value)
        Количество = Application.Cells(Номер, 5).Value
        Цена = Application.Cells(Номер, 6).Value
        Сумма = Application.Cells(Номер, 7).Value

        Rem записываем полученные данные в строку табличной части
        Set Строка = Документ.Товары.Добавить()
        Строка.Товар = Номенклатура
        Строка.Количество = Количество
        Строка.Цена = Цена
        Строка.Сумма = Сумма
        Номер = Номер + 1
        НомерСтроки = Application.Cells(Номер, 1).Value
    Wend
    Документ.Записать

    Документ.ПолучитьФорму.Открыть
End Sub
```

В данном примере создается COM-объект с идентификатором V83.Application, то есть запускается и инициализируется конфигурация «1С:Предприятия» в режиме толстого клиента с базой данных в каталоге c:\InfoBases\Trade. Далее создается экземпляр документа РасходнаяНакладная. Из листа извлекаются данные документа и записываются в объект «1С:Предприятия». В конце алгоритма открывается форма вновь созданного документа.

В результате выполнения макроса из листа Microsoft Excel в информационной базе «1С:Предприятия» создается расходная накладная, документ заполняется данными и открывается его форма (рис. 5.2).

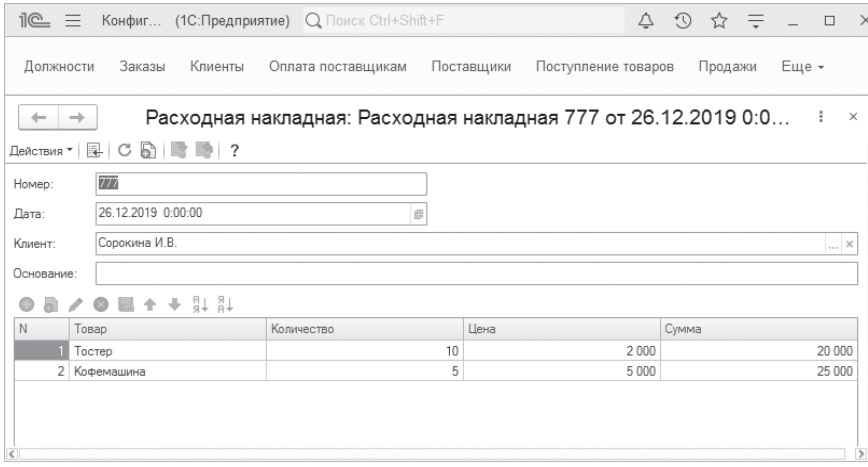


Рис. 5.2. Расходная накладная, заполненная из листа Microsoft Excel

Аналогичные действия можно выполнять и в режиме тонкого клиента, когда создается объект V83c.Application. Однако в тонком клиенте недоступны методы прикладных объектов, невозможно обратиться к менеджерам справочников, документов и т. п. Поэтому новый документ создается с помощью метода `ПолучитьФорму()`, обращение к его реквизитам происходит через основной реквизит полученной формы, а поиск ссылки на справочник выполняется в функции `ПолучитьСсылкуНаСправочник()` общего неглобального модуля РаботаСДанными (листинг 5.2).

Листинг 5.2. Пример чтения данных из листа Excel в тонком клиенте

```
Sub OLE_V83c()
    Dim trade As Object
    Dim Элемент As Object

    Set trade = CreateObject("V83c.Application")
    trade.Connect("File=""c:\InfoBases\Trade"";Usr="""";")

    Rem создаем документ
    Set ФормаДокумента = trade.ПолучитьФорму("Документ.РасходнаяНакладная.ФормаОбъекта")

    Rem получаем данные из листа
    Set Клиент = trade.РаботаСДанными.ПолучитьСсылкуНаСправочник("Клиенты", Application.Cells(1, 2).Value)
    НомерДокумента = Application.Cells(2, 2).Value
    Дата = Application.Cells(3, 2).Value

    Rem записываем полученные данные в документ
    ФормаДокумента.Объект.Клиент = Клиент
    ФормаДокумента.Объект.Дата = Дата
```

```
    ФормаДокумента.Объект.Номер = НомерДокумента

    Номер = 6  'Первая строка табличной части
    НомерСтроки = Application.Cells(Номер, 1).Value

    Rem в последней строке табличной части будет символ #
    While НомерСтроки <> "#"

        Rem получаем данные из листа
            Set Номенклатура = trade.РаботаСДанными.ПолучитьСсылкуНаСправочник(
                                "Товары", Application.Cells(Номер, 2).Value)
        Количество = Application.Cells(Номер, 5).Value
        Цена = Application.Cells(Номер, 6).Value
        Сумма = Application.Cells(Номер, 7).Value

        Rem записываем полученные данные в строку табличной части
        Set Строка = ФормаДокумента.Объект.Товары.Добавить()
        Строка.Товар = Номенклатура
        Строка.Количество = Количество
        Строка.Цена = Цена
        Строка.Сумма = Сумма
        Номер = Номер + 1
        НомерСтроки = Application.Cells(Номер, 1).Value
    Wend

    ФормаДокумента.Открыть

End Sub
```

В конце алгоритма открывается форма вновь созданного документа, и данные в документ могут быть записаны интерактивно из формы.

Результат выполнения этого макроса аналогичен показанному выше на рис. 5.2.

Объект Automation-сервер «1С:Предприятия» в качестве своих свойств может иметь:

- системные константы;
- значения заданных в конфигураторе объектов, доступ к которым осуществляется с помощью менеджеров (например, константы, перечисления, справочники, документы, журналы документов, отчеты, обработки, планы видов характеристик, планы счетов, планы видов расчета, регистры);
- переменные, объявленные в модуле приложения с ключевым словом `Экспорт`.

Automation-сервер «1С:Предприятия» в качестве своих методов может иметь:

- системные процедуры и функции;
- процедуры и функции модуля приложения и общих модулей, объявленные с ключевым словом `Экспорт`;
- два дополнительных метода: `Connect()` и `NewObject()`.

Как и многие современные программные продукты, «1С:Предприятие» может выступать в роли клиентов Automation. Поэтому имеется возможность из системы «1С:Предприятие» обращаться к другой копии «1С:Предприятия» (например, к другой конфигурации) для обмена данными. Здесь также можно запускать «1С:Предприятие» в толстом клиенте (листинг 5.3) и в тонком клиенте (листинг 5.4).

ПОДРОБНЕЕ

Подробнее познакомиться с использованием системы «1С:Предприятие» в качестве Automation Client (толстый и тонкий клиент) можно в демонстрационной конфигурации «Интеграция с 1С», которая прилагается к книге.

Листинг 5.3. Пример использования системы «1С:Предприятие» в качестве Automation Client (толстый клиент)

```
Процедура AutomationТолстыйКлиент(Команда)

	Путь = "c:\InfoBases\Trade";
	Пароль = "";
	Пользователь = "";

	V83 = Новый COMОбъект("V83.Application");

	Попытка
		Открытие = V83.Connect("File=" + Путь + ";Usr=" + Пользователь + ";Pwd=" + Пароль + ";");
	Исключение
		ПоказатьПредупреждение(, "База данных не открыта!!!");
		Возврат;
	КонецПопытки;

	МенеджерНоменклатуры = V83.Справочники.Товары;
	НовыйЭлемент = МенеджерНоменклатуры.СоздатьЭлемент();
	НовыйЭлемент.Наименование = "Созданный";
	НовыйЭлемент.Записать();
	Фрм = НовыйЭлемент.ПолучитьФорму();
	Фрм.Открыть();

КонецПроцедуры
```

Листинг 5.4. Пример использования системы «1С:Предприятие» в качестве Automation Client (тонкий клиент)

```
Процедура AutomationТонкийКлиент(Команда)

	Путь = "c:\InfoBases\Trade";
	Пароль = "";
	Пользователь = "";
```

```
    V83 = Новый COMObject("V83c.Application");

    Попытка
        Открытие = V83.Connect("File=" + Путь + ";Usr=" + Пользователь + ";Pwd=" + Пароль + ";");
    Исключение
        ПоказатьПредупреждение(, "База данных не открыта!!!");
        Возврат;
    КонецПопытки;

    //тонкий клиент - создание нового элемента при открытии формы.
    Фрм = V83.ПолучитьФорму("Справочник.Товары.ФормаОбъекта");
    Фрм.Объект.Наименование = "Добавленный";
    Фрм.Открыть();

КонецПроцедуры
```

Automation Client

Основное назначение Automation-клиента «1С:Предприятия» − управление другими приложениями из системы «1С:Предприятие» и выполнение действий, аналогичных интерактивным действиям.

Можно сказать, что технология рекомендуется к применению, когда не требуется предоставлять пользователю интерфейс, выходящий за рамки возможностей «1С:Предприятия», но в то же время требуется взаимодействие с «внешней функциональностью».

В общем случае применение этой технологии состоит из двух частей: создание объекта по его идентификатору и последующее использование объекта. Создание объекта выполняется оператором `Новый COMОбъект(ИдентификаторОбъекта)`.

Использование созданного объекта ничем не отличается от использования остальных объектов «1С:Предприятия».

Дополнительные возможности предоставляет функция `ПолучитьCOMОбъект()`. С ее помощью можно создать Automation-объект из файла или подключиться к уже существующему в операционной системе экземпляру Automation-объекта.

ПОДРОБНЕЕ

> Подробнее познакомиться с примером работы с книгой программы Microsoft Excel можно в демонстрационной конфигурации «Интеграция с 1С», которая прилагается к книге.

Пример работы с книгой программы Microsoft Excel приведен в листинге 5.5.

Листинг 5.5. Пример работы с книгой Excel

```
&НаКлиенте
Процедура РаботаСКнигойExcel(Команда)

	Путь= "c:\temp\";
	Сообщение = Новый СообщениеПользователю();

	Документ = ПолучитьCOMОбъект(Путь + "Integration.xls");

	НомерЛиста = 2;
	Клиент = Документ.Sheets(НомерЛиста).Cells(1, 2).Value;
	Сообщение.Текст = Клиент;
	Сообщение.Сообщить();

	// Обработать дальнейшие данные шапки документа.

	НомерСтроки=2;
	Номер = Документ.Sheets(НомерЛиста).Cells(НомерСтроки, 1).Value;

	НомерСтроки=6;
	Номер = Документ.Sheets(НомерЛиста).Cells(НомерСтроки, 1).Value;

	Пока СокрЛП(Номер) <> "#" Цикл
		Номенклатура = Документ.Sheets(НомерЛиста).Cells(НомерСтроки, 2).Value;
		Цена = Документ.Sheets(НомерЛиста).Cells(НомерСтроки, 6).Value;

		Сообщение.Текст = Номенклатура;
		Сообщение.Сообщить();
		Сообщение.Текст = Цена;
		Сообщение.Сообщить();

		// Обработать дальнейшие данные табличной части документа.
		НомерСтроки = НомерСтроки + 1;
		Номер = Документ.Sheets(НомерЛиста).Cells(НомерСтроки, 1).Value;
	КонецЦикла;

	// Завершить работу.
	Документ.Application.Quit();

КонецПроцедуры
```

Следует отметить, что для завершения работы с объектами Microsoft Office, имеющими метод `Quit()`, его желательно вызывать в явном виде.

Внешнее соединение

Основная задача, решаемая с помощью внешнего соединения, – обеспечение надежного и быстрого программного доступа к данным системы «1С:Предприятие» из внешних приложений. В общем и целом работа с системой «1С:Предприятие» через внешнее соединение подобна работе с системой «1С:Предприятие» в режиме Automation-сервера. Основные отличия заключаются в следующем:

- в случае Automation-сервера запускается полноценное приложение «1С:Предприятия», а в случае внешнего соединения запускается относительно небольшой внутрипроцессный COM-сервер;
- при работе через внешнее соединение недоступны функциональные возможности, так или иначе связанные с организацией пользовательского интерфейса системы «1С:Предприятие»;
- при работе через внешнее соединение не используется модуль приложения конфигурации «1С:Предприятия». Его роль при работе с внешним соединением играет модуль внешнего соединения.

При использовании внешнего соединения для доступа к данным системы «1С:Предприятие» имеются следующие преимущества по сравнению с использованием Automation-сервера:

- более быстрая установка соединения, так как не требуется создания отдельного процесса операционной системы, а все действия производятся в рамках вызывающего процесса;
- более быстрое обращение к свойствам и методам объектов системы «1С:Предприятие», так как для организации обращения не требуется организации межпроцессной коммуникации;
- меньший расход ресурсов операционной системы.

Для организации доступа к данным системы «1С:Предприятие» через внешнее соединение выполняется следующая последовательность действий:

- создается менеджер COM-соединений, с помощью которого производится установка соединения;
- через полученный объект внешнего соединения производится обращение к допустимым методам, свойствам и объектам базы данных, с которой установлено соединение.

Поставим задачу загрузить данные из листа программы Microsoft Excel в базу данных «1С:Предприятия» (посредством механизма COM).

В листе данные расположены так же, как и в предыдущем примере (см. рис. 5.1). Признаком конца табличной части является наличие в колонке «Номер» символа «#».

Пример кода приведен в листинге 5.6.

Листинг 5.6. Пример чтения данных из листа Excel

```
Sub COM()
	Dim trade As Object
	Dim Элемент As Object

	Set obj = CreateObject("V83.ComConnector")
	Set trade = obj.Connect("File=""c:\InfoBases\Trade"";Usr=""Director"";")

	Rem создаем документ
	Set Документ = trade.Документы.РасходнаяНакладная.СоздатьДокумент()

	Rem получаем данные из листа
	Set Клиент = trade.Справочники.Клиенты.НайтиПоНаименованию(Application.Cells(1, 2).Value)
	НомерДокумента = Application.Cells(2, 2).Value
	Дата = Application.Cells(3, 2).Value

	Rem записываем полученные данные в документ
	Документ.Клиент = Клиент
	Документ.Дата = Дата
	Документ.Номер = НомерДокумента

	Номер = 6   'Первая строка табличной части
	НомерСтроки = Application.Cells(Номер, 1).Value

	Rem в последней строке табличной части будет символ #
	While НомерСтроки <> "#"

		Rem получаем данные из листа
		Set Номенклатура = trade.Справочники.Товары.НайтиПоНаименованию(
					Application.Cells(Номер, 2).Value)
		Количество = Application.Cells(Номер, 5).Value
		Цена = Application.Cells(Номер, 6).Value
		Сумма = Application.Cells(Номер, 7).Value

		Rem записываем полученные данные в строку табличной части
		Set Строка = Документ.Товары.Добавить()
		Строка.Товар = Номенклатура
		Строка.Количество = Количество
		Строка.Цена = Цена
		Строка.Сумма = Сумма
		Номер = Номер + 1
		НомерСтроки = Application.Cells(Номер, 1).Value
	Wend
	Документ.Записать

End Sub
```

В данном примере запускается и инициализируется конфигурация «1С:Предприятия» с базой данных в каталоге c:\InfoBases\Trade. Далее создается экземпляр документа РасходнаяНакладная. Из листа извлекаются данные документа и записываются в объект «1С:Предприятия». В конце алгоритма документ сохраняется.

Обязанности модуля приложения при работе через внешнее соединение выполняет модуль внешнего соединения. Данный модуль может иметь процедуры-обработчики событий, в которых могут быть размещены действия, выполняемые при инициализации и завершении соединения соответственно.

Процедуры, функции и глобальные переменные, определенные в модуле внешнего соединения с ключевым словом `Экспорт`, становятся, как и в случае модуля приложения, частью глобального контекста.

Внешнее соединение с информационной базой «1С:Предприятия» предоставляет полный доступ к глобальному контексту и в качестве своих свойств может иметь:

- системные константы;
- значения заданных в конфигураторе объектов, доступ к которым осуществляется с помощью менеджеров (например, константы, перечисления, справочники, документы, журналы документов, отчеты, обработки, планы видов характеристик, планы счетов, планы видов расчета, регистры);
- переменные, объявленные в модуле внешнего соединения с ключевым словом `Экспорт`.

В качестве своих методов внешнее соединение может иметь:

- системные процедуры и функции;
- процедуры и функции модуля внешнего соединения и общих модулей, объявленные с ключевым словом `Экспорт`.

Открытые внешние соединения могут быть сохранены в пуле для их повторного использования, что позволяет экономить ресурсы и ускорять работу пользователей, которые ранее уже подключались к данной информационной базе. Для управления пулом используются свойства менеджера COM-соединений: `MaxConnections`, `PoolCapacity` и `PoolTimeout`. После создания менеджера COM-соединений используются стандартные значения этих параметров – 0. Поэтому пул хотя и есть, но не используется. Чтобы «1С:Предприятие» начало использовать пул, нужно установить значения, отличные от 0.

Имеется возможность из системы «1С:Предприятие» обращаться к другой копии «1С:Предприятия» (например, к другой конфигурации) для обмена данными (листинг 5.7).

ПОДРОБНЕЕ

Подробнее познакомиться с примером работы с информационной базой «1С:Предприятия» через внешнее соединение можно в демонстрационной конфигурации «Интеграция с 1С», которая прилагается к книге.

Листинг 5.7. Пример обращения к другой информационной базе «1С:Предприятия»

```
Путь = "c:\InfoBases\Trade";
Пароль = "";
Пользователь = "";

V8 = Новый COMОбъект("V83.ComConnector");

Попытка
        Открытие = V8.Connect("File=" + Путь + ";Usr=" + Пользователь + ";Pwd=" + Пароль + ";");
Исключение
        Предупреждение("База данных не открыта!!!");
        Возврат;
КонецПопытки;

МенеджерНоменклатуры = Открытие.Справочники.Товары;
НовыйЭлемент = МенеджерНоменклатуры.СоздатьЭлемент();
НовыйЭлемент.Наименование = "Новый товар";
НовыйЭлемент.Записать();
```

Встраивание веб-клиента «1С:Предприятия» в сторонний сайт

В процессе реализации прикладных систем могут возникать задачи интеграции веб-сайта с прикладным решением, доступ к которому осуществляется с использованием веб-клиента. Для этого платформа «1С:Предприятие» предоставляет возможность открытия веб-клиента внутри элемента `<iframe>` html-страницы веб-сайта.

ПОДРОБНЕЕ

Более подробно про встраивание веб-клиента во внешний сайт рассказано в документации «1С:Предприятия» в разделе «Глава 18. Интеграция веб-клиента и сайта».

Общая информация

Для реализации такой задачи должен быть выполнен ряд следующих действий:

- Прикладное решение должно быть опубликовано на веб-сервере. Адрес публикации необходимо запомнить для вставки в веб-страницу.

- Выделение на странице сайта области, в которой будет исполняться веб-клиент.
- Реализация некоторого интерфейса (на встроенном языке) на стороне прикладного решения, который будет отвечать за обмен данными с веб-сайтом, в который интегрируется прикладное решение. Сюда же можно отнести установку необходимого режима основного окна приложения и предоставление пользователю возможность использовать нужный режим основного окна клиентского приложения с помощью прав доступа.
- Реализация некоторого интерфейса (на языке JavaScript) на стороне сайта, в который интегрируется прикладное решение, с целью предоставления веб-клиенту необходимой информации о режиме работы и обмена данными с прикладным решением.

Для осуществления взаимодействия с сайтом у конфигурации существует свойство глобального контекста `ОкноВнешнегоСайта`. С помощью этого свойства прикладное решение получает доступ к менеджеру окна внешнего веб-сайта (в том случае, если прикладное решение интегрировано в веб-сайт). Используя свойства и методы этого менеджера, можно выполнить несколько действий:

- Проверить, что веб-клиент работает внутри какого-либо веб-сайта. Это можно выполнить с помощью свойства `Доступно` менеджера `ОкноВнешнегоСайта`. В дальнейшем рекомендуется все действия с менеджером окна внешнего веб-сайта выполнять после проверки того, что данное свойство установлено в значение `Истина`.
- Получить сообщения от внешнего (относительно веб-клиента) веб-сайта, предварительно зарегистрировав метод обработчика сообщений от веб-сайта с помощью метода `ПодключитьОбработчикСообщений()` менеджера `ОкноВнешнегоСайта`. Также можно отключить обработку сообщений внешнего веб-сайта, если такая обработка более не требуется. Отключение обработки сообщений выполняется с помощью метода `ОтключитьОбработчикСообщений()` менеджера `ОкноВнешнегоСайта`.
- Отправить сообщения внешнему веб-сайту с помощью метода `ОтправитьСообщение()` менеджера `ОкноВнешнегоСайта`.

Нужно иметь в виду, что получать и отправлять можно только текстовые данные.

На стороне сайта, в который будет встраиваться веб-клиент, необходимо выполнить следующие действия:

- Указать место, где будет отображаться интерфейс веб-клиента. Это можно сделать с помощью элемента . При загрузке веб-клиент автоматически добавит для выбранного элемента подчиненный элемент <iframe>, который будет содержать всю необходимую информацию.
- Указать, откуда загружать код JavaScript-интерфейса веб-клиента (Embedded WebClient API). После загрузки этого интерфейса в веб-браузере становится доступным объект WebClient1CE. Для загрузки JavaScript-интерфейса в текст сайта необходимо включить следующий код: <script src=»%АдресВебСайта%/%ИмяИБ%/scripts/webclient1ce.js»></script>. При этом если веб-клиент опубликован по адресу http://mysite.org/dbName, то %АдресВебСайта% заменяется на http://mysite.org, а %ИмяИБ% заменяется на dbName.
- Реализовать интерфейс взаимодействия с прикладным решением «1С:Предприятия».

Пример реализации

В этом разделе мы рассмотрим простейший пример интеграции веб-клиента и внешнего сайта. В примере мы покажем, какой минимальный набор действий необходимо выполнить в прикладном решении, а также на стороне веб-сайта, чтобы веб-клиент смог запуститься и работать «внутри» этого веб-сайта и взаимодействовать с внешним сайтом.

ПОДРОБНЕЕ

> Подробнее познакомиться с примером интеграции веб-клиента и внешнего сайта можно в демонстрационной конфигурации «Интеграция с 1С», которая прилагается к книге.

Для простоты в качестве веб-сайта мы будем использовать стартовую страницу index.html веб-сервера Apache, с помощью которого мы будем публиковать веб-клиент нашего прикладного решения. В нашем примере веб-клиент будет опубликован по адресу http://localhost/Web_1C.

Страница index.html находится в подкаталоге /htdocs той папки, куда устанавливался Apache. Код страницы изменим следующим образом (листинг 5.8).

Листинг 5.8. Код страницы «index.html»

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
<script src="http://localhost/Web_1C/scripts/webclient1ce.js"></script>
<title>Untitled Document</title>
<style type="text/css">
.style1 {
                                color: #0000FF;
}
.style2 {
                                color: #FF00FF;
}
.style3 {
                                color: #FF0000;
                                font-size: small;
}
</style>
</head>

<body onLoad="init();">
<h1 align="center" class="style1">Пример встраивания веб-клиента в сайт</h1>
<p align="center" class="style2">Здесь работает веб-клиент 1С:Предприятия 8</p>

<div id="webClientContainer" align="center"></div>
<button onClick="messageToWebClient();"><span class="style3">Послать сообщение веб-клиенту</span></button>
<div id="webClientMessageArea" align="center"></div>

<p align="center">
 </p>

<script>
      var webClient = null;
      var onWebClientMessage = function (message, origin) {
         if (origin === 'http://localhost') {
               document.querySelector('#webClientMessageArea').innerText = message;
            }
         };
      var messageToWebClient = function () {
         webClient.postMessage("Поступило сообщение из внешнего сайта");
      };
      var init = function () {
         webClient = new WebClient1CE('webClientContainer',
         {
            webClientURL: 'http://localhost/Web_1C?MainWindowsMode=EmbeddedWorkplace',
            width: '900px',
            height: '500px',
            events:
            {
               onMessage: onWebClientMessage
            }
         });
      };
   </script>
</body>

</html>
```

Прокомментируем некоторые элементы html-кода:

- Элемент <script> выполняет загрузку скрипта, реализующего Embedded WebClient API.
- При загрузке тела веб-страницы срабатывает обработчик init(), который указан в качестве обработчика события onload элемента body.
- Строка определяет элемент , в который будет встроен веб-клиент.
- В обработчике init() выполняется создание объекта WebClient1CE. Конструктору передается идентификатор элемента (в примере это webClientContainer). Также в обработчике init():
  - веб-клиенту передается командная строка запуска, которая указывает, что веб-клиент должен быть запущен в режиме основного окна Встроенное рабочее место;
  - для элемента <iframe>, в котором будет работать веб-клиент, устанавливается ширина, равная 900 пикселям, и высота, равная 500 пикселям. Указание данных параметров не является обязательным, вместо этого можно использовать или размер по умолчанию, или другие способы задания размера области отображения <iframe>, например CSS;
  - регистрируется обработчик события onMessage, который будет принимать сообщение от веб-клиента. Обработка этого события будет выполняться в методе onWebClientMessage(), описанном ранее на странице.
- Строка <button onclick=»messageToWebClient();»>Послать сообщение веб-клиенту</button> формирует на веб-странице кнопку с заголовком «Послать сообщение веб-клиенту», нажатие на которую приводит к тому, что в веб-клиент отправляется сообщение с текстом «Поступило сообщение из внешнего сайта». Отправка выполняется из обработчика messageToWebClient(), подключенного к событию onclick кнопки.
- Строка содержит описание области, в которую будет выводиться сообщение от веб-клиента (из обработчика события onMessage).
- Условие if (origin === 'http://localhost') в общем случае необходимо в том случае, если один обработчик используется для нескольких прикладных решений, внедренных в одну страницу. В этом случае условие будет нужно для того, чтобы понять, от какого веб-клиента поступило сообщение.

Теперь доработаем наше прикладное решение. В нашей демонстрационной конфигурации есть обработка Automation, которая использовалась для показа возможностей технологии Automation, рассмотренных выше в этой главе.

В форме этой обработки создадим команду СообщениеВнешнемуСайту и перетащим ее в окно элементов формы. Обработчик команды заполним следующим образом (листинг 5.9).

Листинг 5.9. Обработчик команды «СообщениеВнешнемуСайту»

```
&НаКлиенте
Процедура СообщениеВнешнемуСайту(Команда)

   Если ОкноВнешнегоСайта.Доступно Тогда
      СообщениеСайту = Новый СообщениеВнешнемуСайту("Получено сообщение от веб-клиента");
      ОкноВнешнегоСайта.ОтправитьСообщение(СообщениеСайту);
        КонецЕсли;

КонецПроцедуры
```

В этом обработчике мы проверяем, что веб-клиент работает внутри веб-сайта. В случае если окно внешнего веб-сайта доступно, с помощью метода `ОтправитьСообщение()` менеджера окна внешнего сайта мы отправляем сообщение сайту с текстом, заданным при создании объекта `СообщениеВнешнемуСайту`.

Затем нам нужно при открытии формы обработки подключить обработчик сообщений, которые будет получать форма от внешнего сайта. Для этого создадим обработчик события формы `ПриОткрытии` и заполним его следующим образом (листинг 5.10).

Листинг 5.10. Обработчик события формы «ПриОткрытии»

```
&НаКлиенте
Процедура ПриОткрытии(Отказ)

   Если ОкноВнешнегоСайта.Доступно Тогда
      ОбработчикСобытия = Новый ОписаниеОповещения(
                                 "ПриПолученииСообщенияОтВнешнегоСайта", ЭтотОбъект);
      ОкноВнешнегоСайта.ПодключитьОбработчикСообщений(ОбработчикСобытия);
        КонецЕсли;

КонецПроцедуры
```

В этом обработчике, в случае если окно внешнего веб-сайта доступно, мы описываем обработчик оповещения и подключаем процедуру `ПриПолученииСообщенияОтВнешнегоСайта()`, которая будет вызвана при получении сообщения из внешнего сайта. Для этого используется метод `ПодключитьОбработчикСообщений()` менеджера окна внешнего сайта.

Экспортную процедуру `ПриПолученииСообщенияОтВнешнегоСайта()` также поместим в модуле формы обработки и заполним ее следующим образом (листинг 5.11).

Листинг 5.11. Обработчик оповещения «ПриПолученииСообщенияОтВнешнегоСайта»

```
&НаКлиенте
Процедура ПриПолученииСообщенияОтВнешнегоСайта(Сообщение, ДополнительныеПараметры) Экспорт

    Сообщить(Сообщение.Данные);

КонецПроцедуры
```

В обработчике оповещения в параметре `Сообщение` содержится объект `СообщениеВнешнегоСайта`. Текст сообщения, полученного от внешнего сайта, просто выводится в окно сообщений формы обработки.

И затем нам нужно отключить обработчик сообщений `ПриПолученииСообщенияОтВнешнегоСайта()` при закрытии формы обработки. Для этого создадим обработчик события формы `ПриЗакрытии` и заполним его следующим образом (листинг 5.12).

Листинг 5.12. Обработчик события формы «ПриЗакрытии»

```
&НаКлиенте
Процедура ПриЗакрытии(ЗавершениеРаботы)

  Если ОкноВнешнегоСайта.Доступно Тогда
    ОбработчикСобытия = Новый ОписаниеОповещения(
                                "ПриПолученииСообщенияОтВнешнегоСайта", ЭтотОбъект);
    ОкноВнешнегоСайта.ОтключитьОбработчикСообщений(ОбработчикСобытия);
    КонецЕсли;

КонецПроцедуры
```

В заключение поместим форму обработки на начальную страницу приложения и опубликуем наше прикладное решение на веб-сервере по адресу `http://localhost/Web _ 1C`.

Проверим, как это работает.

Запустим веб-браузер и в адресной строке наберем адрес `http://localhost`. После этого откроется стартовая демонстрационная страница веб-сервера Apache, на котором мы опубликовали наше прикладное решение.

В соответствии с изменениями, которые мы сделали в странице index.html (см. листинг 5.8), вместо надписи «It works!» теперь на этой странице появятся окно с веб-клиентом нашего прикладного решения и кнопка для отправки сообщения веб-клиенту (рис. 5.3).

При нажатии на кнопку Сообщение внешнему сайту в форме обработки, расположенной на начальной странице веб-клиента, веб-сайту будет отправлено сообщение с текстом «Получено сообщение от веб-клиента», которое он покажет внизу веб-страницы (рис. 5.4).

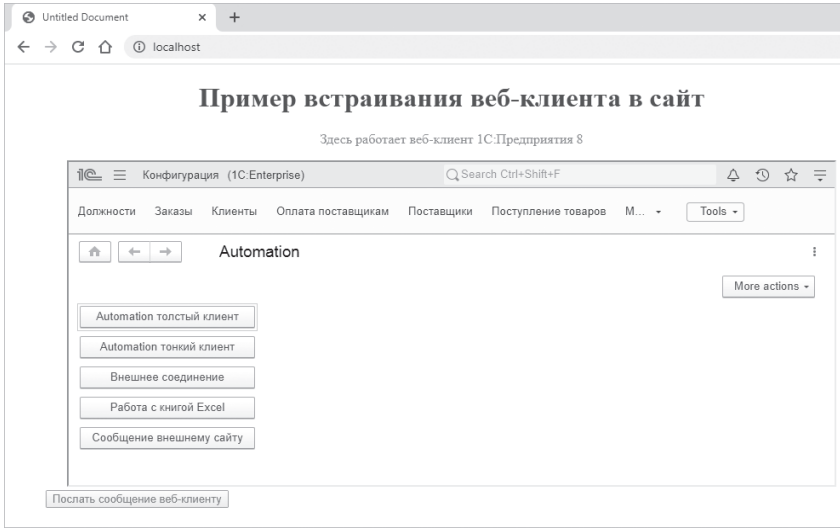


Рис. 5.3. Веб-клиент «1С:Предприятия»,
встроенный в стартовую демонстрационную страницу веб-сервера Apache

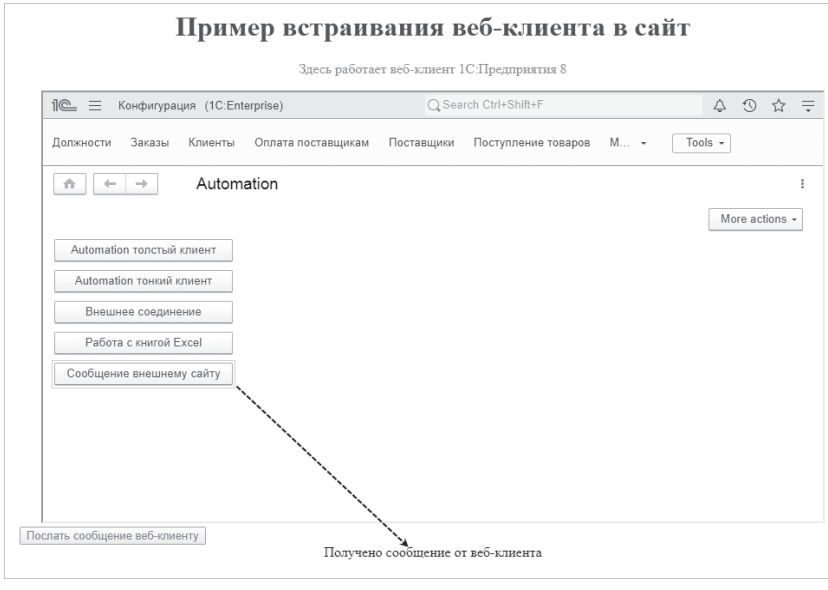


Рис. 5.4. Веб-клиент «1С:Предприятия»,
встроенный в стартовую демонстрационную страницу веб-сервера Apache

А при нажатии на кнопку Послать сообщение веб-клиенту на странице веб-сайта в окне сообщений формы обработки появится сообщение с текстом «Поступило сообщение из внешнего сайта» (рис. 5.5).

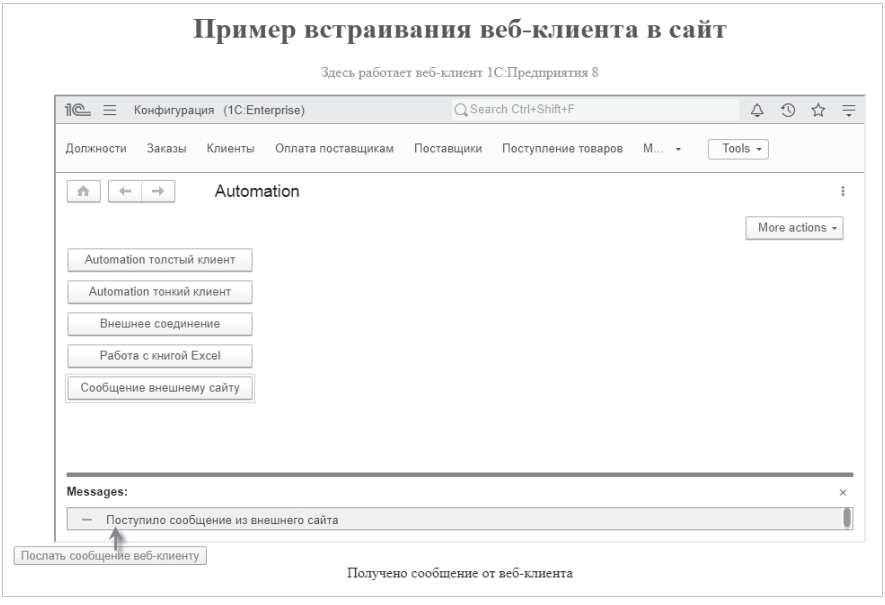


Рис. 5.5. Веб-клиент «1С:Предприятия»,
встроенный в стартовую демонстрационную страницу веб-сервера Apache

Установим теперь свойство конфигурации Режим основного окна клиентского приложения в значение Встроенное рабочее место. В результате мы увидим в окне веб-клиента только форму, находящуюся на начальной странице приложения (рис. 5.6).

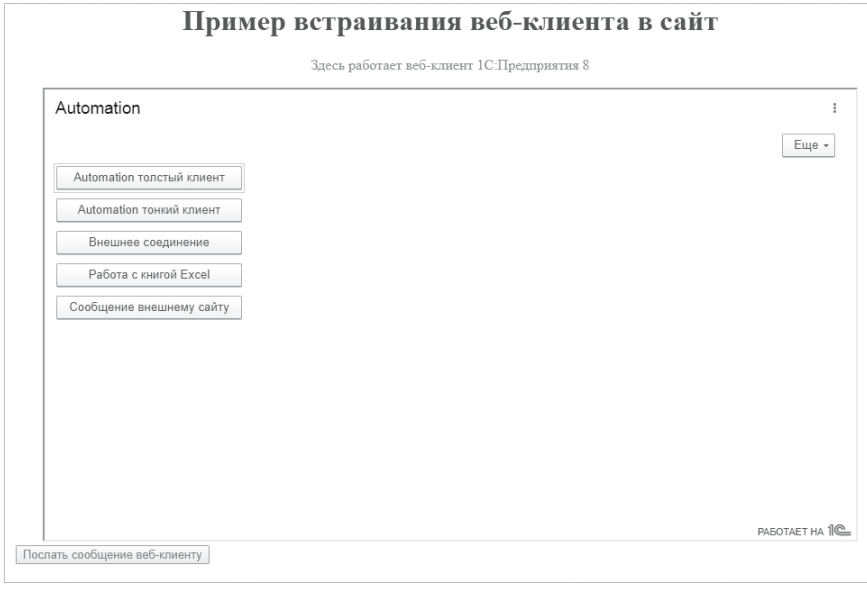


Рис. 5.6. Веб-клиент «1С:Предприятия»,
встроенный в стартовую демонстрационную страницу веб-сервера Apache

Глава 6. Файловое взаимодействие

В этой главе будут рассмотрены примеры работы с файлами различных форматов (XML, HTML, DBF и т. п.) и двоичными данными.

ПОДРОБНЕЕ

Подробнее познакомиться с приемами работы с файлами различных форматов можно в демонстрационной конфигурации «Примеры работы», которая прилагается к книге.

Работа с локальной файловой системой

В этом разделе мы покажем общие приемы работы с файлами на локальном клиентском компьютере. Для этого используются свойства и методы объекта `Файл`, а также различные процедуры и функции работы с файлами, которые представлены глобальным контекстом платформы.

С помощью объекта `Файл` можно получить короткое имя файла, расширение, имя без расширения, каталог, в котором файл находится, и т. п.

Ниже будут рассмотрены небольшие примеры решения наиболее часто встречающихся задач – поиска и удаления файлов, создания и удаления каталога и т. д.

В форме нашей демонстрационной обработки используются следующие параметры, которые хранятся в соответствующих строковых реквизитах формы:

- МаскаФайлов – маска, по которой требуется найти или удалить файлы в каталоге. Например, «*.xml»;
- ИмяНовогоФайла – имя для копирования выбранного файла;
- ИмяНовогоКаталога – имя для создания нового каталога.

Найти файлы в каталоге

Предположим, нам нужно найти файлы, находящиеся в определенном каталоге, которые соответствуют маске, заданной в поле ввода демонстрационной обработки (реквизит МаскаФайлов).

Для решения этой задачи добавим команду НайтиФайлы. Обработчик команды заполним следующим образом (листинг 6.1).

Листинг 6.1. Обработчик команды «НайтиФайлы»

```
&НаКлиенте
Процедура НайтиФайлыВКаталоге(Команда)

    ВыбратьКаталог("Выбор каталога для поиска файлов", Новый ОписаниеОповещения(
                    "ПослеВыбораКаталогаДляПоиска", ЭтотОбъект));

КонецПроцедуры
```

Сначала нам нужно выбрать каталог, в котором будет выполняться поиск файлов. Для этого мы вызываем процедуру `ВыбратьКаталог()`, в которую передаем заголовок диалога и описание оповещения, указывающее на экспортную процедуру, которая будет выполнена после того, как пользователь выберет каталог (листинг 6.2).

Листинг 6.2. Процедура «ВыбратьКаталог()»

```
&НаКлиенте
Процедура ВыбратьКаталог(ЗаголовокДиалога, Оповещение)

    Диалог = Новый ДиалогВыбораФайла(РежимДиалогаВыбораФайла.ВыборКаталога);
    Диалог.Заголовок = ЗаголовокДиалога;
    Диалог.Показать(Оповещение);

КонецПроцедуры
```

Отображение диалога для выбора мы будем выполнять с помощью немодального метода `Показать()` объекта `ДиалогВыбораФайла`. В этот метод мы

передаем описание оповещения, содержащееся в параметре `Оповещение` и указывающее на экспортную процедуру `ПослеВыбораКаталогаДляПоиска()`, которая будет вызвана после выбора каталога пользователем (листинг 6.3).

Листинг 6.3. Процедура «ПослеВыбораКаталогаДляПоиска()»

```
&НаКлиенте
Процедура ПослеВыбораКаталогаДляПоиска(ВыбранныеФайлы, Параметры) Экспорт

    Если ВыбранныеФайлы = Неопределено Тогда
        Возврат;
    КонецЕсли;

    КаталогПоиска = ВыбранныеФайлы[0];
    Если ПустаяСтрока(МаскаФайлов) Тогда
        МаскаФайлов = "*.*";
    КонецЕсли;

    // Найти файлы в выбранном каталоге по указанной маске.
    НачатьПоискФайлов(Новый ОписаниеОповещения(
                    "НайтиФайлыВКаталогеЗавершение", ЭтотОбъект), КаталогПоиска, МаскаФайлов);

КонецПроцедуры
```

В этом обработчике оповещения, в случае если каталог выбран, мы получаем путь к каталогу как первый элемент массива, содержащийся в параметре `ВыбранныеФайлы`, и вызываем процедуру `НачатьПоискФайлов()`, в которую передаем этот каталог и маску для поиска файлов, заданную в поле ввода МаскаФайлов.

Первым параметром в немодальный метод `НачатьПоискФайлов()` мы передаем описание оповещения, указывающее на экспортную процедуру `НайтиФайлыВКаталогеЗавершение()`, которая будет вызвана по окончании поиска файлов (листинг 6.4).

Листинг 6.4. Процедура «НайтиФайлыВКаталогеЗавершение()»

```
&НаКлиенте
Процедура НайтиФайлыВКаталогеЗавершение(МассивФайлов, Дополнительно) Экспорт

    Сообщение = Новый СообщениеПользователю;

    Для Каждого Файл Из МассивФайлов Цикл
        Сообщение.Текст = "Найден файл - " + Файл.Имя;
        Сообщение.Сообщить();
    КонецЦикла;

КонецПроцедуры
```

В этой процедуре в параметре МассивФайлов содержится массив найденных файлов в виде объектов Файл. В цикле обхода этого массива мы показываем короткое имя каждого файла (имя с расширением – Файл.Имя) в сообщении пользователю.

Удалить файлы в каталоге

Предположим, нам нужно удалить файлы, находящиеся в определенном каталоге, которые соответствуют маске, заданной в поле ввода демонстрационной обработки (реквизит МаскаФайлов).

Для решения этой задачи добавим команду УдалитьФайлыВКаталоге. Обработчик команды заполним следующим образом (листинг 6.5).

Листинг 6.5. Обработчик команды «УдалитьФайлыВКаталоге»

```
&НаКлиенте
Процедура УдалитьФайлыВКаталоге(Команда)

    ВыбратьКаталог("Выбор каталога для удаления файлов", Новый ОписаниеОповещения(
                                                    "ПослеВыбораКаталогаДляУдаления", ЭтотОбъект));

КонецПроцедуры
```

Сначала нам нужно выбрать каталог, в котором будет выполняться удаление файлов. Для этого мы вызываем процедуру ВыбратьКаталог(), которая была рассмотрена ранее, в листинге 6.2. В эту процедуру мы передаем заголовок диалога и описание оповещения, указывающее на экспортную процедуру ПослеВыбораКаталогаДляУдаления(), которая будет выполнена после того, как пользователь выберет каталог (листинг 6.6).

Листинг 6.6. Процедура «ПослеВыбораКаталогаДляУдаления()»

```
&НаКлиенте
Процедура ПослеВыбораКаталогаДляУдаления(ВыбранныеФайлы, Параметры) Экспорт

    Если ВыбранныеФайлы = Неопределено Тогда
        Возврат;
    КонецЕсли;

    КаталогУдаления = ВыбранныеФайлы[0];
    Если ПустаяСтрока(МаскаФайлов) Тогда
        МаскаФайлов = "*.*";
    КонецЕсли;

    // Удалить все файлы в выбранном каталоге по указанной маске.
    НачатьУдалениеФайлов(Новый ОписаниеОповещения(
                        "УдалитьФайлыВКаталогеЗавершение", ЭтотОбъект, КаталогУдаления),
        КаталогУдаления, МаскаФайлов);

КонецПроцедуры
```

В этом обработчике оповещения, в случае если каталог выбран, мы получаем путь к каталогу как первый элемент массива, содержащийся в параметре ВыбранныеФайлы, и вызываем процедуру НачатьПоискФайлов(), в которую передаем этот каталог и маску для поиска файлов, заданную в поле ввода МаскаФайлов.

Первым параметром в немодальный метод НачатьУдалениеФайлов() мы передаем описание оповещения, указывающее на экспортную процедуру УдалитьФайлыВКаталогеЗавершение(), которая будет вызвана по окончании удаления файлов (листинг 6.7).

Листинг 6.7. Процедура «УдалитьФайлыВКаталогеЗавершение()»

```
&НаКлиенте
Процедура УдалитьФайлыВКаталогеЗавершение(Каталог) Экспорт

    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = "Файлы по маске: " + МаскаФайлов + " в каталоге: " + Каталог + " удалены";
    Сообщение.Сообщить();

КонецПроцедуры
```

При создании этого обработчика оповещения в качестве дополнительного параметра мы передали в эту процедуру каталог, в котором удалялись файлы. Значение параметра Каталог и маску для удаления файлов мы выводим в сообщении пользователю после удаления файлов.

Обратите внимание, что в нашем примере реквизит МаскаФайлов заполняется значением «*.*», в случае если маска файлов не указана (см. листинг 6.6). В результате в выбранном каталоге могут быть удалены все файлы, но сам каталог будет существовать. Если нужно удалить и сам пустой каталог, то третий параметр метода УдалитьФайлыВКаталогеЗавершение(), в котором задается маска для удаления файлов, можно не указывать.

Создать новый каталог

Предположим, нам нужно создать новый подкаталог для определенного каталога. Короткое имя этого нового каталога задано в поле ввода демонстрационной обработки (реквизит ИмяНовогоКаталога).

Для решения этой задачи добавим команду СоздатьКаталог. Обработчик команды заполним следующим образом (листинг 6.8).

Листинг 6.8. Обработчик команды «СоздатьКаталог»

```
&НаКлиенте
Процедура СоздатьНовыйКаталог(Команда)

    ВыбратьКаталог("Выбор родительского каталога для создания каталога",
        Новый ОписаниеОповещения("ПослеВыбораКаталогаДляСоздания", ЭтотОбъект));

КонецПроцедуры
```

Сначала нам нужно выбрать каталог, для которого будет создаваться дочерний подкаталог. Для этого мы вызываем процедуру `ВыбратьКаталог()`, которая была рассмотрена ранее, в листинге 6.2. В эту процедуру мы передаем заголовок диалога и описание оповещения, указывающее на экспортную процедуру `ПослеВыбораКаталогаДляСоздания()`, которая будет выполнена после того, как пользователь выберет каталог (листинг 6.9).

Листинг 6.9. Процедура «ПослеВыбораКаталогаДляСоздания()»

```
&НаКлиенте
Процедура ПослеВыбораКаталогаДляСоздания(ВыбранныеФайлы, Параметры) Экспорт

    Если ВыбранныеФайлы = Неопределено Тогда
        Возврат;
    КонецЕсли;

    РодительскийКаталог = ВыбранныеФайлы[0];
    ИмяКаталога = РодительскийКаталог + "\" + ИмяНовогоКаталога + "\";

    // Создать подкаталог в выбранном каталоге.
    НачатьСозданиеКаталога(Новый ОписаниеОповещения(
                                "СоздатьКаталогЗавершение", ЭтотОбъект), ИмяКаталога);

КонецПроцедуры
```

В этом обработчике оповещения, в случае если каталог выбран, мы получаем путь к родительскому каталогу как первый элемент массива, содержащийся в параметре `ВыбранныеФайлы`. Чтобы получить полный путь к создаваемому каталогу, прибавляем к нему «через слеш» имя нового каталога. Затем вызываем процедуру `НачатьСозданиеКаталога()`, в которую передаем этот путь (`ИмяКаталога`).

Первым параметром в немодальный метод `НачатьСозданиеКаталога()` мы передаем описание оповещения, указывающее на экспортную процедуру `СоздатьКаталогЗавершение()`, которая будет вызвана по окончании создания каталога (листинг 6.10).

Листинг 6.10. Процедура «СоздатьКаталогЗавершение()»

```
&НаКлиенте
Процедура СоздатьКаталогЗавершение(ИмяКаталога, Дополнительно) Экспорт

	Сообщение = Новый СообщениеПользователю;
	Сообщение.Текст = "Каталог: " + ИмяКаталога + " создан";
	Сообщение.Сообщить();

КонецПроцедуры
```

Если создаваемый каталог уже существует, то повторно он создан не будет и ошибки не возникнет.

Копировать файл

Предположим, нам нужно скопировать файл, находящийся в определенном каталоге, в тот же каталог, но с новым именем, которое задано в поле ввода демонстрационной обработки (реквизит ИмяНовогоФайла).

Для решения этой задачи добавим команду КопироватьФайлСНовымИменем. Обработчик команды заполним следующим образом (листинг 6.11).

Листинг 6.11. Обработчик команды «КопироватьФайлСНовымИменем»

```
&НаКлиенте
Процедура КопироватьФайлСНовымИменем(Команда)

	Диалог = Новый ДиалогВыбораФайла(РежимДиалогаВыбораФайла.Открытие);
	Диалог.Заголовок = "Выберите файл";
	Диалог.Показать(Новый ОписаниеОповещения("КопироватьФайлЗавершение", ЭтотОбъект));

КонецПроцедуры
```

В этом обработчике мы выбираем файл, который нам нужно скопировать. Для этого мы показываем пользователю диалог выбора файла с помощью немодального метода `Показать()`. В этот метод мы передаем описание оповещения, указывающее на экспортную процедуру `КопироватьФайлЗавершение()`, которая будет выполнена после того, как пользователь выберет файл для копирования (листинг 6.12).

Листинг 6.12. Процедура «КопироватьФайлЗавершение()»

```
&НаКлиенте
Процедура КопироватьФайлЗавершение(ВыбранныеФайлы, Параметры) Экспорт

	Если ВыбранныеФайлы = Неопределено Тогда
		Возврат;
	КонецЕсли;
```

```
    ИмяФайлаИсточника = ВыбранныеФайлы[0];
    ФайлИсточник = Новый Файл(ИмяФайлаИсточника);
    ИмяФайлаПриемника = ФайлИсточник.Путь + СокрЛП(ИмяНовогоФайла) + ФайлИсточник.Расширение;

    НачатьКопированиеФайла(Новый ОписаниеОповещения(
                        "КопироватьФайлСНовымИменемЗавершение", ЭтотОбъект),
        ИмяФайлаИсточника, ИмяФайлаПриемника);

КонецПроцедуры
```

В этом обработчике оповещения, в случае если файл выбран, мы получаем полный путь к файлу-источнику как первый элемент массива, содержащийся в параметре `ВыбранныеФайлы` (для простоты примера будем выбирать только один файл).

Затем создаем на основе пути к файлу-источнику объект `Файл`. Используя свойства этого объекта, получаем отдельно путь (`ФайлИсточник.Путь`) и расширение (`ФайлИсточник.Расширение`) файла-источника. Вставив имя нового файла, мы получаем полный путь к файлу-приемнику. После этого мы вызываем процедуру `НачатьКопированиеФайла()`, в которую передаем полные пути к файлу-источнику и файлу-приемнику.

Первым параметром в немодальный метод `НачатьКопированиеФайла()` мы передаем описание оповещения, указывающее на экспортную процедуру `КопироватьФайлСНовымИменемЗавершение()`, которая будет вызвана по окончании копирования файла (листинг 6.13).

Листинг 6.13. Процедура «КопироватьФайлСНовымИменемЗавершение()»

```
&НаКлиенте
Процедура КопироватьФайлСНовымИменемЗавершение(СкопированныйФайл, Дополнительно) Экспорт

    РезультирующийФайл = Новый Файл(СкопированныйФайл);

    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = "Выбранный файл скопирован в: " + РезультирующийФайл.Имя;
    Сообщение.Сообщить();

КонецПроцедуры
```

В этой процедуре в параметре `СкопированныйФайл` содержится полный путь к новому файлу. На его основе мы создаем объект `Файл`. И затем показываем короткое имя нового файла (имя с расширением – `РезультирующийФайл.Имя`) в сообщении пользователю.

Переместить файл

Предположим, что файл находится в некотором каталоге. Нам нужно создать рядом с файлом новый каталог, и переместить файл в него. Имя нового каталога задано в поле ввода демонстрационной обработки (реквизит ИмяНовогоКаталога). Этот каталог мы создадим перед перемещением файла.

Для решения этой задачи добавим команду ПереместитьФайлВНовыйКаталог. Обработчик команды заполним следующим образом (листинг 6.14).

Листинг 6.14. Обработчик команды «ПереместитьФайлВНовыйКаталог»

```
&НаКлиенте
Процедура КопироватьФайлСНовымИменем(Команда)

    Диалог = Новый ДиалогВыбораФайла(РежимДиалогаВыбораФайла.Открытие);
    Диалог.Заголовок = "Выберите файл";
    Диалог.Показать(Новый ОписаниеОповещения("ПереместитьФайлЗавершение", ЭтотОбъект));

КонецПроцедуры
```

В этом обработчике мы выбираем файл, который нам нужно переместить. Для этого мы показываем пользователю диалог выбора файла с помощью немодального метода `Показать()`. В этот метод мы передаем описание оповещения, указывающее на экспортную процедуру `ПереместитьФайлЗавершение()`, которая будет выполнена после того, как пользователь выберет файл для перемещения (листинг 6.15).

Листинг 6.15. Процедура «ПереместитьФайлЗавершение()»

```
&НаКлиенте
Процедура ПереместитьФайлЗавершение(ВыбранныеФайлы, Параметры) Экспорт

    Если ВыбранныеФайлы = Неопределено Тогда
        Возврат;
    КонецЕсли;

    ИмяФайлаИсточника = ВыбранныеФайлы[0];
    ФайлИсточник = Новый Файл(ИмяФайлаИсточника);
    ИмяФайлаПриемника = ФайлИсточник.Путь + СокрЛП(ИмяНовогоКаталога) + "\" + ФайлИсточник.Имя;

    ДопПараметры = Новый Структура("ИмяФайлаИсточника, ИмяФайлаПриемника"
                                   , ИмяФайлаИсточника, ИмяФайлаПриемника);
    НачатьСозданиеКаталога(Новый ОписаниеОповещения("СоздатьКаталогИПереместитьЗавершение"
                                   , ЭтотОбъект, ДопПараметры),
        ФайлИсточник.Путь + СокрЛП(ИмяНовогоКаталога) + "\");

КонецПроцедуры
```

В этом обработчике оповещения, в случае если файл выбран, мы получаем полный путь к файлу-источнику как первый элемент массива, содержащийся в параметре ВыбранныеФайлы (для простоты примера будем выбирать только один файл).

Затем создаем на основе пути к файлу-источнику объект Файл. Используя свойства этого объекта, получаем отдельно путь (ФайлИсточник.Путь) и имя файла с расширением (ФайлИсточник.Имя). Вставив имя нового каталога, мы получаем полный путь к файлу-приемнику.

После этого упаковываем полные пути к файлу-источнику и файлу-приемнику в структуру ДопПараметры.

Чтобы создать новый каталог для перемещения файла, мы вызываем метод НачатьСозданиеКаталога(), в который передаем полный путь к создаваемому каталогу (ФайлИсточник.Путь + СокрЛП(ИмяНовогоКаталога) + «\»).

Первым параметром в немодальный метод НачатьСозданиеКаталога() мы передаем описание оповещения, указывающее на экспортную процедуру СоздатьКаталогИПереместитьЗавершение(), которая будет вызвана по окончании создания каталога (листинг 6.16). Кроме того, при создании этого обработчика оповещения мы заполняем дополнительные параметры структурой ДопПараметры, содержащей полные пути к файлу-источнику и файлу-приемнику.

Листинг 6.16. Процедура «СоздатьКаталогИПереместитьЗавершение()»

```
&НаКлиенте
Процедура СоздатьКаталогИПереместитьЗавершение(Каталог, ДопПараметры) Экспорт

    НачатьПеремещениеФайла(Новый ОписаниеОповещения(
                                "ПереместитьФайлВНовыйКаталогЗавершение", ЭтотОбъект),
        ДопПараметры.ИмяФайлаИсточника, ДопПараметры.ИмяФайлаПриемника);

КонецПроцедуры
```

Когда каталог для перемещения файла будет создан, мы вызываем немодальный метод НачатьПеремещениеФайла() и передаем в него поля структуры ДопПараметры (полные пути к файлу-источнику и файлу-приемнику). Первым параметром мы передаем в этот метод описание оповещения, указывающее на экспортную процедуру ПереместитьФайлВНовыйКаталогЗавершение(), которая будет вызвана по окончании перемещения файла (листинг 6.17).

Листинг 6.17. Процедура «ПереместитьФайлВНовыйКаталогЗавершение()»

```
&НаКлиенте
Процедура ПереместитьФайлВНовыйКаталогЗавершение(ПеремещенныйФайл, Дополнительно) Экспорт

    РезультирующийФайл = Новый Файл(ПеремещенныйФайл);

    Сообщение = Новый СообщениеПользователю;
    Сообщение.Текст = "Файл: " + РезультирующийФайл.Имя + " перемещен в каталог: "
                                                    + ИмяНовогоКаталога;
    Сообщение.Сообщить();

КонецПроцедуры
```

В этой процедуре в параметре `ПеремещенныйФайл` содержится полный путь к новому файлу. На его основе мы создаем объект `Файл`. И затем показываем короткое имя нового файла (имя с расширением – `РезультирующийФайл.Имя`) в сообщении пользователю.

Передача файлов между клиентом и сервером

Для передачи файлов между клиентом и сервером через временное хранилище в платформе «1С:Предприятия» реализованы новые методы глобального контекста `НачатьПомещениеФайлаНаСервер()`, `НачатьПомещениеФайловНаСервер()`, `НачатьПолучениеФайлаССервера()`, `НачатьПолучениеФайловССервера()`. Существенным преимуществом этих методов является возможность их работы интерактивном режиме (например, при отображении диалога выбора помещаемых/получаемых файлов). В этом режиме работы в веб-клиенте методы не требуют установки расширения работы с файлами.

При помещении файлов на сервер есть возможность отображения прогресса помещения файлов. Можно проверить характеристики помещаемых файлов и отказаться от загрузки перед началом или в процессе помещения файлов на сервер. Кроме того, можно несколько раз вызывать методы для помещения файлов на сервер, не дожидаясь окончания предыдущего вызова.

При получении файлов с сервера есть очень удобная возможность получить несколько файлов в виде архива.

При выполнении некоторых операций в веб-клиенте может потребоваться получить разрешение на операции по работе с файлами. Чтобы не подтверждать каждую такую операцию по отдельности, можно воспользоваться методом `НачатьЗапросРазрешенияПользователя()`. При использовании этого метода пользователю отображается список всех операций, которые планируется выполнить, и предлагается разрешить выполнение

группы операций. Если пользователь разрешил выполнение, то запрошенные операции будут выполняться без дополнительных запросов пользователю. Если разрешение не предоставлено, операции будут выполняться в обычном режиме: один запрос на каждую операцию.

В тонком и толстом клиентских приложениях метод `НачатьЗапросРазрешенияПользователя()` всегда возвращает значение `Истина`, без взаимодействия с пользователем.

Передача и получение одного файла с сервера

Предположим, у нас есть сформированный ранее JSON-файл (листинг 6.18) и нам нужно прочитать из него информацию, изменить ее и записать в новый файл. Хотя сделать все это можно и на клиенте, более правильно – передать файл на сервер (поместить его на сервере во временное хранилище), обработать там содержимое файла, вернуть адрес измененных данных на клиент и затем по этому адресу получить файл.

Листинг 6.18. Исходный JSON-файл

```
{
	"Контрагент": "ОАО Топаз",
	"ОбъемПродаж": 5000000,
	"Адрес": "Страна: Россия, Индекс: 112233, Город: Москва",
	"Поставщик": false
}
```

Например, в показанном выше файле с информацией о контрагенте нам нужно заменить вывод значений true/false поля Поставщик (типа `Булево`) на более привычные Да/Нет.

Для решения этой задачи добавим команду ОбработатьФайлНаСервере. Обработчик команды заполним следующим образом (листинг 6.19).

Листинг 6.19. Обработчик команды «ОбработатьФайлНаСервере»

```
&НаКлиенте
Процедура ОбработатьФайлНаСервере(Команда)

	ОповещениеОЗавершении = Новый ОписаниеОповещения(
				"ОбработатьФайлНаСервереЗавершение", ЭтотОбъект);
	НачатьПомещениеФайлаНаСервер(ОповещениеОЗавершении, , , "", , УникальныйИдентификатор);
	//НачатьПомещениеФайлаНаСервер(ОповещениеОЗавершении, , , "", "c:\temp\example.json"
				, УникальныйИдентификатор);

КонецПроцедуры
```

В этом обработчике с помощью метода глобального контекста НачатьПомещениеФайлаНаСервер() мы начинаем помещение файла с данными из локальной файловой системы во временное хранилище.

При этом пользователю предлагается диалог для выбора помещаемого файла. Параметры этого диалога мы можем описать и передать в пятом параметре метода. Но это будет сделано в следующем примере. В данном случае мы опускаем этот параметр, но диалог все равно будет показан.

Можно также использовать вариант вызова этого метода с указанием в пятом параметре имени помещаемого файла (см. закомментированный фрагмент кода), но следует иметь в виду, что для работы этого метода в веб-клиенте потребуется установить расширение для работы с файлами. В то время как в первом случае это расширение не требуется.

В метод НачатьПомещениеФайлаНаСервер() первым параметром мы передаем описание оповещения, указывающее на экспортную процедуру ОбработатьФайлНаСервереЗавершение(), которая будет выполнена, после того как файл с данными будет помещен во временное хранилище (листинг 6.20).

Листинг 6.20. Процедура «ОбработатьФайлНаСервереЗавершение()»

```
&НаКлиенте
Процедура ОбработатьФайлНаСервереЗавершение(ОписаниеПомещенногоФайла, Дополнительно) Экспорт

    ПомещенныйФайл = ОписаниеПомещенногоФайла.ССылкаНаФайл.Файл;
    ПолучаемыйФайл = ПомещенныйФайл.Путь + ПомещенныйФайл.ИмяБезРасширения + "_2"
                                    + ПомещенныйФайл.Расширение;
    АдресДокументаВХранилище = ОбработкаФайла(ОписаниеПомещенногоФайла.Адрес);

    //НачатьПолучениеФайлаССервера(АдресДокументаВХранилище);
    НачатьПолучениеФайлаССервера(, АдресДокументаВХранилище, ПолучаемыйФайл);

КонецПроцедуры
```

После того как файл будет помещен, в параметре ОписаниеПомещенногоФайла будет содержаться описание помещенного файла (адрес данных в хранилище и ссылка на файл). Адрес помещенного файла (ОписаниеПомещенногоФайла.Адрес) мы передаем в серверную функцию ОбработкаФайла(), в которой данные по этому адресу получаются из временного хранилища, записываются во временный файл и производится чтение файла (листинг 6.21).

Листинг 6.21. Функция «ОбработкаФайла()»

```
&НаСервереБезКонтекста
Функция ОбработкаФайла(АдресДокументаВХранилище)

	ДанныеДокумента = ПолучитьИзВременногоХранилища(АдресДокументаВХранилище);
	ИмяФайла = КаталогВременныхФайлов() + "temp.xml";
	ДанныеДокумента.Записать(ИмяФайла);

	Чтение = Новый ЧтениеJSON;
	Чтение.ОткрытьФайл(ИмяФайла);

	Запись = Новый ЗаписьJSON;
	Запись.УстановитьСтроку();
	Запись.ЗаписатьНачалоОбъекта();

	Пока Чтение.Прочитать() Цикл

		Если Чтение.ТипТекущегоЗначения = ТипЗначенияJSON.ИмяСвойства Тогда

			Если Чтение.ТекущееЗначение = "Поставщик" Тогда
				Чтение.Прочитать();
				Запись.ЗаписатьИмяСвойства("Поставщик");
				Запись.ЗаписатьЗначение(Строка(Чтение.ТекущееЗначение));
			Иначе
				Запись.ЗаписатьИмяСвойства(Чтение.ТекущееЗначение);
			КонецЕсли;

		КонецЕсли;

		Если Чтение.ТипТекущегоЗначения = ТипЗначенияJSON.Число Или
			Чтение.ТипТекущегоЗначения = ТипЗначенияJSON.Строка Тогда
			Запись.ЗаписатьЗначение(Чтение.ТекущееЗначение);
		КонецЕсли;

	КонецЦикла;

	Чтение.Закрыть();

	Запись.ЗаписатьКонецОбъекта();
	СтрокаJSON = Запись.Закрыть();

	Возврат ПоместитьВоВременноеХранилище(СтрокаJSON);

КонецФункции
```

В данной функции в режиме потокового чтения и записи производятся чтение, изменение и запись данных в формате JSON. Этот вопрос подробно рассматривался в разделе «Потоковая работа. Запись и чтение» и здесь приведен просто в качестве примера.

Отметим лишь, что для поля Поставщик типа `Булево` мы записываем строковое представление значения, в результате значения true/false будут заменены на Да/Нет.

В заключение в функции ОбработкаФайла() измененные данные записываются в строку JSON, которая помещается во временное хранилище, и адрес этих данных в хранилище возвращается в процедуру ОбработатьФайлНаСервереЗавершение(), приведенную выше (см. листинг 6.20).

Затем в этой процедуре с помощью метода глобального контекста НачатьПолучениеФайлаССервера() мы начинаем получение файла по адресу во временном хранилище и сохраняем файл в локальную файловую систему пользователя. При этом мы явно указываем имя получаемого файла.

Отметим, что для работы этого варианта метода (с указанием имени получаемого файла) в веб-клиенте требуется установить расширение работы с файлами.

Для формирования имени получаемого файла мы на основе описания помещенного файла (ОписаниеПомещенногоФайла.ССылкаНаФайл.Файл) создаем объект Файл. Используя свойства этого объекта: Путь, ИмяБезРасширения и Расширение, мы формируем имя сохраняемого файла с постфиксом «_2».

Передача нескольких файлов на сервер

Предположим, у нас на локальном диске есть файлы изображений с картинками товаров и нам нужно поместить эти картинки в виде двоичных данных в справочник товаров. Будем считать, что для соотнесения файла картинки и товара имена файлов должны совпадать с наименованиями товаров.

Для решения этой задачи добавим команду УстановитьКартинкиТоваров. Обработчик команды заполним следующим образом (листинг 6.22).

Листинг 6.22. Обработчик команды «УстановитьКартинкиТоваров»

```
&НаКлиенте
Процедура УстановитьКартинкиТоваров(Команда)

    ОповещениеОЗавершении = Новый ОписаниеОповещения(
                                "УстановитьКартинкиТоваровЗавершение", ЭтотОбъект);

    //СтрокаПоиска = "e:\pictures\*.jpg";
    //НачатьПомещениеФайловНаСервер(ОповещениеОЗавершении, , ,
                                СтрокаПоиска, УникальныйИдентификатор);

    ПараметрыДиалога = Новый ПараметрыДиалогаПомещенияФайлов("Выберите файлы картинок"
                                , Истина, "*.jpg");
    НачатьПомещениеФайловНаСервер(ОповещениеОЗавершении, , , ПараметрыДиалога
                                , УникальныйИдентификатор);

    // ОповещениеОХодеВыполнения = Новый ОписаниеОповещения(
    // "УстановитьКартинкиТоваровХодВыполнения", ЭтотОбъект);
    // НачатьПомещениеФайловНаСервер(ОповещениеОЗавершении, ОповещениеОХодеВыполнения
    // , , ПараметрыДиалога, УникальныйИдентификатор);

КонецПроцедуры
```

В этом обработчике сначала мы создаем описание оповещения, указывающее на экспортную процедуру `УстановитьКартинкиТоваровЗавершение()`, которая будет выполнена, после того как все файлы картинок будут помещены во временное хранилище (листинг 6.23).

Затем мы создаем объект `ПараметрыДиалогаПомещенияФайлов` и задаем заголовок, возможность множественного выбора файлов и строку фильтра файлов (в нашем случае «*.jpg»), отображаемых для выбора в диалоге. В общем случае можно задать несколько вариантов фильтра (например: «Текстовый документ(*.txt)|*.txt;*rtf |Табличный документ(*.mxl)|*.mxl»), а при отображении диалога выбрать нужный фильтр.

И передаем описание оповещения о завершении и параметры диалога помещения файлов в метод глобального контекста `НачатьПомещениеФайловНаСервер()`, с помощью которого начинается помещение файлов с картинками товаров из локальной файловой системы во временное хранилище. При этом пользователю показывается диалог выбора файлов, соответствующих заданному фильтру.

Можно также использовать вариант вызова этого метода с указанием в четвертом параметре строки поиска файлов вместо диалога (см. закомментированный фрагмент кода), но следует иметь в виду, что для работы этого метода в веб-клиенте потребуется установить расширение для работы с файлами. В то время как в первом случае это расширение не требуется.

Листинг 6.23. Процедура «УстановитьКартинкиТоваровЗавершение()»

```
&НаКлиенте
Процедура УстановитьКартинкиТоваровЗавершение(ПомещенныеФайлы, Дополнительно) Экспорт

    Если ПомещенныеФайлы = Неопределено Тогда
        Возврат;
    КонецЕсли;

    Для Каждого ПомещенныйФайл Из ПомещенныеФайлы Цикл
        НаименованиеТовара = ПомещенныйФайл.ССылкаНаФайл.Файл.ИмяБезРасширения;
        ИмяФайла = ПомещенныйФайл.ССылкаНаФайл.Имя;
        АдресФайла = ПомещенныйФайл.Адрес;
        ПоместитьФайлКартинки(НаименованиеТовара, ИмяФайла, АдресФайла);
    КонецЦикла;

КонецПроцедуры
```

В этой процедуре в параметре `ПомещенныеФайлы` будет содержаться массив объектов `ОписаниеПомещенногоФайла` (адрес данных в хранилище и ссылка на помещенный файл).

По мере обхода элементов этого массива мы получаем по ссылке на каждый помещенный файл имя файла без расширения и имя файла с расширением, а также получаем адрес каждого помещенного файла картинки.

И передаем все эти параметры в серверную процедуру `ПоместитьФайлКартинки()`, в которой данные по этому адресу получаются из временного хранилища и записываются в виде двоичных данных в реквизит товара типа `ХранилищеЗначения` (листинг 6.24).

Листинг 6.24. Процедура «ПоместитьФайлКартинки()»

```
&НаСервереБезКонтекста
Процедура ПоместитьФайлКартинки(НаименованиеТовара, ИмяФайла, АдресФайла)

	ТоварСсылка = Справочники.Товары.НайтиПоНаименованию(НаименованиеТовара, Истина);

	Если ТоварСсылка <> Справочники.Товары.ПустаяСсылка() Тогда

		Товар = ТоварСсылка.ПолучитьОбъект();

		ДвоичныеДанные = ПолучитьИзВременногоХранилища(АдресФайла);
		Товар.ДанныеФайлаКартинки = Новый ХранилищеЗначения(ДвоичныеДанные
					, Новый СжатиеДанных());
		Товар.ИмяФайлаКартинки = ИмяФайла;
		Товар.Записать();

		УдалитьИзВременногоХранилища(АдресФайла);
	КонецЕсли;

КонецПроцедуры
```

В этой процедуре в справочнике товаров находится товар с наименованием, совпадающим с именем помещенного файла без расширения (параметр `НаименованиеТовара`). Затем на основе двоичных данных, полученных из временного хранилища по адресу файла (параметр `АдресФайла`), создается объект `ХранилищеЗначения` и записывается в реквизит товара ДанныеФайлаКартинки, а в реквизите ИмяФайлаКартинки сохраняется имя (с расширением) помещенного файла (параметр `ИмяФайла`).

Если на сервер помещается большой объем данных, то имеет смысл вывести в окно состояния прогресс помещения файлов. Для этого в метод `НачатьПомещениеФайловНаСервер()` вторым параметром нужно передать описание оповещения о ходе выполнения (см. закомментированный фрагмент кода в листинге 6.22), указывающее на экспортную процедуру, которая будет вызвана несколько раз по мере помещения файлов (листинг 6.25).

Листинг 6.25. Процедура «УстановитьКартинкиТоваровХодВыполнения()»

```
&НаКлиенте
Процедура УстановитьКартинкиТоваровХодВыполнения(ПомещаемыйФайлы, Помещено
                , ОтказОтПомещенияФайла, ПомещеноВсего, ОтказОтПомещенияФайлов, Дополнительно) Экспорт

        Состояние("Помещение файлов картинок товаров", ПомещеноВсего);

КонецПроцедуры
```

В параметре `ПомещеноВсего` содержится процент помещенной части всех файлов, который мы и выводим в окно Состояние (рис. 6.1).

Рис. 6.1. Информация о ходе помещения файлов на сервер

Получение нескольких файлов с сервера

Рассмотрим теперь обратную задачу. Предположим, нам нужно сохранить на локальном компьютере все картинки товаров в виде файлов изображений и, возможно, поместить их в архив.

Для решения этой задачи добавим команду СохранитьКартинкиТоваровНаДиск. Обработчик команды заполним следующим образом (листинг 6.26).

Листинг 6.26. Обработчик команды «СохранитьКартинкиТоваровНаДиск»

```
&НаКлиенте
Процедура СохранитьКартинкиТоваровНаДиск(Команда)

        ОписаниеФайловКартинок = ПолучитьКартинкиТоваров();

        ФайлыКартинок = Новый Массив;
        Для Каждого ФайлКартинки Из ОписаниеФайловКартинок Цикл
                ФайлыКартинок.Добавить(Новый ОписаниеПередаваемогоФайла(
                                ФайлКартинки.ИмяФайла, ФайлКартинки.АдресСсылки));
        КонецЦикла;

        ПараметрыДиалога = Новый ПараметрыДиалогаПолученияФайлов(
                                "Выберите каталог для сохранения файлов картинок", Истина);
        ПараметрыАрхива = Новый ПараметрыПолученияАрхиваФайлов(
                                "pict.zip", РежимПолученияАрхиваФайлов.ПолучатьАрхивВсегда);

        НачатьПолучениеФайловССервера(ФайлыКартинок, ПараметрыДиалога);
        //НачатьПолучениеФайловССервера(, ФайлыКартинок, "e:\test\");
        //НачатьПолучениеФайловССервера(ФайлыКартинок, ПараметрыДиалога, ПараметрыАрхива);

КонецПроцедуры
```

В этом обработчике сначала с помощью функции ПолучитьКартинкиТоваров() мы получаем массив ОписаниеФайловКартинок, описывающий файлы картинок, которые должны быть получены с сервера (листинг 6.27).

Каждый файл описывается структурой с полями ИмяФайла и АдресСсылки. На каждом шаге цикла обхода элементов массива ОписаниеФайловКартинок мы создаем объект ОписаниеПередаваемогоФайла на основе значений полей этой структуры. Таким образом мы создаем массив ФайлыКартинок, описывающий файлы, которые мы будем получать на клиенте по адресу навигационной ссылки на реквизит товара и сохранять в локальную файловую систему с именем, заданным в описании файла.

Затем мы создаем объект ПараметрыДиалогаПолученияФайлов и задаем заголовок и признак отображения диалога.

И передаем массив получаемых файлов и параметры диалога получения файлов в метод глобального контекста НачатьПолучениеФайловССервера(), с помощью которого начинается получение файлов с картинками товаров и сохранение их в локальную файловую систему. При этом пользователю показывается диалог выбора каталога для сохранения файлов.

Можно также использовать вариант вызова этого метода с указанием в третьем параметре базового каталога расположения файлов вместо диалога выбора (см. закомментированный фрагмент кода). Но следует иметь в виду, что для работы этого метода в веб-клиенте потребуется установить расширение для работы с файлами. В то время как в первом случае это расширение не требуется.

Листинг 6.27. Функция «ПолучитьКартинкиТоваров()»

```
&НаСервереБезКонтекста
Функция ПолучитьКартинкиТоваров()

    ССылкиКартинок = Новый Массив;
    СписокТоваров = Справочники.Товары.Выбрать();

    Пока СписокТоваров.Следующий() Цикл
        Если Не СписокТоваров.ЭтоГруппа И Не ПустаяСтрока(СписокТоваров.ИмяФайлаКартинки) Тогда
          ДанныеКартинки = Новый Структура("АдресСсылки, ИмяФайла",);

            АдресКартинкиВХранилище = ПолучитьНавигационнуюССылку(
                                СписокТоваров.Ссылка, "ДанныеФайлаКартинки");
            ДанныеКартинки.АдресСсылки = АдресКартинкиВХранилище;
            ИмяФайлаКартинки = СписокТоваров.ИмяФайлаКартинки;
            ДанныеКартинки.ИмяФайла = ИмяФайлаКартинки;

            ССылкиКартинок.Добавить(ДанныеКартинки);
        КонецЕсли;
```

```
    КонецЦикла;

    Возврат СсылкиКартинок;

КонецФункции
```

В этой функции мы обходим выборку из элементов справочника товаров. На каждом шаге цикла мы создаем структуру `ДанныеКартинки` с полями `ИмяФайла` и `АдресСсылки` и заполняем их соответственно значением реквизита ИмяФайлаКартинки и навигационной ссылкой на реквизит ДанныеФайлаКартинки каждого товара. И затем добавляем эту структуру в массив `СсылкиКартинок`, который и возвращает функция после завершения обхода выборки товаров.

Если требуется получить файлы в виде архива, то в метод `НачатьПомещениеФайловНаСервер()` последним параметром нужно передать объект `ПараметрыПолученияАрхиваФайлов` (см. закомментированный фрагмент кода в листинге 6.26).

Текстовые файлы

Работа с текстовыми файлами в «1С:Предприятии» может осуществляться в контексте двух моделей:

- работа с текстовым документом, который полностью загружается в память (объект `ТекстовыйДокумент`);
- работа в модели последовательного доступа (объекты `ЗаписьТекста`, `ЧтениеТекста`).

В первом случае содержимое текстового файла полностью загружается в память, и если файл большой, то оперативной памяти может не хватить. При использовании модели последовательного доступа как при чтении, так и при записи обрабатывается определенный фрагмент текста, поэтому можно обрабатывать файлы любого размера. Доступ к таким фрагментам осуществляется последовательно.

Прежде чем приступить к реализации обмена посредством текстовых файлов, сторонам (между которыми будет производиться обмен) необходимо «договориться» о логической структуре этого файла:

- вариант использования текста: с фиксированной длиной полей, с переменной длиной полей;
- если выбран вариант с переменной длиной полей, то какой используется разделитель;
- порядок следования данных;

- возможная структура текстового документа и т. д.

Предположим, что в результате были достигнуты следующие договоренности.

В первой строке текстового документа указывается наименование организации – отправителя данных. А затем, через тире от наименования, указывается дата отправки в формате «ДД.ММ.ГГГГ», т. е. первые две цифры – это день месяца, далее две цифры − это номер месяца, далее четыре цифры − номер года.

В каждой следующей строке выгружается элемент справочника Сотрудники. Последовательно производится выгрузка кода, наименования, даты рождения и количества детей сотрудника. Выгружаемые значения разделяются заранее оговоренным символом разделителя (например, запятой).

Пример файла, который будет использоваться при обмене, приведен в листинге 6.28.

Листинг 6.28. Пример файла обмена

```
ООО Быстрее, выше, сильнее-28.01.2020
000000001,Алексеев Сергей Иванович,10.12.1980,1
REST-0003,Артемов Игорь Владимирович,17.05.2019,2
000000002,Смирнова Светлана Ивановна,22.02.1990,0
```

Только после фиксирования подобных договоренностей можно приступать к реализации механизмов выгрузки и загрузки.

Текстовый документ, поле текстового документа

Для иллюстрации возможностей работы с текстовым файлом (с использованием объекта `ТекстовыйДокумент` и поля формы вида `ПолеТекстовогоДокумента`) добавим в нашу демонстрационную обработку реквизит ТД типа `ТекстовыйДокумент` и реквизит СтрокаРазделителя, в котором будет храниться разделитель между значениями текстового файла. Перетащим их в окно элементов формы.

Затем добавим команду ЗаписатьДанныеСотрудников. Обработчик команды заполним следующим образом (листинг 6.29).

Листинг 6.29. Обработчик команды «ЗаписатьДанныеСотрудников»

```
&НаКлиенте
Процедура ЗаписатьДанныеСотрудников(Команда)

    НовыйТД = Новый ТекстовыйДокумент;
    // Установить кодировку документа UTF8. Это кодировка по умолчанию.
    НовыйТД.УстановитьТипФайла(КодировкаТекста.UTF8);
```

```
    Если ПустаяСтрока(СтрокаРазделителя) Тогда
        СтрокаРазделителя = ",";
    КонецЕсли;
    Текст = ПолучитьСтрокиСотрудников(СтрокаРазделителя);
    Текст = СформироватьЗаголовок("ООО Быстрее, выше, сильнее") + Символы.ПС + Текст;
    НовыйТД.УстановитьТекст(Текст);

    НовыйТД.НачатьЗапись(, "c:\temp\doc_text.txt");

КонецПроцедуры
```

В этом обработчике мы создаем объект `ТекстовыйДокумент` и устанавливаем тип кодировки этого документа.

Затем вызываем функцию `ПолучитьСтрокиСотрудников()`, в которую передаем строку разделителя значений (см. листинг 6.30). Эта функция выполняется на сервере. В ней в цикле обхода выборки элементов справочника Сотрудники мы формируем строки с данными сотрудников и в виде одной большой строки возвращаем в процедуру `ЗаписатьДанныеСотрудников()`. Для объединения данных мы используем функцию глобального контекста `СтрСоединить()`.

После этого в полученную строку мы добавляем заголовок файла обмена. Для получения заголовка используется функция `СформироватьЗаголовок()`, см. листинг 6.31. Затем методом `УстановитьТекст()` объекта `ТекстовыйДокумент` мы устанавливаем эту строку в виде текста в текстовый документ и записываем документ в текстовый файл методом `НачатьЗапись()`.

Функции `СформироватьЗаголовок()` и `ПолучитьСтрокиСотрудников()` реализуют достигнутые при переговорах «договоренности» о формате файла обмена.

Листинг 6.30. Функция «ПолучитьСтрокиСотрудников()»

```
&НаСервере
Функция ПолучитьСтрокиСотрудников(Разделитель)

    СтрокаСотрудника = Новый Массив();
    МассивСтрокСотрудников = Новый Массив();

    Выборка = Справочники.Сотрудники.Выбрать();
    Пока Выборка.Следующий() Цикл
        СтрокаСотрудника.Очистить();
        СтрокаСотрудника.Добавить(Выборка.Код);
        СтрокаСотрудника.Добавить(Выборка.Наименование);
        СтрокаСотрудника.Добавить(Формат(Выборка.ДатаРождения, "ДЛФ=D"));
        СтрокаСотрудника.Добавить(Выборка.КоличествоДетей);

        Данные = СтрСоединить(СтрокаСотрудника, Разделитель);
```

```
        МассивСтрокСотрудников.Добавить(Данные);
    КонецЦикла;

    Возврат СтрСоединить(МассивСтрокСотрудников, Символы.ПС);

КонецФункции
```

Листинг 6.31. Функция «СформироватьЗаголовок()»

```
&НаКлиенте
Функция СформироватьЗаголовок(НаименованиеКомпании)

    Возврат СокрЛП(НаименованиеКомпании) + "-" + Формат(ТекущаяДата(), "ДЛФ=D");

КонецФункции
```

Для чтения данных из полученного файла добавим команду ПрочитатьДанныеСотрудников. Обработчик команды заполним следующим образом (листинг 6.32).

Листинг 6.32. Обработчик команды «ПрочитатьДанныеСотрудников»

```
Процедура ПрочитатьДанныеСотрудников(Команда)

    ТД.НачатьЧтение(Новый ОписаниеОповещения(
                        "ПрочитатьДанныеЗавершение", ЭтотОбъект), "c:\temp\doc_text.txt");

КонецПроцедуры
```

В этом обработчике мы начинаем чтение из записанного ранее текстового файла методом `НачатьЧтение()` объекта `ТекстовыйДокумент`. В результате содержимое текстового файла полностью загружается в реквизит ТД, находящийся в нашей демонстрационной обработке. В метод `НачатьЧтение()` первым параметром мы передаем имя процедуры-обработчика оповещения `ПрочитатьДанныеЗавершение()`, которая будет выполнена, после того как текстовый файл будет прочитан (листинг 6.33).

Листинг 6.33. Процедура «ПрочитатьДанныеЗавершение()»

```
&НаКлиенте
Процедура ПрочитатьДанныеЗавершение(Дополнительно) Экспорт

    Сообщение = Новый СообщениеПользователю();

    ЗаголовокФайла = ТД.ПолучитьСтроку(1);
    СтрокаЗаголовка = СтрРазделить(ЗаголовокФайла, "-");
    НазваниеОтправителя = СтрокаЗаголовка[0];
    Сообщение.Текст = НазваниеОтправителя;
    Сообщение.Сообщить();
    ДатаТекстом = СтрокаЗаголовка[1];
```

```
    Сообщение.Текст = ДатаТекстом;
    Сообщение.Сообщить();

    Для ТекущаяСтрока = 2 По ТД.КоличествоСтрок() Цикл
        ПрочитаннаяСтрока = ТД.ПолучитьСтроку(ТекущаяСтрока);
        СтрокиСотрудников = СтрРазделить(ПрочитаннаяСтрока, СтрокаРазделителя);

        Для Каждого Строка Из СтрокиСотрудников Цикл
            Если НЕ ПустаяСтрока(Строка) Тогда
                Сообщение.Текст = Строка;
                Сообщение.Сообщить();
            КонецЕсли;
        КонецЦикла;

    КонецЦикла;

КонецПроцедуры
```

Содержимое прочитанного файла мы получаем с помощью метода `ПолучитьСтроку()` текстового документа. Строку заголовка, согласно договоренности, мы получаем в первой строке документа. Остальные строки с данными сотрудников получаем в цикле обхода строк текстового документа.

Для разбора строк мы используем функцию глобального контекста `СтрРазделить()`.

Отображение текстового документа

Для чтения и отображения в форме обработки содержимого текстового документа добавим команду ПросмотрТД. Обработчик команды заполним следующим образом (листинг 6.34).

Листинг 6.34. Обработчик команды «ПросмотрТД»

```
&НаКлиенте
Процедура ПросмотрТД(Команда)

    ТД.Очистить();
    ТД.НачатьЧтение(Новый ОписаниеОповещения(
                                "ПросмотрТДЗавершение", ЭтотОбъект), "c:\temp\doc_text.txt");

КонецПроцедуры
```

В этом обработчике мы начинаем чтение из записанного ранее текстового файла методом `НачатьЧтение()` объекта `ТекстовыйДокумент`. В результате содержимое текстового файла полностью загружается в реквизит ТД, находящийся в нашей демонстрационной обработке. В метод

НачатьЧтение() первым параметром мы передаем имя процедуры-обработчика оповещения ПросмотрТДЗавершение(), которая будет выполнена, после того как текстовый файл будет прочитан (листинг 6.35).

Листинг 6.35. Процедура «ПросмотрТДЗавершение()»

```
&НаКлиенте
Процедура ПросмотрТДЗавершение(Дополнительно) Экспорт

	ТД.ВставитьСтроку(1, "Показывается в оригинальном виде");

КонецПроцедуры
```

Перед выводом текстового документа реквизит ПолеТД типа ТекстовыйДокумент очищается, после чтения в качестве первой вставляется строка «Показывается в оригинальном виде» (она «сдвигает» ранее загруженные строки).

Пользователь может производить любые корректировки текста, представленного в поле формы, отражающем текстовый документ. Чтобы записать результат работы в файл, добавим команду ЗаписьТД. Обработчик команды заполним следующим образом (листинг 6.36).

Листинг 6.36. Обработчик команды «ЗаписьТД»

```
&НаКлиенте
Процедура ЗаписьТД(Команда)

	ТД.НачатьЗапись(Новый ОписаниеОповещения(
				"ЗаписьТДЗавершение", ЭтотОбъект), "c:\temp\doc_text2.txt");

КонецПроцедуры
```

В этом обработчике методом НачатьЗапись() объекта ТекстовыйДокумент мы начинаем запись содержимого текстового документа, отображаемого в форме, в указанный вторым параметром текстовый файл. Первым параметром в этот метод мы передаем имя процедуры-обработчика оповещения ЗаписьТДЗавершение(), которая будет выполнена, после того как текстовый файл будет записан (листинг 6.37).

Листинг 6.37. Процедура «ЗаписьТДЗавершение()»

```
&НаКлиенте
Процедура ЗаписьТДЗавершение(Результат, Дополнительно) Экспорт

	Если Результат Тогда
		ТД.Очистить();
	КонецЕсли;

КонецПроцедуры
```

После записи реквизит, содержащий данные текстового документа, очищается. Этого можно и не делать, в этом случае обработчик оповещения в методе `НачатьЗапись()` можно не указывать.

При работе с текстовыми файлами в «1С:Предприятии» следует учитывать особенности представления разделителей строк в файлах Windows и в тексте, используемом для обработки внутри «1С:Предприятия». Для целей внутренней обработки текста используется разделитель, состоящий из одного символа, в то время как в файлах Windows в качестве разделителя строк используется комбинация из двух последовательных символов.

Внутри «1С:Предприятия» разделителем строк является символ `Символы.ПС`, он же LF (Line Feed, Перевод Строки), который имеет шестнадцатеричный код `0A`. В то же время в кодировке текстовых файлов для Windows принято, что разделителем строк является пара символов – «CR-LF». CR (Carriage Return, Возврат Каретки) имеет шестнадцатеричный код `0D`.

При записи и чтении текстовых файлов «1С:Предприятие» производит преобразование внутренних разделителей строк в представление, принятое в текстовых файлах Windows. То есть при записи символ «ПС» (LF) преобразуется в пару символов «ВК-ПС» («CR-LF»). При чтении происходит обратное преобразование, то есть пара «ВК-ПС» («CR-LF») преобразуется в одиночный символ ПС (LF).

Символ «ВК» («CR») при записи файла не преобразуется и не выбрасывается, то есть попадает в файл как есть. При чтении файла символы «ВК» («CR») и «ПС» («LF»), не образующие пару, также считываются без преобразований.

Модель последовательного доступа

В ранее рассмотренном примере работы с текстовыми файлами содержимое файла загружалось в оперативную память полностью. При работе с файлами большого размера такой подход может привести к замедлению обработки данных из файла. В этом случае можно воспользоваться возможностью «1С:Предприятия» работать с текстовыми файлами в рамках модели последовательного доступа.

Данная модель реализована объектами `ЗаписьТекста` и `ЧтениеТекста`.

Для записи данных добавим команду ПоследовательнаяЗапись. Обработчик команды заполним следующим образом (листинг 6.38).

Листинг 6.38. Обработчик команды «ПоследовательнаяЗапись»

```
&НаКлиенте
Процедура ПоследовательнаяЗапись(Команда)

    ЗаписьТекста = Новый ЗаписьТекста("c:\temp\doc_text3.txt", КодировкаТекста.UTF8);

    ЗаписьТекста.ЗаписатьСтроку(СформироватьЗаголовок("Наименование компании"));

    Если ПустаяСтрока(СтрокаРазделителя) Тогда
        СтрокаРазделителя = ";";
    КонецЕсли;
    Текст = ПолучитьСтрокиСотрудников(СтрокаРазделителя);
    ЗаписьТекста.ЗаписатьСтроку(Текст);

    ЗаписьТекста.Закрыть();

КонецПроцедуры
```

В этом обработчике мы создаем объект `ЗаписьТекста` и с помощью метода `ЗаписатьСтроку()` построчно записываем заголовок документа и данные сотрудников в файл. Здесь, так же как и в случае с объектом `ТекстовыйДокумент`, строки элементов справочника Сотрудники мы получаем с сервера в виде одной большой строки функцией `ПолучитьСтрокиСотрудников()` и затем записываем на клиенте методом `ЗаписатьСтроку()` объекта `ЗаписьТекста`. Необходимо заметить, что в случае записи файлов большого размера строку для записи нужно разбивать на небольшие порции (например, ограничить размер записываемой строки 1000 элементами справочника).

Тексты функций `СформироватьЗаголовок()` и `ПолучитьСтрокиСотрудников()` приводились ранее (см. листинги 6.31 и 6.30).

Для чтения данных добавим команду ПоследовательноеЧтение. Обработчик команды заполним следующим образом (листинг 6.39).

Листинг 6.39. Обработчик команды «ПоследовательноеЧтение»

```
&НаКлиенте
Процедура ПоследовательноеЧтение(Команда)

    Сообщение = Новый СообщениеПользователю();
    Текст = Новый ЧтениеТекста("c:\temp\doc_text3.txt", КодировкаТекста.UTF8);

    // Прочитать первую строку с заголовком.
    ПрочитаннаяСтрока = Текст.ПрочитатьСтроку();
    СтрокаЗаголовка = СтрРазделить(ПрочитаннаяСтрока, "-");
    НазваниеОтправителя = СтрокаЗаголовка[0];
    Сообщение.Текст = НазваниеОтправителя;
    Сообщение.Сообщить();
    ДатаТекстом = СтрокаЗаголовка[1];
    Сообщение.Текст = ДатаТекстом;
```

```
    Сообщение.Сообщить();

    // Прочитать строки в цикле пока не будет достигнут конец файла.
    Пока ПрочитаннаяСтрока <> Неопределено Цикл
        ПрочитаннаяСтрока = Текст.ПрочитатьСтроку();
        СтрокиСотрудников = СтрРазделить(ПрочитаннаяСтрока, СтрокаРазделителя);

        Для Каждого Строка Из СтрокиСотрудников Цикл
            Если НЕ ПустаяСтрока(Строка) Тогда
                Сообщение.Текст = Строка;
                Сообщение.Сообщить();
            КонецЕсли;
        КонецЦикла;

    КонецЦикла;

КонецПроцедуры
```

В этом обработчике мы создаем объект `ЧтениеТекста` и с помощью метода `ПрочитатьСтроку()`построчно считываем содержимое записанного ранее текстового файла. Сначала мы считываем строку заголовка. Остальные строки с данными сотрудников получаем последовательно по мере обхода содержимого текстового файла.

Для разбора строк мы используем функцию глобального контекста `СтрРазделить()`.

Следует отметить, что чтение данных может осуществляться не только построчно, но и по указанному количеству символов (метод `Прочитать()` объекта `ЧтениеТекста`).

XML-файлы

С течением времени, а также с развитием программных систем в качестве универсального формата файлов обмена был принят обмен посредством XML-документов, сформированных на языке XML (расширяемый язык разметки).

Основные положения

XML-документ в общем случае представляет собой файл с расширением XML, имеющий текстовое наполнение. Исходя из этого можно сказать, что самым простейшим редактором таких документов может служить Блокнот (Notepad), хотя существует довольно большое количество специализированных редакторов.

XML-документы по сравнению с обычными текстовыми документами имеют следующие отличительные черты:

- XML-документ создается по строгим правилам, внутри него соблюдается четкая иерархия составных частей, вместе с данными передается структура этих данных;
- описание возможной структуры документа определяется на специальном формализованном языке, вследствие чего эта структура может быть программно разобрана (на приемной стороне можно разобрать возможную структуру, описанную на специальном языке, сопоставить структурные элементы с метаданными системы и после этого произвести загрузку данных);
- при работе с XML-документами используется объектный подход (а не разбор строки с помощью специальных функций работы со строковыми значениями);
- формальное описание типов.

При необходимости произвести обмен данными посредством XML-документов:

- требуется XML-документ с данными (созданный в соответствии со стандартом, определенным ко всем XML-документам);
- к документу должно существовать описание его структуры, которое можно формально разобрать (схема XML-документа);
- должны быть определены пространства имен (в которых в том числе описаны типы данных, используемые в документе).

Следует отметить, что, рассматривая особенности механизмов, позволяющих работать с XML-документами из «1С:Предприятия», мы в дальнейшем не будем подробно касаться таких понятий, как пространство имен (описанное в нем пространство типов), схема XML-документа. Но необходимо отдавать себе отчет в том, что при организации обмена с произвольными информационными системами понадобится более детальная работа с ними.

Перейдем к рассмотрению структуры и общих принципов формирования XML-документов.

При рассмотрении внутреннего наполнения XML-документов в них можно выделить определенные форматирующие конструкции. По аналогии с HTML их можно назвать тегами. Под тегом будем понимать некое выражение, заключенное в угловые скобки <выражение>.

Различают открывающий тег <тег> и закрывающий тег </тег>. В отличие от языка HTML, в языке XML имена тегов не фиксированы, разработчик волен давать им собственные имена, определять у них произвольные

свойства с любыми типами значений. В имени тега XML могут использоваться точки (правда, это может не поддерживаться рядом специализированных редакторов). В содержимом атрибутов, элементов пробелы и символы табуляции не игнорируются.

Перейдем от термина «тег» к терминам более правильным по отношению к XML-документу. Можно сказать, что минимальной логической единицей XML-документа является *элемент*. У элемента может быть определено несколько атрибутов (упрощенно атрибут можно назвать свойством элемента). У любого элемента может быть один (непосредственный) элемент-владелец (его нет только у единственного корневого элемента XML-документа) и любое количество подчиненных элементов (рис. 6.2).

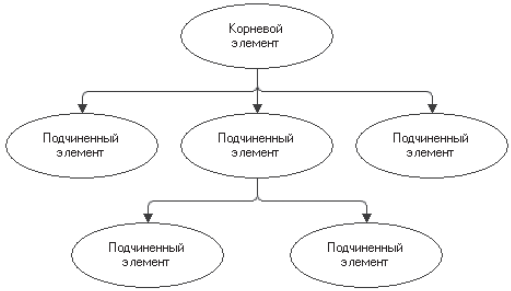


Рис. 6.2. Пример иерархии элементов XML

Следует отметить, что в различной литературе используемая терминология может различаться. То, что в данной главе называется элементом, может называться узлом. В используемой нами терминологии понятие «узел» также встречается, но под ним понимается структурная часть элемента XML (такая как начало элемента, текст и т. д.).

Рассмотрим структуру элемента XML (рис. 6.3).

Рис. 6.3. Схема элемента XML

Пример, иллюстрирующий приведенную схему, приведен в листинге 6.40.

Листинг 6.40. Пример элемента XML

```
<Товар ИмяСправочника = «Номенклатура» Код = «14»>Ardo TL 1000 EX-1</Товар>
```

В данной строке определен элемент с именем Товар. В начале элемента определяются два атрибута с именами ИмяСправочника и Код (соответственно со значениями Номенклатура и 14). Между началом элемента и его концом (</Товар>) расположен текст элемента (его значение Ardo TL 1000 EX-1). Вместо текста между тегами, описывающими начало и конец элемента, может находиться описание вложенных элементов.

Также допускается следующий вариант определения элемента XML-документа (рис. 6.4).

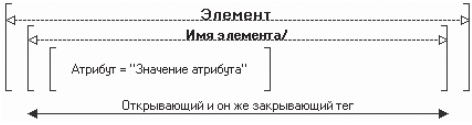


Рис. 6.4. Схема элемента XML

Приведенная схема представляет следующий вариант описания элемента (листинг 6.41).

Листинг 6.41. Пример элемента XML

```
<Товар Код = «14» Наименование = «Ardo TL 1000 EX-1»/>
```

Данный вариант допустим, когда между открывающим и закрывающим тегом нет текста и нет вложенных элементов. Поэтому вместо <Товар></Товар> допустима конструкция <Товар/>. В приведенном описании определяются элемент с именем Товар и два атрибута с именами Код и Наименование.

Рассмотрим пример XML-документа (листинг 6.42).

Листинг 6.42. Пример XML-документа

```
<?xml version="1.0" encoding="UTF-8" ?>
<Корневой>
<DocumentObject.ПриходнаяНакладная>
        <Ref>4c0a4c73-905d-11d8-92a4-00030d0ca645</Ref>
        <DeletionMark>false</DeletionMark>
        <Date>2020-01-17T14:52:43</Date>
        <Number>0000000001</Number>
        <Posted>true</Posted>
        <Контрагент>4c0a4c72-905d-11d8-92a4-00030d0ca645</Контрагент>
        <Товары>
                <Row>
                        <Номенклатура>efbfeaf4-831d-11d8-957e-00c026abdd5e</Номенклатура>
```

```
                <Количество>10</Количество>
                <Цена>900</Цена>
                <Сумма>1000</Сумма>
            </Row>
            <Row>
                <Номенклатура>efbfeaf6-831d-11d8-957e-00c026abdd5e</Номенклатура>
                <Количество>10</Количество>
                <Цена>1000</Цена>
                <Сумма>10000</Сумма>
            </Row>
        </Товары>
</DocumentObject.ПриходнаяНакладная>
</Корневой>
```

В первой строке XML-документа определяется директива, указывающая на то, что данный документ с текстовым содержимым является XML-документом. Далее определяется элемент Корневой. Внутри него определяются другие элементы. Можно сказать, что в данном документе просматривается определенная древовидная структура (она показана на рис. 6.5).

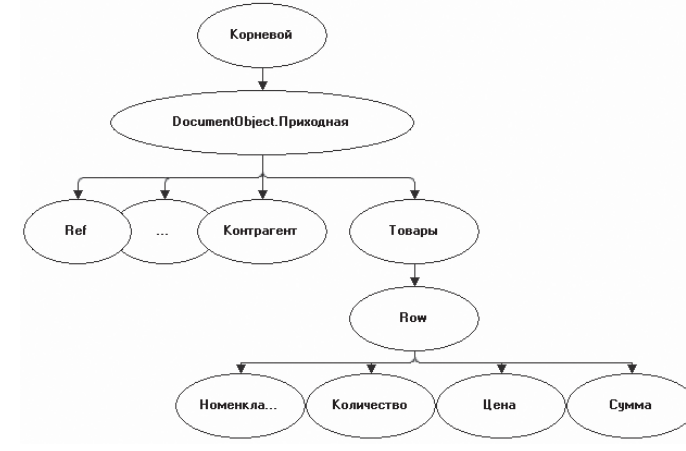


Рис. 6.5. Древовидная структура XML-документа

Эта структура обеспечивается тем, что у XML-документа всегда один корневой элемент, описание одного элемента всегда находится внутри описания другого элемента (исключение составляет корневой элемент). Можно сказать, что древовидность структуры является обязательным признаком правильности оформленного XML-документа.

При рассмотрении любого XML-документа можно говорить о его синтаксической правильности и корректности.

Синтаксически правильный XML-документ, упрощенно, удовлетворяет следующим условиям:

- Язык XML чувствителен к регистру символов. Это обязательно нужно учитывать при описании начала и конца элемента.
- Все значения атрибутов, используемых в определении элементов, должны быть заключены в кавычки.
- Каждому открывающему тегу соответствует закрывающий. Можно сказать, что XML-документ является совокупностью элементов. Два возможных варианта описания элементов были представлены выше.
- Элементы образуют древовидную структуру с одним корневым элементом (он обязательно один). Это достигается тем, что области действия тегов не могут пересекаться (либо они рядом, либо одна находится целиком внутри другой).

Базовые средства «1С:Предприятия» для работы с XML

В «1С:Предприятии» существует базовая подсистема работы с XML-документами. Основой этой подсистемы являются два объекта «1С:Предприятия»:

- `ЧтениеXML`,
- `ЗаписьXML`.

При работе с XML-документами данные объекты реализуют модель последовательного доступа. Основная особенность работы в этой модели заключается в том, что документ не загружается полностью, работа идет только с текущим его фрагментом (частью элемента XML, например началом элемента, текстом элемента и т. д.). Поэтому при работе с документом нет возможности получать выборки элементов, «перепрыгивать» через узлы и т. п.

Запись данных в XML-документ

Вернемся к задаче обмена данными об элементах справочника Сотрудники. Рассмотрим пример использования объекта `ЗаписьXML` (листинг 6.43).

Листинг 6.43. Пример записи XML-документа

```
&НаКлиенте
Процедура ЗаписьДанных(Команда)

    СтрокаРазделителя = "*";

    ЗаписьXML = Новый ЗаписьXML;
    ЗаписьXML.ОткрытьФайл("c:\temp\document.xml");
```

```
    // Записать директиву.
    ЗаписьXML.ЗаписатьОбъявлениеXML();

    // Записать начало корневого элемента.
    ЗаписьXML.ЗаписатьНачалоЭлемента("Корневой");

    // Записать атрибут корневого элемента.
    ЗаписьXML.ЗаписатьАтрибут("ИмяСправочника", "Сотрудники");

    ЗаписьXML.ЗаписатьКомментарий("Выгрузка элементов справочника");

    // Получить данные сотрудников в виде одной большой строки.
    СтрокаСотрудников = ПолучитьСтрокиСотрудников(СтрокаРазделителя);
    // Получить массив строк дла каждого сотрудника.
    СтрокиСотрудников = СтрРазделить(СтрокаСотрудников, Символы.ПС);

    Для ТекущаяСтрока = 0 По СтрокиСотрудников.Количество() - 1 Цикл
        // Получить данные каждого сотрудника.
        Данные = СтрРазделить(СтрокиСотрудников[ТекущаяСтрока], СтрокаРазделителя);

        ЗаписьXML.ЗаписатьНачалоЭлемента("ЭлементСправочника");

        ЗаписьXML.ЗаписатьНачалоЭлемента("Код");
        ЗаписьXML.ЗаписатьТекст(Данные[0]);
        ЗаписьXML.ЗаписатьКонецЭлемента();

        ЗаписьXML.ЗаписатьНачалоЭлемента("Наименование");
        ЗаписьXML.ЗаписатьТекст(Данные[1]);
        ЗаписьXML.ЗаписатьКонецЭлемента();

        ЗаписьXML.ЗаписатьНачалоЭлемента("ДатаРождения");
        ЗаписьXML.ЗаписатьТекст(Данные[2]);
        ЗаписьXML.ЗаписатьКонецЭлемента();

        ЗаписьXML.ЗаписатьНачалоЭлемента("КоличествоДетей");
        ЗаписьXML.ЗаписатьТекст(Данные[3]);
        ЗаписьXML.ЗаписатьКонецЭлемента();

        ЗаписьXML.ЗаписатьКонецЭлемента();
    КонецЦикла;

    ЗаписьXML.ЗаписатьКонецЭлемента();
    ЗаписьXML.Закрыть();

КонецПроцедуры
```

Здесь, так же как и при записи текстового файла, вызывается серверная функция `ПолучитьСтрокиСотрудников()`, в которой в цикле обхода выборки элементов справочника Сотрудники формируются строки с данными сотрудников и в виде одной большой строки возвращаются на клиент (см. листинг 6.30). Затем мы производим обратную операцию, получаем массив строк с данными для каждого сотрудника и разбираем строки

с данными с помощью функции глобального контекста `СтрРазделить()`. Полученные элементы массива с данными каждого сотрудника мы записываем в виде элементов XML-файла.

В результате выполнения представленного кода будет получен следующий XML-документ (листинг 6.44).

Листинг 6.44. Полученный XML-документ

```
<?xml version="1.0" encoding="UTF-8"?>
<Корневой ИмяСправочника="Сотрудники">
	<!--Выгрузка элементов справочника-->
	<ЭлементСправочника>
		<Код>000000001</Код>
		<Наименование>Алексеев Сергей Иванович</Наименование>
		<ДатаРождения>10.12.1980</ДатаРождения>
		<КоличествоДетей>1</КоличествоДетей>
	</ЭлементСправочника>
	<ЭлементСправочника>
		<Код>REST-0003</Код>
		<Наименование>Артемов Игорь Владимирович</Наименование>
		<ДатаРождения>17.05.2019</ДатаРождения>
		<КоличествоДетей>2</КоличествоДетей>
	</ЭлементСправочника>
	<ЭлементСправочника>
		<Код>000000002</Код>
		<Наименование>Смирнова Светлана Ивановна</Наименование>
		<ДатаРождения>22.02.1990</ДатаРождения>
		<КоличествоДетей>0</КоличествоДетей>
	</ЭлементСправочника>
</Корневой>
```

Сначала создается объект `ЗаписьXML`. Через созданный объект в модели последовательного доступа будет производиться запись данных в XML-документ. С помощью метода `ОткрытьФайл()` указывается имя файла, куда будет производиться запись данных. Метод `ЗаписатьОбъявлениеXML()` определяет директиву, что создаваемый документ будет являться XML-документом (директива размещается в первой строке созданного документа).

Для записи элемента XML-документа используются методы `ЗаписатьНачалоЭлемента()`, `ЗаписатьАтрибут()`, `ЗаписатьТекст()`, `ЗаписатьКонецЭлемента()`, которые производят запись соответствующего узла элемента XML в файл. При этом важно соблюдать порядок (иерархию) вызова этих методов.

В заключение методом `Закрыть()` объекта `ЗаписьXML` завершается запись, и файл с данными закрывается.

Напомним, что элемент можно представить в виде следующих узлов (листинг 6.45).

Листинг 6.45. Узлы элемента XML

```
<ИМЯ АТРИБУТ=Значение>
Содержимое (Текст или описание другого элемента)
</ИМЯ>
```

Для записи вышеприведенного элемента можно использовать следующую последовательность методов (листинг 6.46).

Листинг 6.46. Пример записи элемента XML

```
ЗаписьXML.ЗаписатьНачалоЭлемента("ИМЯ");
ЗаписьXML.ЗаписатьАтрибут("АТРИБУТ", "Значение");
ЗаписьXML.ЗаписатьТекст("Содержимое (Текст или описание другого элемента)");
ЗаписьXML.ЗаписатьКонецЭлемента();
```

Следует обратить внимание на тот факт, что если при определении начала элемента в соответствующем методе указывается его имя, то при закрытии имя не указывается. Закрывается элемент, открытый последним. В связи с этим следует внимательно относиться к последовательности исполняемых методов, чтобы не запутаться в иерархии (в ряде случаев для упрощения алгоритмов создания документов сложной структуры могут использоваться рекурсивные вызовы процедур, ответственных за создание элементов).

Последовательность методов объекта `ЗаписьXML`, вызываемых при создании элементов, можно проиллюстрировать следующей схемой (рис. 6.6).

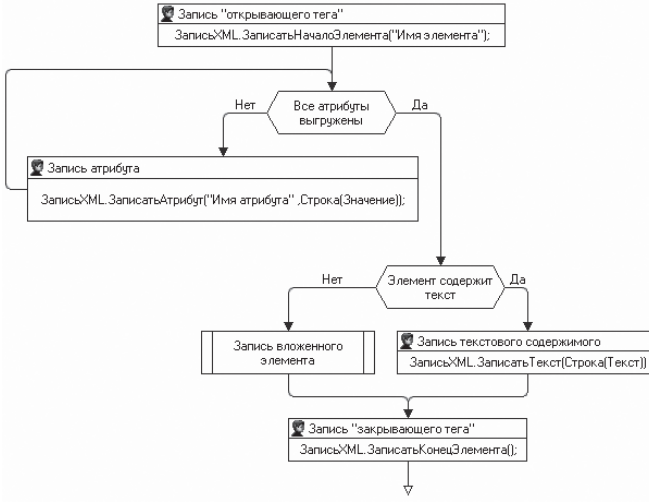


Рис. 6.6. Последовательность использования методов при записи XML-документа

Правильный XML-документ содержит единственный корневой элемент, запись всего документа начинается с записи этого элемента.

Запись любого элемента XML начинается с записи открывающего тега (начала элемента, выполняемого с помощью метода `ЗаписатьНачалоЭлемента()`). Только в контексте начала элемента можно записывать атрибуты XML-документа (используется метод `ЗаписатьАтрибут()`, вызываемый последовательно для каждого атрибута). Записать атрибут после записи текста или конца элемента нельзя.

После записи начала элемента и всех его атрибутов можно провести запись текста (это означает, что у формируемого элемента XML нет вложенных элементов) или рекурсивно перейти к записи другого вложенного элемента.

После того как либо текст, либо другой вложенный элемент (элементы) закрыт, производится запись конца элемента (метод `ЗаписатьКонецЭлемента()`).

Чтение данных из XML-документа

Объект `ЧтениеXML` предназначен для чтения данных из XML-документа. Так же как при записи данных, чтение элемента производится узлами. Перечень узлов определен в системном перечислении `ТипУзлаXML`:

- `НачалоЭлемента`,
- `Атрибут`,
- `Текст`,
- `КонецЭлемента`,
- `Ничего`,
- `ИнструкцияОбработки`.

Какой узел получен при чтении следующего структурного элемента, можно узнать, обратившись к свойству `ТипУзла` объекта `ЧтениеXML`.

Будем исходить из того, что необходимо прочитать следующий XML-документ (полученный при выгрузке с помощью объекта `ЗаписьXML`) – листинг 6.47.

Листинг 6.47. Читаемый XML-документ

```
<?xml version="1.0" encoding="UTF-8"?>
<Корневой ИмяСправочника="Сотрудники">
        <!--Выгрузка элементов справочника-->
        <ЭлементСправочника>
                <Код>000000001</Код>
                <Наименование>Алексеев Сергей Иванович</Наименование>
                <ДатаРождения>10.12.1980</ДатаРождения>
```

```
        <КоличествоДетей>1</КоличествоДетей>
    </ЭлементСправочника>
    <ЭлементСправочника>
        <Код>REST-0003</Код>
        <Наименование>Артемов Игорь Владимирович</Наименование>
        <ДатаРождения>17.05.2019</ДатаРождения>
        <КоличествоДетей>2</КоличествоДетей>
    </ЭлементСправочника>
    <ЭлементСправочника>
        <Код>000000002</Код>
        <Наименование>Смирнова Светлана Ивановна</Наименование>
        <ДатаРождения>22.02.1990</ДатаРождения>
        <КоличествоДетей>0</КоличествоДетей>
    </ЭлементСправочника>
</Корневой>
```

Пример использования объекта `ЧтениеXML` (листинг 6.48).

Листинг 6.48. Чтение XML-документа

```
&НаКлиенте
Процедура ЧтениеДанных(Команда)

    Файл = Новый ЧтениеXML;
    Файл.ОткрытьФайл("c:\temp\document.xml");

    Сообщение = Новый СообщениеПользователю();

    // Прочитать фрагменты элемента.
    Пока Файл.Прочитать() Цикл

        // Проверить тип узла.
        Если Файл.ТипУзла = ТипУзлаXML.НачалоЭлемента Тогда
            Сообщение.Текст = Файл.Имя;
            Сообщение.Сообщить();

            // Прочитать атрибуты.
            Пока Файл.ПрочитатьАтрибут() Цикл
                Сообщение.Текст = "атрибут:" + Файл.Имя + "=" + Файл.Значение;
                Сообщение.Сообщить();
            КонецЦикла;
        КонецЕсли;

        Если Файл.ТипУзла = ТипУзлаXML.Текст Тогда
            Сообщение.Текст = Файл.Значение;
            Сообщение.Сообщить();
        КонецЕсли;

        Если Файл.ТипУзла = ТипУзлаXML.КонецЭлемента Тогда
            Сообщение.Текст = "/" + Файл.Имя;
            Сообщение.Сообщить();
        КонецЕсли;
    КонецЦикла;

    Файл.Закрыть();

КонецПроцедуры
```

Процесс чтения начинается с создания объекта `ЧтениеXML`. С помощью метода `ОткрытьФайл()` производится указание файла, из которого в дальнейшем будет производиться чтение данных.

Само чтение осуществляется с помощью метода `Прочитать()`. Данный метод возвращает значение `Истина`, в случае если получилось позиционироваться на следующем узле элемента XML. Для того чтобы понять, какой узел является текущим, производится обращение к свойству `ТипУзла`. В случае если это `НачалоЭлемента`, можно провести выборку атрибутов (они существуют только в контексте начала элемента).

После того как метод `Прочитать()` возвращает значение `Ложь` (достигнут конец документа), работа с XML-документом завершается вызовом метода `Закрыть()`.

Порядок работы с объектом `ЧтениеXML` можно проиллюстрировать следующей схемой (рис. 6.7).

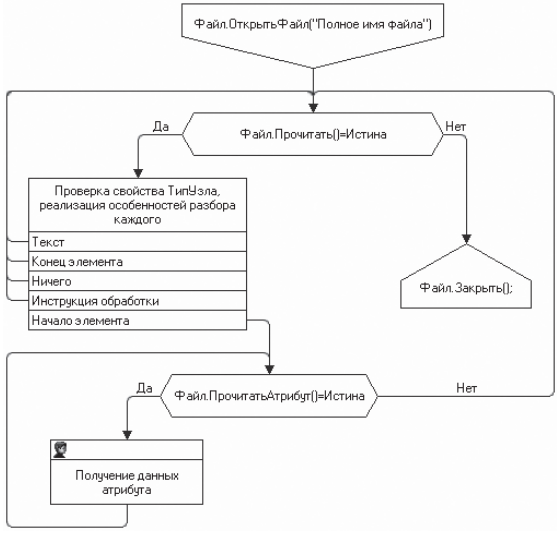


Рис. 6.7. Последовательность использования методов при чтении XML-документа

Общий принцип работы довольно прост. Открывается XML-документ, далее в цикле выполняется метод `Прочитать()`. После чтения анализируется тип полученного узла. В зависимости от типа предпринимаются некие специфические действия по извлечению данных. Если полученный узел является началом элемента, предпринимается попытка получить все атрибуты, которые могут быть в нем определены.

XML-сериализация

Как уже говорилось ранее, XML-документ имеет текстовое содержимое. Это в первую очередь означает, что при записи в него любого значения (числа, даты, ссылки на объект и т. д.) оно преобразуется к строке. При получении данных возникает обратная задача по преобразованию значения из строкового в нужный тип.

С одной стороны, с помощью базовых средств чтения и записи документов XML можно решить любую задачу, связанную с организацией обмена. Но решение задачи преобразования типов в строковый и обратно потребует от разработчика очень больших затрат времени (на самостоятельную реализацию данных преобразований). Придется в полной мере работать со стандартными типами (W3C), определять собственное пространство имен (для описания некоторых типов «1С:Предприятия» в XML) и т. д.

Альтернативным (более простым и удобным) решением данной задачи может быть использование уже реализованных средств XML-сериализации.

С точки зрения данной системы каждый объект данных «1С:Предприятия» представляется как элемент XML-документа. Этот элемент может иметь подчиненные элементы. Как раз с этой точки зрения (есть у элемента подчиненные или нет) типы значений делятся на простые и сложные.

К простым типам данных относятся типы, значения которых представляются подсистемой XML-сериализации в виде элементов XML только с текстовым содержимым (листинг 6.49).

Листинг 6.49. Пример сериализации значения ссылки на справочник

```
<CatalogRef.Номенклатура>
        178a492a-3fd4-11da-a1f2-0011d8388a5d
</CatalogRef.Номенклатура>
```

Значения сложных типов представляются в виде элементов XML, содержащих вложенные элементы (листинг 6.50).

Листинг 6.50. Пример сериализации значения типа «УдалениеОбъекта»

```
<v8:ObjectDeletion xmlns="http://v8.1c.ru/data">
        <v8:Ref xsi:type="CatalogRef.Банки">
                60c5cec3-7f6f-4ec3-9620-e757fe3614ca
        </v8:Ref>
</v8:ObjectDeletion>
```

Если проанализировать типы, которые требуется подвергать преобразованию, то их можно разделить на три группы:

- Типы, которым можно найти прямое соответствие в типах, определенных в документе XML Schema Part 2: Datatypes консорциума W3C (пространство имен – http://www.w3.org/2001/XMLSchema). Значения данных типов могут представляться в виде элемента с текстовым содержимым.
- Предопределенные типы «1С:Предприятия» (пространство имен – http://v8.1c.ru/data, http://v8.1c.ru/8.1/data, http://v8.1c.ru/8.2/data). Эти типы не зависят от структуры метаданных и существуют в любой информационной базе. Исходя из этого, имеет смысл описать их в специализированном пространстве имен. Они могут относиться как к простым типам, так и к сложным.
- Типы, производные от метаданных конфигурации «1С:Предприятия». Наличие таких типов зависит от состава и специфики определения объектов конфигурации. В каждой информационной базе они могут иметь произвольную реализацию. Исходя из этого, нет возможности описать все возможные варианты этих типов в каком-либо пространстве имен (при создании объекта конфигурации разработчик имеет столько степеней свободы, что описать все возможные варианты создаваемых объектов в принципе невозможно). В итоге такие типы не относятся ни к какому пространству имен и тоже могут являться как простыми, так и сложными типами (например, ссылка на объект относится к простым типам, сам объект − к сложным).

При рассмотрении примеров представления различных значений в XML и при дальнейшем изложении будем исходить из предположения, что определены следующие соответствия пространств имен (после xmlns определены их префиксы):

- xmlns:xsd="http://www.w3.org/2001/XMLSchema",
- xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance",
- xmlns:v8="http://v8.1c.ru/data",
- xmlns:v8="http://v8.1c.ru/8.1/data",
- xmlns:v8="http://v8.1c.ru/8.2/data".

К простым типам с точки зрения представления в XML относятся следующие типы «1С:Предприятия»:

- `Число`;
- `Строка`;
- `Дата`;
- `Булево`;

- ДвоичныеДанные;
- Null;
- УникальныйИдентификатор;
- ХранилищеЗначения;
- все ссылки на объекты базы данных;
- ссылки на перечисления, определяемые в конфигурации.

К сложным типам, значения которых могут быть представлены в XML, относятся следующие типы:

- Тип;
- ОписаниеТипов;
- КонстантаМенеджерЗначения.<имя>;
- все объекты базы данных;
- наборы записей регистров, последовательностей, перерасчетов;
- УдалениеОбъекта.

Простые типы

Рассмотрим различные варианты представления значений простых типов в XML-документе.

Число

Типу Число соответствует тип данных XML decimal из пространства имен http://www.w3.org/2001/XMLSchema (префикс xsd).

Правила представления значений данного типа определены в документе XML Schema Part 2: Datatypes. Ниже приведены примеры представления значений типа Число в XML (листинг 6.51).

Листинг 6.51. Варианты представления значения типа «Число» – 3.14156

```
<!-- Не задано явно имя корневого элемента -->
<decimal>3.14156</decimal>

<!-- Явно задано имя корневого элемента XML -->
<Amount>3.14156</Amount>

<!-- Явно указан тип данных XML -->
<Data xsi:type="xsd:decimal">3.14156</Data>
```

В первом случае имя типа (пространство имен http://www.w3.org/2001/XMLSchema) совпадает с именем элемента, в котором передается значение. Во втором случае именем узла является имя переменной (свойства, реквизита), из которого/в который должна производиться запись. В третьем

случае тип определен явно. Используется тот факт, что в пространстве имен с префиксом xsi определена возможность использования такого атрибута, как type. С помощью значения этого атрибута может указываться тип, который был преобразован к строковому и размещен в тексте элемента XML. В нашем случае указано, что при загрузке текст «3.14156» необходимо интерпретировать как тип decimal из пространства имен с префиксом xsd.

Строка

Типу `Строка` соответствует тип данных `string` из пространства имен http://www.w3.org/2001/XMLSchema. Строка записывается в XML как есть.

Ниже приведены примеры представления в XML значений типа `Строка` (листинг 6.52).

Листинг 6.52. Варианты представления значений типа «Строка»

```
<!-- Не задано явно имя корневого элемента -->
<string>Это такая строка</string>

<!-- Явно задано имя корневого элемента XML -->
<Name>Иванов</Name>

<!-- Явно указан тип данных XML -->
<Data xsi:type="xsd:string">Это такая строка</Data>
```

Дата

Типу `Дата` соответствует тип данных XML `dateTime` из пространства имен http://www.w3.org/2001/XMLSchema.

Значения типа `Дата` представляются в виде CCYY-MM-DDTHH:MM:SS, где CCYY – год, представленный в виде четырех цифр; MM – месяц, представленный двумя цифрами; DD – день месяца двумя цифрами; T – латинская буква T; HH – час суток; MM – минута; SS – секунда. Такой формат даты определен как допустимый в документе XML Schema Part 2: Datatypes.

Ниже приведены примеры представления в XML значений типа `Дата` (листинг 6.53).

Листинг 6.53. Варианты представления значений типа «Дата»

```
<!-- Не задано явно имя корневого элемента -->
<dateTime>2010-09-21T12:00:00</dateTime>

<!-- Явно задано имя корневого элемента XML -->
<Started>2010-10-30T19:00:00</Started>

<!-- Явно указан тип данных XML -->
<Data xsi:type="xsd:dateTime">2010-08-25T10:00:00</Data>
```

Булево

Типу `Булево` соответствует тип данных `boolean` из пространства имен http://www.w3.org/2001/XMLSchema.

Значение `Ложь` представляется строкой false, а значение `Истина` – строкой true. Такой формат предусмотрен в документе XML Schema Part 2: Datatypes.

Ниже приведены примеры представления в XML значений типа `Булево` (листинг 6.54).

Листинг 6.54. Варианты представления значений типа «Булево»

```
<!-- Не задано явно имя корневого элемента -->
<boolean>false</boolean>

<!-- Явно задано имя корневого элемента XML -->
<Posted>true</Posted>

<!-- Явно указан тип данных XML -->
<Data xsi:type="xsd:boolean">true</Data>
```

ДвоичныеДанные

Типу `ДвоичныеДанные` соответствует тип данных XML `base64Binary` из пространства имен http://www.w3.org/2001/XMLSchema.

Значения данного типа представляются как двоичные данные, закодированные с использованием алгоритма Base64, описанного в RFC 2045.

Ниже приведены примеры представления в XML значений типа `ДвоичныеДанные` (листинг 6.55).

Листинг 6.55. Варианты представления значений типа «ДвоичныеДанные»

```
<!-- Не задано явно имя корневого элемента -->
<base64Binary>YWJjZGVm</base64Binary>

<!-- Явно задано имя корневого элемента XML -->
<BinaryData>YWJjZGVm</BinaryData>

<!-- Явно указан тип данных XML -->
<Data xsi:type="xsd:base64Binary">YWJjZGVm</Data>
```

Null

Типу `Null` соответствует тип данных XML `Null` из пространства имен http://v8.1c.ru/data. Данный тип имеет одно-единственное значение, которое представляется пустой строкой.

Ниже приведены примеры представления в XML значений типа Null (листинг 6.56).

Листинг 6.56. Варианты представления значения типа Null

```
<!-- Не задано явно имя корневого элемента -->
<v8:Null/>

<!-- Явно задано имя корневого элемента XML -->
<Selected/>

<!-- Явно указан тип данных XML -->
<Data xsi:type="v8:Null"/>
```

УникальныйИдентификатор

Типу УникальныйИдентификатор соответствует тип данных XML UUID из пространства имен http://v8.1c.ru/data.

Значения данного типа представляются в XML в соответствии с общепринятой практикой и стандартами (ISO-11578, DCE 1.1: Remote Procedure Call – Universal Unique Identifier).

Ниже приведены примеры представления в XML значений типа УникальныйИдентификатор (листинг 6.57).

Листинг 6.57. Варианты представления значений типа «УникальныйИдентификатор»

```
<!-- Не задано явно имя корневого элемента -->
<v8:UUID>3294be0f-c039-41a9-bd65-596da0dcfe68</v8:UUID>

<!-- Явно задано имя корневого элемента XML -->
<Id>da035e32-3f7a-4d87-9c6d-accf7db8cb4b</Id>

<!-- Явно указан тип данных XML -->
<Data xsi:type="v8:UUID"> 08893b0b-5ec3-4a53-a9f5-173312316919</Data>
```

ХранилищеЗначения

Типу ХранилищеЗначения соответствует тип данных XML ValueStorage из пространства имен http://v8.1c.ru/data.

Значения данного типа представляются в XML как данные ХранилищеЗначения, сохраненные в файл, а затем закодированные с использованием алгоритма Base64.

Ниже приведены примеры представления в XML значений типа ХранилищеЗначения (листинг 6.58).

Листинг 6.58. Варианты представления значений типа «ХранилищеЗначения»

```
<!-- Не задано явно имя корневого элемента -->
<v8:ValueStorage>AQEOAAAAAAAAAO+7v3siUyIsIjHQoSJ9</v8:ValueStorage>

<!-- Явно задано имя корневого элемента XML -->
<Data>AQEOAAAAAAAAAO+7v3siUyIsIjHQoSJ9</Data>

<!-- Явно указан тип данных XML -->
<Data xsi:type="v8:ValueStorage">AQEOAAAAAAAAAO+7v3siUyIsIjHQoSJ9</Data>
```

Тип ссылки на объект базы данных

Каждому из типов ссылок на объекты базы данных соответствует свой собственный тип данных XML. Имя типа данных XML для ссылок на объекты базы данных соответствует англоязычному имени типа значения ссылки «1С:Предприятия».

Так, например, для справочника Валюты англоязычное имя типа ссылки будет выглядеть как CatalogRef.Валюты. Так же будет выглядеть и имя типа данных XML.

Типы данных XML для ссылок на объекты базы данных не относятся ни к какому пространству имен (зависят от определенных в конфигурации объектов).

Значения ссылок представляются в XML как значения типа `УникальныйИдентификатор`, полученные из ссылок.

Ниже приведены примеры представления в XML значений ссылок на объекты базы данных (листинг 6.59).

Листинг 6.59. Варианты представления значений типа ссылки на объект базы данных

```
<!-- Не задано явно имя корневого элемента -->
<CatalogRef.Банки>911b5b8b-11f5-4993-9673-2c9a7a8995d5</CatalogRef.Банки>

<!-- Явно задано имя корневого элемента XML -->
<Ref>911b5b8b-11f5-4993-9673-2c9a7a8995d5</Ref>

<!-- Явно указан тип данных XML -->
<Data xsi:type="CatalogRef.Банки">911b5b8b-11f5-4993-9673-2c9a7a8995d5</Data>
```

Тип ссылки на значение перечисления

Каждому из типов ссылок на значения перечислений, определенных в конфигурации, соответствует свой собственный тип данных XML. Имя типа данных XML для ссылок на значения перечисления соответствует англоязычному имени типа «1С:Предприятия».

Так, например, для перечисления ВидыАдресов англоязычное имя типа ссылки на значение будет выглядеть как EnumRef.ВидыАдресов.

Так же будет выглядеть и имя типа данных XML.

Типы данных XML для ссылок на значения перечислений не относятся ни к какому пространству имен (также определяются структурой любой конфигурации «1С:Предприятия»). В XML ссылки на значения перечислений представляются в виде имени соответствующего значения перечисления.

Ниже приведены примеры представления в XML ссылок на значения перечислений (листинг 6.60).

Листинг 6.60. Варианты представления значений типа ссылки на значение перечисления

```
<!-- Не задано явно имя корневого элемента -->
<EnumRef.ВидыАдресов>Юридический</EnumRef.ВидыАдресов >

<!-- Явно задано имя корневого элемента XML -->
<Ref>Юридический</Ref>

<!-- Явно указан тип данных XML -->
<Data xsi:type="EnumRef.ВидыАдресов">Физический</Data>
```

Работа с простыми типами

Для работы с XML-представлениями значений простых типов предназначены два метода глобального контекста: XMLСтрока() и XMLЗначение().

Метод XMLСтрока() имеет единственный параметр – значение, для которого нужно получить XML-представление. Это значение должно относиться к типу, являющемуся простым с точки зрения XML-сериализации. В противном случае будет вызвано исключение. При нормальном завершении функция возвращает строку, которая может быть использована как текст элемента XML, представляющего значение простого типа.

Метод XMLЗначение() выполняет противоположную задачу. У этого метода два параметра. Первый параметр – тип значения, которое нужно получить из строки, второй параметр – сама строка.

Для преобразования типа данных «1С:Предприятия» в тип данных XML и наоборот предназначены методы XMLТип() и ИзXMLТипа(). Метод XMLТип() имеет один параметр – тип, для которого нужно получить соответствующий тип данных XML. Если соответствующий тип данных XML определен, то метод возвращает значение типа ТипДанныхXML. Если же соответствующего типа данных XML нет, то метод возвращает значение Неопределено.

Метод `ИзXMLТипа()` имеет два варианта вызова. В первом варианте метод имеет единственный параметр типа `ТипДанныхXML`. Во втором варианте параметра два: имя типа XML и пространство имен. В обоих случаях метод возвращает соответствующий типу данных XML тип данных «1С:Предприятия», если таковой имеется, или `Неопределено` в противном случае.

Поскольку большинство методов для работы с данными XML (за исключением примитивных типов данных) не работают на клиенте, а получать файл с данными пользователю нужно в основном именно на клиенте, здесь и далее мы будем использовать следующий подход. При записи данных XML будем производить необходимую обработку данных на сервере, записывать эти данные во временный файл, в виде двоичных данных сохранять файл во временном хранилище и адрес этих данных в хранилище передавать на клиент. И затем уже получать файл на клиенте по адресу во временном хранилище. При чтении данных XML будем сохранять XML-документ во временном хранилище и адрес данных в хранилище передавать в северную процедуру, в которой документ будет получаться из временного хранилища, записываться во временный файл, и будет производиться чтение документа.

Для записи простых типов данных XML можно использовать следующий фрагмент кода (листинг 6.61).

Листинг 6.61. Процедура «ЗаписьПростыхТиповДанных()»

```
&НаКлиенте
Процедура ЗаписьПростыхТиповДанных(Команда)

    АдресДокументаВХранилище = ЗаписьПростыхДанныхXML();
    НачатьПолучениеФайлаССервера(АдресДокументаВХранилище);

КонецПроцедуры
```

В обработчике команды для записи XML-документа вызывается серверная функция `ЗаписьПростыхДанныхXML()`, в которой производится необходимая обработка данных, затем эти данные записываются во временный файл, который в виде двоичных данных сохраняется во временном хранилище, и адрес этих данных в хранилище возвращается на клиент (листинг 6.62).

Листинг 6.62. Функция «ЗаписьПростыхДанныхXML()»

```
&НаСервереБезКонтекста
Функция ЗаписьПростыхДанныхXML()

    ИмяФайлаXML = КаталогВременныхФайлов() + "temp.xml";

    Файл = Новый ЗаписьXML;
```

```
    Файл.ОткрытьФайл(ИмяФайлаXML);
    Файл.ЗаписатьОбъявлениеXML();

    Файл.ЗаписатьНачалоЭлемента("Корневой");

    // Записать значение типа Число.
    // Получить значение "ТипДанныхXML" для числа.
    ТипXML = XMLТипЗнч(1000);

    // Создать элемент, имя которого совпадает с именем типа XML,
    // при этом записать и пространство имен типа XML.
    Файл.ЗаписатьНачалоЭлемента(ТипXML.ИмяТипа, ТипXML.URIПространстваИмен);

    Файл.ЗаписатьТекст(XMLСтрока(1000));
    Файл.ЗаписатьКонецЭлемента();

    // Записать значение типа Строка.
    ТипXML = XMLТипЗнч("Иванов");
    Файл.ЗаписатьНачалоЭлемента(ТипXML.ИмяТипа, ТипXML.URIПространстваИмен);
    Файл.ЗаписатьТекст(XMLСтрока("Иванов"));
    Файл.ЗаписатьКонецЭлемента();

    // Записать значение типа Дата.
    ТипXML = XMLТипЗнч('2020-01-31T12:00:00Z');
    Файл.ЗаписатьНачалоЭлемента(ТипXML.ИмяТипа, ТипXML.URIПространстваИмен);
    Файл.ЗаписатьТекст(XMLСтрока('2020-01-31T12:00:00Z'));
    Файл.ЗаписатьКонецЭлемента();

    // Записать значение типа Булево.
    ТипXML = XMLТипЗнч(true);
    Файл.ЗаписатьНачалоЭлемента(ТипXML.ИмяТипа, ТипXML.URIПространстваИмен);
    Файл.ЗаписатьТекст(XMLСтрока(true));
    Файл.ЗаписатьКонецЭлемента();

    // Получить ссылку на элемент номенклатуры
    СсылкаНаНоменклатуру = Справочники.Товары.НайтиПоКоду("000000003");

    ТипXML = XMLТипЗнч(СсылкаНаНоменклатуру);
    Файл.ЗаписатьНачалоЭлемента(ТипXML.ИмяТипа, ТипXML.URIПространстваИмен);
    Файл.ЗаписатьТекст(XMLСтрока(СсылкаНаНоменклатуру));
    Файл.ЗаписатьКонецЭлемента();

    Файл.ЗаписатьНачалоЭлемента(ТипXML.ИмяТипа, ТипXML.URIПространстваИмен);
    Файл.ЗаписатьТекст(Строка(СсылкаНаНоменклатуру));
    Файл.ЗаписатьКонецЭлемента();

    Файл.ЗаписатьКонецЭлемента();
    Файл.Закрыть();

    Возврат ПоместитьВоВременноеХранилище(Новый ДвоичныеДанные(ИмяФайлаXML));

КонецФункции
```

Особенность вышеприведенного механизма заключается в том, что при записи элемента в его имя записывается имя типа XML (получаемого при обращении к значению, возвращенному методом `XMLТипЗнч()`). Также записывается пространство имени типа. Следует отметить, что тип XML можно было получить и при обращении к методу `XMLТип()`.

Затем в процедуре, инициировавшей запись данных XML на клиенте, с помощью метода глобального контекста `НачатьПолучениеФайлаССервера()` начинается получение файла по адресу во временном хранилище, и файл сохраняется в локальную файловую систему пользователя. При этом пользователю предлагается диалог для выбора месторасположения получаемого файла.

В результате выполнения фрагмента кода сформируется следующий документ (листинг 6.63).

Листинг 6.63. Пример сформированного XML-документа

```
<?xml version="1.0" encoding="UTF-8"?>
<Корневой>
	<d2p1:decimal xmlns:d2p1="http://www.w3.org/2001/XMLSchema">1000</d2p1:decimal>
	<d2p1:string xmlns:d2p1="http://www.w3.org/2001/XMLSchema">Иванов</d2p1:string>
	<d2p1:dateTime xmlns:d2p1="http://www.w3.org/2001/XMLSchema">2020-01-31T12:00:00</d2p1:dateTime>
	<d2p1:boolean xmlns:d2p1="http://www.w3.org/2001/XMLSchema">true</d2p1:boolean>
	<CatalogRef.Товары>34c5d799-7636-11e9-88b7-642737df2048</CatalogRef.Товары>
	<CatalogRef.Товары>Кофемашина</CatalogRef.Товары>
</Корневой>
```

Обратите внимание на следующий факт. Пространство имен типов должно быть записано для каждого элемента с данными. Однако в пятом и шестом элементах (для ссылки справочника Товары) оно не определено. Так и должно быть, ведь для типов данных, производных от объектов конфигурации, пространство имен не определяется.

Кроме того, обратите внимание на различие результата записи ссылки. В первом случае используется метод `XMLСтрока()`, записывается значение уникального идентификатора ссылки. Во втором записывается представление ссылки (чаще всего это наименование).

Для чтения выгруженного XML-документа (см. листинг 6.63) может использоваться следующий программный код (листинг 6.64).

Листинг 6.64. Пример чтения XML-документа

```
&НаКлиенте
Процедура ЧтениеПростыхТиповДанных(Команда)

	АдресВременногоХранилища = "";
	ОповещениеОЗавершении = Новый ОписаниеОповещения(
				"ЧтениеПростыхТиповДанныхЗавершение", ЭтотОбъект);
	НачатьПомещениеФайлаНаСервер(ОповещениеОЗавершении, , , АдресВременногоХранилища
				, , УникальныйИдентификатор);

КонецПроцедуры
```

В обработчике команды для чтения XML-документа с помощью метода глобального контекста `НачатьПомещениеФайлаНаСервер()` начинается помещение файла с данными XML из локальной файловой системы во временное хранилище. При этом пользователю предлагается диалог для выбора помещаемого файла.

В метод `НачатьПомещениеФайлаНаСервер()` первым параметром передается описание оповещения, указывающее на экспортную процедуру `ЧтениеПростыхТиповДанныхЗавершение()`, которая будет выполнена, после того как файл с данными XML будет помещен во временное хранилище (листинг 6.65).

Листинг 6.65. Процедура «ЧтениеПростыхТиповДанныхЗавершение()»

```
&НаКлиенте
Процедура ЧтениеПростыхТиповДанныхЗавершение(ОписаниеПомещенногоФайла, Дополнительно) Экспорт

    ЧтениеПростыхДанныхXML(ОписаниеПомещенногоФайла.Адрес);

КонецПроцедуры
```

После того как файл будет помещен, адрес данных в хранилище (`ОписаниеПомещенногоФайла.Адрес`) передается в серверную процедуру `ЧтениеПростыхДанныхXML()`, в которой документ по этому адресу получается из временного хранилища, записывается во временный файл, и производится чтение документа (листинг 6.66).

Листинг 6.66. Процедура «ЧтениеПростыхДанныхXML()»

```
&НаСервереБезКонтекста
Процедура ЧтениеПростыхДанныхXML(АдресДокументаВХранилище)

    Сообщение = Новый СообщениеПользователю();

    ДанныеДокумента = ПолучитьИзВременногоХранилища(АдресДокументаВХранилище);
    ИмяФайлаXML = КаталогВременныхФайлов() + "temp.xml";
    ДанныеДокумента.Записать(ИмяФайлаXML);

    Файл = Новый ЧтениеXML;
    Файл.ОткрытьФайл(ИмяФайлаXML);

    // Спозиционироваться на начале элемента "Корневой".
    Файл.Прочитать();

    // Начало элемента "decimal".
    Файл.Прочитать();
    // Получить тип «1С:Предприятия» из имени элемента (
                                совпадающего с именем типа) и URI пространства имен.
    ТипЗначения = ИзXMLТипа(Файл.ЛокальноеИмя, Файл.URIПространстваИмен);
    // Спозиционироваться на тексте.
    Файл.Прочитать();
    ПолученноеЗначение = XMLЗначение(ТипЗначения, Файл.Значение);
```

```
    Сообщение.Текст = Строка(ПолученноеЗначение);
    Сообщение.Сообщить();
    // Конец элемента "decimal".
    Файл.Прочитать();

    // Начало элемента "string".
    Файл.Прочитать();
    ТипЗначения = ИзXMLТипа(Файл.ЛокальноеИмя, Файл.URIПространстваИмен);
    Файл.Прочитать();
    ПолученноеЗначение = XMLЗначение(ТипЗначения, Файл.Значение);
    Сообщение.Текст = Строка(ПолученноеЗначение);
    Сообщение.Сообщить();
    // Конец элемента "string".
    Файл.Прочитать();

    // Начало элемента "dateTime".
    Файл.Прочитать();
    ТипЗначения = ИзXMLТипа(Файл.ЛокальноеИмя, Файл.URIПространстваИмен);
    Файл.Прочитать();
    ПолученноеЗначение = XMLЗначение(ТипЗначения, Файл.Значение);
    Сообщение.Текст = Строка(ПолученноеЗначение);
    Сообщение.Сообщить();
    // Конец элемента "dateTime".
    Файл.Прочитать();

    // Начало элемента "boolean".
    Файл.Прочитать();
    ТипЗначения = ИзXMLТипа(Файл.ЛокальноеИмя, Файл.URIПространстваИмен);
    Файл.Прочитать();
    ПолученноеЗначение = XMLЗначение(ТипЗначения, Файл.Значение);
    Сообщение.Текст = Строка(ПолученноеЗначение);
    Сообщение.Сообщить();
    // Конец элемента "boolean".
    Файл.Прочитать();

    // Начало элемента "CatalogRef.Товары".
    Файл.Прочитать();
    // Получить тип «1С:Предприятия» из имени элемента (совпадающего с именем типа)
    // и URI пространства имен.
    ТипЗначения = ИзXMLТипа(Файл.ЛокальноеИмя, Файл.URIПространстваИмен);
    Файл.Прочитать();
    ПолученноеЗначение = XMLЗначение(ТипЗначения, Файл.Значение);
    Сообщение.Текст = Строка(ПолученноеЗначение);
    Сообщение.Сообщить();
    // Конец элемента "CatalogRef.Товары".
    Файл.Прочитать();

    // Начало элемента "CatalogRef.Товары".
    Файл.Прочитать();
    Файл.Прочитать();
    ПолученноеЗначение = XMLЗначение(Тип("Строка"), Файл.Значение);
    Сообщение.Текст = Строка(ПолученноеЗначение);
    Сообщение.Сообщить();
    // Конец элемента "CatalogRef.Товары"
    Файл.Прочитать();

    Файл.Закрыть();

КонецПроцедуры
```

Сложные типы

Перейдем к рассмотрению сложных (с точки зрения сериализации) типов.

Тип

Типу `Тип` соответствует тип данных XML `Type` из пространства имен http://v8.1c.ru/data. Элемент XML, представляющий значение данного типа, содержит текст. В нем записано имя типа XML, соответствующего типу данных «1С:Предприятия».

Примеры представления в XML значений типа `Тип` приведены ниже (листинг 6.67).

Листинг 6.67. Варианты представления значений типа «Тип»

```
<!-- Не задано явно имя корневого элемента -->
<v8:Type>v8:ValueStorage</v8:Type>

<!-- Явно задано имя корневого элемента XML -->
<Tp>xsd:string</Tp>

<!-- Явно указан тип данных XML -->
<Data xsi:type="v8:Type">v8:ValueStorage<Data>
```

На первый взгляд тип `Тип` относится не к сложным, а к простым типам данных, так как элемент, представляющий значение данного типа, не содержит вложенных элементов. Однако это не так. Вложенных элементов действительно нет. Но при этом текст элемента, содержащий имя типа данных XML, содержит префикс пространства имен типа, который должен быть определен в данном элементе или одном из родительских элементов, что делает текст элемента не вполне самодостаточным. Поэтому данный тип не отнесен к простым типам.

ОписаниеТипов

Типу `ОписаниеТипов` соответствует тип данных XML `TypeDescription` из пространства имен http://v8.1c.ru/data. Корневой элемент, представляющий значение типа `ОписаниеТипов`, содержит ряд вложенных элементов, каждый из которых содержит некоторую составляющую часть описания типов.

Назначение составляющих частей понятно из примера (листинг 6.68).

Листинг 6.68. Пример представления значений типа «ОписаниеТипов»

```
<v8:TypeDescription>
	<v8:Types>
		<v8:Type>v8:UUID</v8:Type>
		<v8:Type>CatalogRef.Банки</v8:Type>
		<v8:Type>xsd:boolean</v8:Type>
		<v8:Type>xsd:decimal</v8:Type>
	</v8:Types>
	<v8:NumberQualifiers>
		<v8:Digits>10</v8:Digits>
		<v8:FractionDigits>2</v8:FractionDigits>
		<v8:AllowedSign>Any</v8:AllowedSign>
	</v8:NumberQualifiers>
	<v8:StringQualifiers>
		<v8:Length>30</v8:Length>
		<v8:AllowedLength>Variable</v8:AllowedLength>
	</v8:StringQualifiers>
	<v8:DateQualifiers>
		<v8:DateFractions>Date</v8:DateFractions>
	</v8:DateQualifiers>
</v8:TypeDescription>
```

Вложенный элемент Types из пространства имен http://v8.1c.ru/data содержит представления отдельных типов, входящих в описание типов. Элемент с именем NumberQualifiers из пространства имен http://v8.1c.ru/data содержит квалификаторы числового значения. А элементы с именами StringQualifiers и DateQualifiers из того же пространства имен содержат квалификаторы строки и даты соответственно.

КонстантаМенеджерЗначения.<имя>

Каждому из типов `КонстантаМенеджерЗначения.<имя>` соответствует тип данных XML `ConstantValueManager.<имя>`, не относящийся ни к какому пространству имен.

Пример приведен ниже (листинг 6.69).

Листинг 6.69. Пример представления значения типа «КонстантаМенеджерЗначения.НазваниеОрганизации»

```
<ConstantValueManager.НазваниеОрганизации>
	<Value>ООО "Мебиус"</Value>
</ConstantValueManager.НазваниеОрганизации>
```

Объект

Объекты базы данных представляются в XML как совокупность значений реквизитов и табличных частей. Имя типа данных XML, соответствующего объекту базы данных, определяется как англоязычное имя типа значения «1С:Предприятия». Типы данных XML для объектов базы данных не относятся ни к какому пространству имен. Состав элементов XML, вложенных в корневой элемент, определяется типом объекта, а также составом реквизитов и табличных частей.

Каждый из реквизитов представляется элементом XML, имя которого соответствует имени реквизита. Если тип значения имеет составной тип, то элемент XML, представляющий реквизит, содержит атрибут xsi:type, в котором указан тип значения XML.

Каждая из табличных частей представляется элементом XML, имя которого совпадает с именем табличной части.

Каждая из строк табличной части представляется элементом XML с именем Row. Реквизиты табличной части представлены элементами XML, вложенными в элемент Row.

Ниже приведен пример представления в XML объекта типа `Документ.РасходнаяНакладная` (листинг 6.70).

Листинг 6.70. Пример представления значения типа «Документ.РасходнаяНакладная»

```
<?xml version="1.0" encoding="UTF-8"?>
<DocumentObject.РасходнаяНакладная xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                   xsi:type="DocumentObject.РасходнаяНакладная">
        <Ref>564baba3-9c2b-11e9-8e83-642737df2048</Ref>
        <DeletionMark>false</DeletionMark>
        <Date>2019-06-17T00:00:00</Date>
        <Number>0000000001</Number>
        <Posted>false</Posted>
        <Клиент>ba552984-9393-11e9-be71-642737df2048</Клиент>
        <Основание>Заказ 000000001 от 17.06.2019 12:00:00</Основание>
        <Товары>
                <Row>
                        <Товар>ba552988-9393-11e9-be71-642737df2048</Товар>
                        <Количество>2</Количество>
                        <Цена>3000</Цена>
                        <Сумма>6000</Сумма>
                </Row>
                <Row>
                        <Товар>ba552987-9393-11e9-be71-642737df2048</Товар>
                        <Количество>1</Количество>
                        <Цена>5000</Цена>
                        <Сумма>5000</Сумма>
                </Row>
        </Товары>
</DocumentObject.РасходнаяНакладная>
```

Набор записей

Представление в XML набора записей включает отбор, по которому получен набор записей, и сами записи, входящие в отбор. Значения отбора представлены во вложенном элементе XML с именем Filter, не относящемся ни к какому пространству имен. А все записи, составляющие набор записей, – во вложенном элементе с именем Records, также не относящемся ни к какому пространству имен. Записи представлены элементами XML с именем Record, вложенными в элемент Records. Имя элемента Record также не относится ни к какому пространству имен.

Ниже приведен пример представления в XML набора записей регистра накопления `ОстаткиТоваровКомпании` (листинг 6.71).

Листинг 6.71. Пример представления значения типа «РегистрНакопленияНаборЗаписей.ОстаткиТоваровКомпании»

```
<AccumulationRegisterRecordSet.ОстаткиТоваровКомпании>
	<Filter>
		<Recorder xsi:type="DocumentRef.РеализацияТоваров">
				1725f36e-6f35-11d7-932d-0050ba8480bd</Recorder>
	</Filter>
	<Records>
		<Record>
			<Recorder xsi:type="DocumentRef.РеализацияТоваров">
				1725f36e-6f35-11d7-932d-0050ba8480bd</Recorder>
			<Period>2020-01-13T17:45:39</Period>
			<MovementType>Expense</MovementType>
			<Active>true</Active>
			<Номенклатура>297c6556-5a08-11d7-9324-0050ba8480bd</Номенклатура>
			<СкладКомпании>317f1317-5a08-11d7-9324-0050ba8480bd</СкладКомпании>
			<Заказ xsi:nil="true"/>
			<ЦенаВРознице>0</ЦенаВРознице>
			<ХарактеристикаНоменклатуры>00000000-0000-0000-0000-000000000000
				</ХарактеристикаНоменклатуры>
			<Количество>2</Количество>
			<ПодразделениеКомпании>317f130d-5a08-11d7-9324-0050ba8480bd
				</ПодразделениеКомпании>
		</Record>
		<Record>
			<Recorder xsi:type="DocumentRef.РеализацияТоваров">
				1725f36e-6f35-11d7-932d-0050ba8480bd</Recorder>
			<Period>2020-01-13T17:45:39</Period>
			<MovementType>Expense</MovementType>
			<Active>true</Active>
			<Номенклатура>297c6558-5a08-11d7-9324-0050ba8480bd</Номенклатура>
			<СкладКомпании>317f1317-5a08-11d7-9324-0050ba8480bd</СкладКомпании>
			<Заказ xsi:nil="true"/>
			<ЦенаВРознице>0</ЦенаВРознице>
			<ХарактеристикаНоменклатуры>ac47d77e-5ec7-11d7-9329-0050ba8480bd
				</ХарактеристикаНоменклатуры>
			<Количество>2</Количество>
			<ПодразделениеКомпании>317f130d-5a08-11d7-9324-0050ba8480bd
				</ПодразделениеКомпании>
		</Record>
	</Records>
</AccumulationRegisterRecordSet.ОстаткиТоваровКомпании>
```

УдалениеОбъекта

Типу `УдалениеОбъекта` соответствует тип данных XML `ObjectDeletion` из пространства имен http://v8.1c.ru/data. Корневой элемент XML-представления значения типа `УдалениеОбъекта` содержит один вложенный элемент с именем Ref из пространства имен http://v8.1c.ru/data, в котором находится представление ссылки на объект базы данных.

Ниже приведен пример представления в XML объекта типа `УдалениеОбъекта` (листинг 6.72).

Листинг 6.72. Пример представления значения типа «УдалениеОбъекта»

```
<v8:ObjectDeletion xmlns="http://v8.1c.ru/data">
        <v8:Ref xsi:type="CatalogRef.Банки">60c5cec3-7f6f-4ec3-9620-e757fe3614ca</v8:Ref>
</v8:ObjectDeletion>
```

Работа со сложными типами

Для работы со сложными с точки зрения сериализации значениями могут использоваться следующие методы «1С:Предприятия» (они же могут использоваться и для простых типов):

- `ЗаписатьXML()`,
- `ПрочитатьXML()`.

Метод `ЗаписатьXML()` имеет два обязательных параметра. Первый параметр – это объект типа `ЗаписьXML`, через который осуществляется запись XML, а второй – значение, которое должно быть записано в XML. Необязательные параметры образуют три различных варианта вызова метода.

В простейшем случае параметра три, и в качестве третьего параметра указывается значение перечисления `НазначениеТипаXML`, определяющее необходимость явного указания типа данных XML в атрибуте xsi:type корневого элемента XML.

У следующего варианта вызова в качестве третьего параметра используется строковое значение, указывается имя корневого элемента XML. При этом подразумевается, что пространство имен не определено. Четвертый параметр – значение типа `НазначениеТипаXML`, определяющее необходимость явного указания типа данных XML.

И, наконец, у последнего варианта вызова после параметра, указывающего имя корневого элемента XML, появляется еще один параметр – строковое значение, обозначающее пространство имен, к которому относится корневой элемент. Последний параметр по-прежнему имеет тип `НазначениеТипаXML`.

Рассмотрим пример использования метода ЗаписатьXML (листинг 6.73).

Листинг 6.73. Пример использования метода «ЗаписатьXML»

```
Знач = "Строка такая";
ЗаписатьXML(Зп, Знач);
ЗаписатьXML(Зп, Знач, "Root", НазначениеТипаXML.Явное);
ЗаписатьXML(Зп, Знач, "Root", "urn:some-namespace");
```

В результате выполнения приведенного кода будет получен следующий XML-фрагмент (листинг 6.74).

Листинг 6.74. Фрагмент XML-документа

```
<string>Строка такая</string>
<Root xsi:type="xsd:string">Строка такая</Root>
<d1p1:Root xmlns:d1p1="urn:some-namespace">Строка такая</d1p1:Root>
```

Если в качестве значения, помещаемого в XML, будет передано значение типа, который не может быть представлен в XML, то будет вызвано исключение.

Метод ПрочитатьXML() предназначен для чтения значений из XML. Данный метод имеет один обязательный параметр – объект ЧтениеXML, из которого должно быть прочитано значение. В качестве второго параметра может быть указан тип значения, которое должно быть прочитано из XML. Если тип значения явно указан в XML, то в качестве второго параметра может быть указано значение Неопределено или же он может быть вообще опущен. В этом случае метод ПрочитатьXML() пытается определить тип читаемого значения по содержимому атрибута xsi:type, а если атрибут xsi:type отсутствует – то по имени элемента. Если тип установить не удалось или значение указанного типа не может быть прочитано из XML, то вызывается исключение. При удачном завершении метод ПрочитатьXML() возвращает считанное значение.

Следует обратить внимание на то, как считываются менеджеры значений констант, объекты базы данных и наборы записей. После успешного выполнения чтения метод ПрочитатьXML() возвращает считанное из XML значение, но это значение еще не записано в базу данных. Если, например, считан элемент справочника, то для того, чтобы считанный элемент справочника оказался записанным в базу данных, необходимо обратиться к его методу Записать(), как и при «обычной» записи измененного состояния объекта. Это же относится и к другим объектам базы данных, менеджерам записи констант и наборам записей.

При чтении объекта базы данных из XML в базе данных производится поиск объекта с таким же значением ссылки. Если такой объект найден, то считывание из XML выглядит так, как будто объект был прочитан из базы данных, после чего значения его реквизитов, табличных частей и т. п. перезаписываются полученными из XML значениями. Если же объект по ссылке не найден, то считывание из XML выглядит как создание нового объекта, установка ему значения ссылки и заполнение его содержимого значениями, прочитанными из XML.

Метод `ВозможностьЧтенияXML()` определяет, возможно ли считывание значения из объекта `ЧтениеXML`, находящегося в текущей позиции документа XML. Объект `ЧтениеXML` передается данному методу в качестве параметра. Если метод возвращает `Истина`, то чтение возможно, если `Ложь` – значение не может быть считано.

Метод `ПолучитьXMLТип()` позволяет получить из объекта `ЧтениеXML` тип данных XML, соответствующий текущей позиции документа XML. Данный метод также имеет один параметр – `ЧтениеXML`.

Рассмотрим использование вышеперечисленных методов на примере. В контексте процедуры `ЗаписьСложныхТиповДанных()` существует переменная `Документ` (тип `ДокументСсылка.<имя>`). Данный документ имеет следующую структуру:

- реквизит Клиент, тип `СправочникСсылка.Клиенты`;
- реквизит Основание, тип `Строка`;
- табличная часть Товары, реквизиты табличной части:
  - Товар, тип `СправочникСсылка.Товары`;
  - Количество, тип `Число`;
  - Цена, тип `Число`;
  - Сумма, тип `Число`.

Следует отметить, что метод `ЗаписатьXML()` сам определяет структуру выгружаемого объекта. В данном случае она приводится, для того чтобы пояснить полученный XML-документ.

При записи и чтении сложных типов XML данных так же, как это подробно описано в разделе «Работа с простыми типами», передача XML-документа между клиентом и сервером происходит через временное хранилище.

Для записи сложных типов XML может использоваться следующий программный код (листинги 6.75, 6.76).

Листинг 6.75. Пример записи XML-документа

```
&НаКлиенте
Процедура ЗаписьСложныхТиповДанных(Команда)

	АдресДокументаВХранилище = ЗаписьСложныхДанныхXML();
	НачатьПолучениеФайлаССервера(, АдресДокументаВХранилище, "c:\temp\example3.xml");

КонецПроцедуры
```

Листинг 6.76. Функция «ЗаписьСложныхДанныхXML()»

```
&НаСервереБезКонтекста
Функция ЗаписьСложныхДанныхXML()

	ИмяФайлаXML = КаталогВременныхФайлов() + "temp.xml";

	Файл = Новый ЗаписьXML;
	Файл.ОткрытьФайл(ИмяФайлаXML);
	Файл.ЗаписатьОбъявлениеXML();

	// Получить ссылку на документ.
	ССылкаНаДокумент = Документы.РасходнаяНакладная.НайтиПоНомеру("000000001");
	ДокументОбъект = ССылкаНаДокумент.ПолучитьОбъект();

	ЗаписатьXML(Файл, ДокументОбъект, НазначениеТипаXML.Явное);
	Файл.Закрыть();

	Возврат ПоместитьВоВременноеХранилище(Новый ДвоичныеДанные(ИмяФайлаXML));

КонецФункции
```

Заметим, что в этом примере используется другой (по сравнению с предыдущим примером) вариант вызова метода `НачатьПолучениеФайлаССервера()` – с указанием имени получаемого файла. При этом пользователь не имеет возможности выбрать имя и расположение сохраняемого файла.

В результате работы процедуры сформируется XML-документ следующего вида (листинг 6.77).

Листинг 6.77. Пример сформированного XML-документа

```
<?xml version="1.0" encoding="UTF-8"?>
<DocumentObject.РасходнаяНакладная xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                   xsi:type="DocumentObject.РасходнаяНакладная">
	<Ref>564baba3-9c2b-11e9-8e83-642737df2048</Ref>
	<DeletionMark>false</DeletionMark>
	<Date>2019-06-17T00:00:00</Date>
	<Number>000000001</Number>
	<Posted>false</Posted>
	<Клиент>ba552984-9393-11e9-be71-642737df2048</Клиент>
```

```
    <Основание>Заказ 000000001 от 17.06.2019 12:00:00</Основание>
    <Товары>
        <Row>
            <Товар>ba552988-9393-11e9-be71-642737df2048</Товар>
            <Количество>2</Количество>
            <Цена>3000</Цена>
            <Сумма>6000</Сумма>
        </Row>
        <Row>
            <Товар>ba552987-9393-11e9-be71-642737df2048</Товар>
            <Количество>1</Количество>
            <Цена>5000</Цена>
            <Сумма>5000</Сумма>
        </Row>
    </Товары>
</DocumentObject.РасходнаяНакладная>
```

Можно сказать, что структура выгруженного документа повторяет структуру его метаданных, причем дополнительно включаются такие данные, как:

- ссылка на документ (`Ref`);
- пометка на удаление (`DeletionMark`);
- дата (`Date`);
- номер (`Number`);
- проведен (`Posted`).

Для чтения выгруженного XML-документа (см. листинг 6.77) может использоваться следующий программный код (листинг 6.78).

Листинг 6.78. Процедура чтения XML-документа

```
&НаКлиенте
Процедура ЧтениеСложныхТиповДанных(Команда)

    АдресВременногоХранилища = "";
    ОповещениеОЗавершении = Новый ОписаниеОповещения(
                            "ЧтениеСложныхТиповДанныхЗавершение", ЭтотОбъект);
    НачатьПомещениеФайлаНаСервер(ОповещениеОЗавершении, , ,
                            АдресВременногоХранилища, "c:\temp\example2.xml",
        УникальныйИдентификатор);

КонецПроцедуры
```

Заметим, что в этом примере используется другой (по сравнению с предыдущим примером) вариант вызова метода `НачатьПомещениеФайлаНаСервер()` – с указанием имени помещаемого файла. При этом пользователю не показывается диалог выбора файла.

В метод `НачатьПомещениеФайлаНаСервер()` первым параметром передается описание оповещения, указывающее на экспортную процедуру `ЧтениеСложныхТиповДанныхЗавершение()`, которая будет выполнена, после того,как файл с данными XML будет помещен во временное хранилище (листинг 6.79).

Листинг 6.79. Процедура «ЧтениеСложныхТиповДанныхЗавершение()»

```
&НаКлиенте
Процедура ЧтениеПростыхТиповДанныхЗавершение(ОписаниеПомещенногоФайла, Дополнительно) Экспорт

    ЧтениеСложныхДанныхXML(ОписаниеПомещенногоФайла.Адрес);

КонецПроцедуры
```

После того как файл будет помещен, вызывается серверная процедура `ЧтениеСложныхДанныхXML()`, в которую передается адрес помещенного файла во временном хранилище. В этой процедуре и производится, собственно, чтение документа (листинг 6.80).

Листинг 6.80. Процедура «ЧтениеСложныхДанныхXML()»

```
&НаСервереБезКонтекста
Процедура ЧтениеСложныхДанныхXML(АдресДокументаВХранилище)

    ДанныеДокумента = ПолучитьИзВременногоХранилища(АдресДокументаВХранилище);
    ИмяФайлаXML = КаталогВременныхФайлов() + "temp.xml";
    ДанныеДокумента.Записать(ИмяФайлаXML);

    Файл = Новый ЧтениеXML;
    Файл.ОткрытьФайл(ИмяФайлаXML);

    // Спозиционироваться на начале элемента, содержащего документ.
    Если Файл.Прочитать() Тогда

        // Проверить возможность чтения значения.
        Если ВозможностьЧтенияXML(Файл) Тогда

            // Получить значение ДокументОбъект.Имя
            Объект = ПрочитатьXML(Файл);
            Объект.Дата = '2020-01-31';
            Объект.Записать();
        КонецЕсли;
    КонецЕсли;

    Файл.Закрыть();

КонецПроцедуры
```

Следует отметить, что если объект, в который производится загрузка, имеет отличную от выгружаемого объекта структуру (отличается состав реквизитов, табличных частей, реквизитов табличных частей или порядок их следования), то метод `ВозможностьЧтенияXML()` может возвратить значение `Истина`. При попытке проведения чтения объекта из XML-документа будет вызвано исключение. Это объясняется тем, что метод `ВозможностьЧтенияXML()` анализирует текущее состояние объекта `ЧтениеXML`. В этот момент текущим является начало соответствующего элемента. Если имя элемента, атрибуты «понятны» методу, он и возвращает значение `Истина`. Проверка структуры, других моментов методом `ВозможностьЧтенияXML()` не производится.

HTML-документ

Можно сказать, что HTML-документ представляет собой текстовый файл, имеющий определенную структуру, которому принудительно дано расширение HTM или HTML. На самом деле более правильно говорить о языке HTML (язык разметки гипертекста). С этой точки зрения HTML-документ – это документ с текстовым содержимым, написанный на языке HTML.

Структура HTML-документа задается с помощью так называемых «тегов» (выражений, заключенных в угловые скобки). Набор тегов и их свойств (значений свойств) зафиксирован. Изучение языка HTML сводится к изучению конечного множества тегов, свойств тегов и их значений.

Если говорить о «задаче, решаемой языком HTML», то это форматирование отображаемых данных. В этом смысле теги можно разбить на «теги структуры документа», «теги форматирования» и другие.

Основные теги, задающие структуру HTML-документа, следующие (листинг 6.81).

Листинг 6.81. Основные теги, задающие структуру HTML-документа

```
<html>
	<head>
		<meta content="text/html; charset=windows-1251">
		<title>Пример страницы</title>
	</head>
	<body bgcolor=lightyellow text=darkblue>
		«Тело» документа
	</body>
</html>
```

В теле документа могут располагаться различные форматирующие теги. Пример таких тегов приведен в листинге 6.82.

Листинг 6.82. Пример форматирующих тегов

```
<h1>Заголовок</h1>
<b>Полужирный;<i> полужирный и наклонный;</i></b><i> наклонный текст</i>
<p>новый абзац</p>
```

Данные в HTML-документе можно организовывать в таблицы (листинг 6.83).

Листинг 6.83. Пример таблицы

```
<table>
        <tr>
                <td>первое</td>
                <td>второе</td>
        </tr>
        <tr>
                <td>третье</td>
                <td>четвертое</td>
        </tr>
</table>
```

Для навигации (переходу к другим ресурсам) могут использоваться гиперссылки. Для обозначения так называемых якорей в документе используются теги <a> (листинг 6.84).

Листинг 6.84. Пример использования гиперссылки

```
<a href='http://www.v8.1c.ru'>1C</a>
```

Для отображения картинки используются теги <img> (листинг 6.85).

Листинг 6.85. Пример использования картинки

```
<img src='C:/TEMP/pict_1.png' alt='Jardin'></img>
```

Следует отметить, что в данном разделе не ставится задача познакомить со всем множеством тегов: это удел специализированной литературы. Мы показали только основные теги HTML-документа.

Если собрать все продемонстрированные фрагменты и запустить полученный HTML-документ на просмотр в веб-браузере, то получим, например, следующую картинку (рис. 6.8).

Рис. 6.8. Пример HTML-документа

Поле HTML-документа

В платформе «1С:Предприятие» существует специализированный вид поля формы, который позволяет просматривать HTML-документы.

Функциональность поля формы вида `ПолеHTMLДокумента` целиком зависит от установленной на локальном компьютере программы интернет-браузера (она ею обеспечивается).

Для просмотра различных интернет-ресурсов, возвращающих ответ в виде HTML-документа (это не обязательно должны быть определенные в явном виде HTML-документы, также можно смотреть результат обращения к файлам *.asp, *.jsp и т. д.), нужно подставить адрес запрашиваемого ресурса в значение строкового реквизита, который отображается полем HTML-документа (листинг 6.86).

Листинг 6.86. Пример использования поля HTML-документа

```
АдресHTML = "http://www.v8.1c.ru";
```

Для работы с историей просмотренных страниц можно использовать методы `Вперед()` и `Назад()` – листинг 6.87.

Листинг 6.87. Пример использования методов «Вперед()» и «Назад()»

```
Элементы.ПолеHTML.Вперед();
Элементы.ПолеHTML.Назад();
```

У поля формы вида Поле HTML документа существуют следующие события:

- Событие ПриНажатии. Возникает при нажатии (с помощью мыши или клавиатуры) на любой элемент внутри окна поля HTML-документа.
- Событие ДокументСформирован. Возникает, когда HTML-документ построен и готов к использованию.

Если понадобится активно работать с объектной моделью, рекомендуется использовать обработчик события поля HTML-документа ДокументСформирован. Событие возникает, когда документ загружен полностью (либо в ряде случаев, когда загружен пустой документ: blank). Этот момент можно использовать для начала работы с объектной моделью.

Платформа «1С:Предприятие» поддерживает работу с объектной моделью HTML-документа, которая зависит от установленной операционной системы и используемого интернет-браузера. Поэтому в каждом конкретном случае программная модель может быть разной.

Объектная модель документа

Для доступа к объектной модели документа используется свойство Документ поля HTML-документа. С помощью этого свойства можно получить объект document типа ВнешнийОбъект, представленный в объектной модели.

Используя свойства (documentElement, URL, innerHTML и др.) и методы (createElement(), appendChild(), getElementsByTagName() и др.) объекта document, можно программным образом изменять содержимое HTML-документа. Полностью описание и приемы работы с объектной моделью документа можно найти в специализированной литературе. Ниже мы рассмотрим только небольшой пример на эту тему.

Пусть существует следующий HTML-документ (листинг 6.88).

Листинг 6.88. Пример HTML-документа

```
<html>
    <head>
        <title>Анкета</title>
    </head>
    <body>
        <form name='anketa' action='input.asp'>
            <table>
                <tr>
                    <td>Организация</td>
                    <td>
                        <input type=text name='firma'>
                    </td>
```

```
                    </tr>
                    <tr>
                        <td>Кому</td>
                        <td>
                            <input type=text name='to'>
                        </td>
                    </tr>
                    <tr>
                        <td>Пожелание</td>
                        <td>
                            <input type=text name='message'>
                        </td>
                    </tr>
                    <tr>
                        <td colspan=2 align=center>
                            <input type=submit name='ok' value='Отправить'>
                        </td>
                    </tr>
                </table>
            </form>
        </body>
</html>
```

Визуально этот документ отображается следующим образом (рис. 6.9).

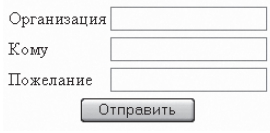


Рис. 6.9. HTML-документ

С точки зрения объектной модели его можно упрощенно представить следующим образом (рис. 6.10).

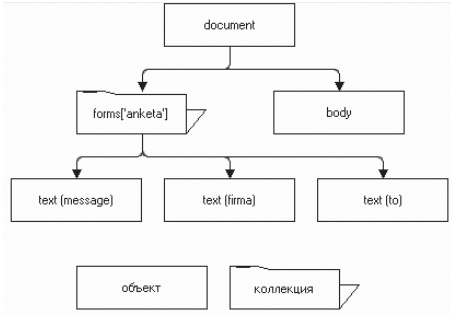


Рис. 6.10. Структура HTML-документа

Для того чтобы заполнить анкету, загруженную в поле HTML-документа, можно воспользоваться следующим фрагментом кода (листинг 6.89).

Листинг 6.89. Пример заполнения анкеты

```
Док = Элементы.ПолеHTML.Документ;
Док.forms["anketa"].firma.Value = "ООО Вперед";
Док.forms["anketa"].to.Value = "директору";
Док.forms["anketa"].message.Value = "здоровья и счастья!";
```

Сначала мы получаем доступ к объектной модели документа. Затем получаем элемент коллекции форм (`dok.forms`) с именем «anketa». И заполняем значения полей (`firma`, `to`, `message`) этой формы.

В результате заполненная форма будет выглядеть следующим образом (рис. 6.11).

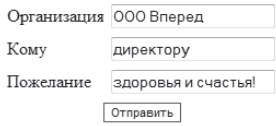


Рис. 6.11. Заполненная форма HTML-документа

Примеры работы

В этом разделе мы рассмотрим небольшие примеры решения наиболее часто встречающихся задач при работе с HTML-документами: как прочитать из файла и записать в файл HTML-документ, как получить HTML-текст документа, как из текста получить сам HTML-документ, как программно изменить содержимое HTML-документа, вставить фрагмент в HTML-текст и т. д.

В форме нашей демонстрационной обработки используются следующие параметры:

- АдресHTML – адрес ресурса. Строковый реквизит, который отображается полем HTML-документа;
- ПолеHTML – поле формы для отображения HTML-документа, связанное с реквизитом АдресHTML;
- Текст_HTML – строковый реквизит, содержащий текст HTML-документа. В форме существует соответствующее текстовое поле, связанное с этим реквизитом.

Отобразить содержимое, URL и HTML-текст веб-страницы

Прежде всего добавим в форму поле ввода, связанное с реквизитом АдресHTML, и убедимся, что при вводе в это поле (с заголовком URL) адреса интернет-ресурса (например, «http://www.v8.1c.ru») содержимое этого ресурса отображается в поле формы ПолеHTML типа Поле HTML документа .

В результате запрашиваемая веб-страница в форме обработки будет выглядеть следующим образом (рис. 6.12).

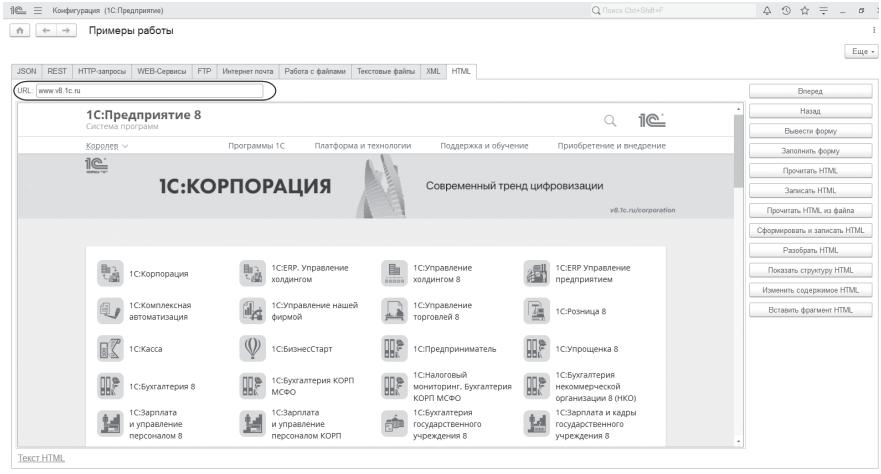


Рис. 6.12. Просмотр веб-страницы

Теперь для навигации по просмотренным страницам добавим команды Вперед и Назад. Обработчики этих команд заполним следующим образом (листинги 6.90, 6.91).

Листинг 6.90. Обработчик команды «Вперед»

```
&НаКлиенте
Процедура Вперед(Команда)

        Элементы.ПолеHTML.Вперед();

КонецПроцедуры
```

Листинг 6.91. Обработчик команды «Назад»

```
&НаКлиенте
Процедура Назад(Команда)

    Элементы.ПолеHTML.Назад();

КонецПроцедуры
```

Затем для отображения URL и текста загруженной веб-страницы (HTML-документа) создадим обработчик события `ДокументСформирован` поля ПолеHTML и заполним следующим образом (листинг 6.92).

Листинг 6.92. Обработчик события «ДокументСформирован» поля HTML-документа

```
&НаКлиенте
Процедура ПолеHTMLДокументСформирован(Элемент)

    ДокументHTML = Элементы.ПолеHTML.Документ;
    АдресHTML = ДокументHTML.URL;
    Текст_HTML = ДокументHTML.documentElement.innerHTML;

КонецПроцедуры
```

В этом обработчике с помощью свойства `Документ` поля ПолеHTML мы получаем доступ к объектной модели документа. И, используя свойства `URL` и `innerHTML` внешнего объекта `document`, сохраняем соответственно URL и текст HTML-документа в реквизитах АдресHTML и Текст_HTML.

В результате при навигации по веб-страницам URL ресурса будет соответствующим образом меняться (рис. 6.13), а в текстовом поле, отображающем реквизит Текст_HTML, будет показываться HTML-текст ресурса (рис. 6.14).

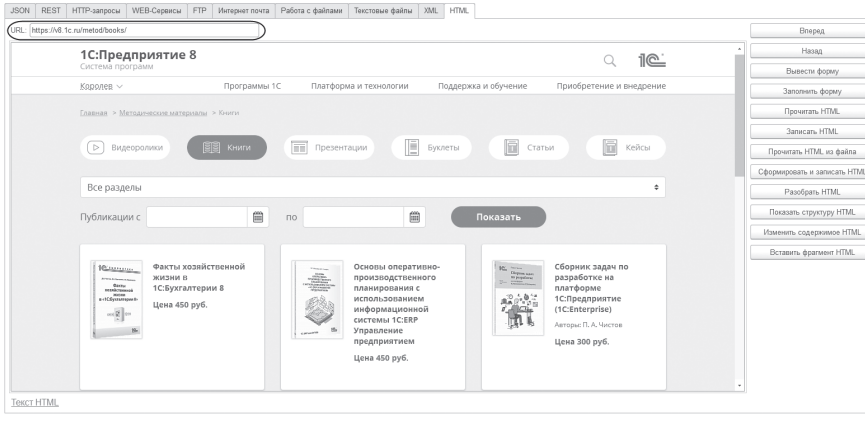


Рис. 6.13. Навигация по веб-ресурсам

Рис. 6.14. HTML-текст

Прочитать HTML-документ из текста

Предположим, у нас есть произвольный текст, написанный на языке HTML, и нам нужно отобразить соответствующий этому тексту HTML-документ в форме обработки.

Для решения этой задачи добавим команду ПрочитатьHTML. Обработчик команды заполним следующим образом (листинг 6.93).

Листинг 6.93. Обработчик команды «ПрочитатьHTML»

```
&НаКлиенте
Процедура ПрочитатьHTML(Команда)

    СтрокаHTML = "<html><head><meta content='text/html; charset=windows-1251'>
                 |    <title>Пример страницы</title></head><body><h1>Заголовок</h1>
                 |    <b>Полужирный;<i> полужирный и наклонный;</i></b><i> наклонный текст
                 |    </i><p>новый абзац</p>
                 |    <table><tr><td>Первое</td><td>Второе</td></tr>
                 |    <tr><td>Третье</td><td>Четвертое</td></tr></table>
                 |    <p><img src='C:/TEMP/pict_1.png' alt='Jardin'></img></p>
                 |    <p><a href='http://www.v8.1c.ru'>1С</a></p></body></html>";

    Документ_HTML = Элементы.ПолеHTML.Документ;
    Документ_HTML.documentElement.innerHTML = СтрокаHTML;
    Текст_HTML = СтрокаHTML;
    //АдресHTML = СтрокаHTML;

КонецПроцедуры
```

В этом обработчике мы устанавливаем HTML-текст документа напрямую в свойство `innerHTML` объектной модели документа. На самом деле можно было бы просто установить строку с HTML-текстом в качестве значения

реквизита АдресHTML. Тогда платформа все сделает сама и предыдущие три строчки кода не понадобятся. Этот вариант мы показали в качестве альтернативы, а более простой способ рассмотрим в следующем примере.

Прочитать HTML-документ из файла

Предположим, нам нужно прочитать HTML-документ из файла и отобразить его содержимое.

Для решения этой задачи добавим команду ПрочитатьHTMLИзФайла. Обработчик команды заполним следующим образом (листинг 6.94).

Листинг 6.94. Обработчик команды «ПрочитатьHTMLИзФайла»

```
&НаКлиенте
Процедура ПрочитатьHTMLИзФайла(Команда)

	ЧтениеHTML = Новый ЧтениеHTML;
	ЧтениеHTML.ОткрытьФайл("c:\temp\text_html.html", "UTF-8");

	ПостроительDOM = Новый ПостроительDOM;
	ДокументHTML = ПостроительDOM.Прочитать(ЧтениеHTML);
	ЧтениеHTML.Закрыть();

	АдресHTML = ПолучитьТекстHTML(ДокументHTML);

КонецПроцедуры
```

В этом обработчике мы создаем объект `ЧтениеHTML` и открываем для чтения HTML-файл методом `ОткрытьФайл()` этого объекта. Затем создаем объект `ПостроительDOM` и вызываем его метод `Прочитать()`, в который передаем объект чтения `ЧтениеHTML`. В результате на основе прочитанных данных будет создан объект `ДокументHTML`.

Теперь нам нужно получить HTML-текст этого документа, чтобы запомнить его реквизите АдресHTML. Для этого мы используем функцию `ПолучитьТекстHTML()`, в которую передаем полученный объект `ДокументHTML` (листинг 6.95).

Листинг 6.95. Функция «ПолучитьТекстHTML()»

```
&НаКлиенте
Функция ПолучитьТекстHTML(ДокументHTML)

	ЗаписьHTML = Новый ЗаписьHTML;
	ЗаписьHTML.УстановитьСтроку();
	ЗаписьDOM = Новый ЗаписьDOM;
	ЗаписьDOM.Записать(ДокументHTML, ЗаписьHTML);

	Возврат ЗаписьHTML.Закрыть();

КонецФункции
```

В этой функции мы создаем объект ЗаписьHTML и устанавливаем запись данных в строку методом этого объекта УстановитьСтроку(). Затем создаем объект ЗаписьDOM и вызываем его метод Записать(), в который передаем полученный в виде параметра ДокументHTML и объект записи ЗаписьHTML. Поскольку была установлена запись в строку, то при выполнении метода Закрыть() будет получена строка с текстом HTML-документа, который и возвращает функция ПолучитьТекстHTML().

После загрузки этого текста в реквизит АдресHTML (см. листинг 6.94) содержимое документа будет показано в поле HTML-документа в форме обработки и у этого поля будет вызвано событие ДокументСформирован. Согласно обработчику этого события (см. листинг 6.92) текст HTML-документа будет запомнен в реквизите Текст_HTML (рис. 6.15).

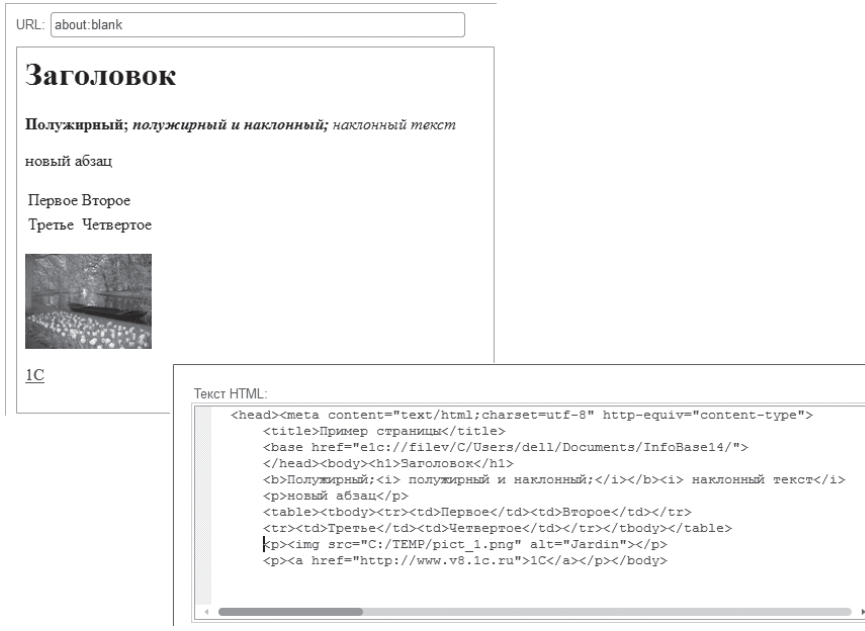


Рис. 6.15. Содержимое и текст прочитанного из файла HTML-документа

Поскольку текст HTML-документа отображается в текстовом поле, связанном с реквизитом Текст_HTML, то можно изменить этот текст, сохранить изменения и посмотреть результат.

Для этого добавим команду СохранитьИзменения. Обработчик команды заполним следующим образом (листинг 6.96).

Листинг 6.96. Обработчик команды «СохранитьИзменения»

```
&НаКлиенте
Процедура СохранитьИзменения(Команда)

    Документ_HTML = Элементы.ПолеHTML.Документ;
    Документ_HTML.documentElement.innerHTML = Текст_HTML;

КонецПроцедуры
```

В этом обработчике, используя свойство `innerHTML` объектной модели документа, мы заменяем текстовое содержимое HTML-документа на новое, хранящееся в реквизите Текст_HTML.

Например, изменив имя рисунка в тексте HTML-документа и нажав кнопку Сохранить изменения, мы увидим следующий результат (рис. 6.16).

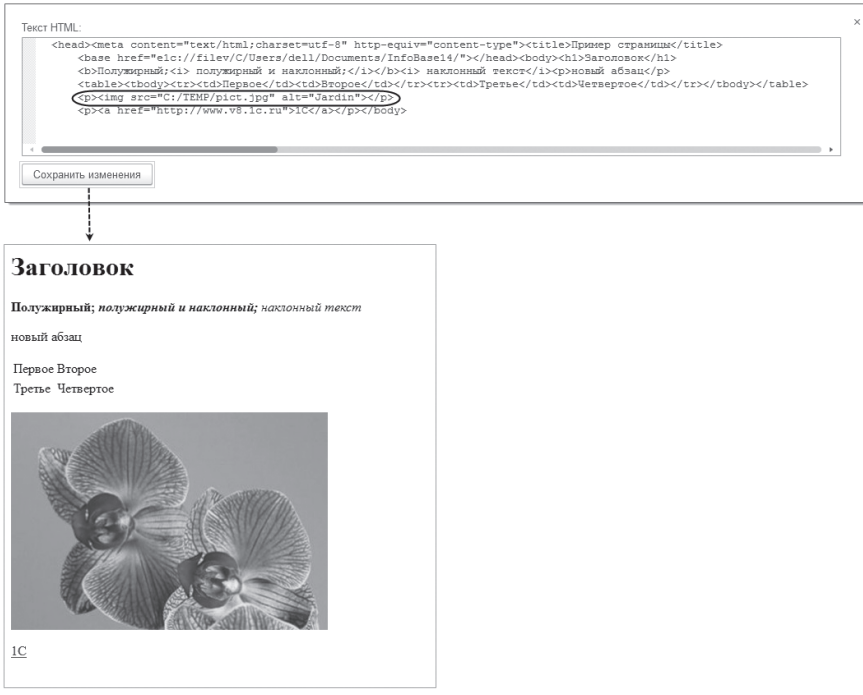


Рис. 6.16. Содержимое и текст измененного HTML-документа

Записать HTML-документ в файл

Предположим, нам нужно записать HTML-документ, показанный в форме обработки, в файл.

Для решения этой задачи добавим команду ЗаписатьHTML. Обработчик команды заполним следующим образом (листинг 6.97).

Листинг 6.97. Обработчик команды «ЗаписатьHTML»

```
&НаКлиенте
Процедура ЗаписатьHTML(Команда)

	ДокументHTML = ПолучитьДокументHTML(Текст_HTML);

	ЗаписьHTML = Новый ЗаписьHTML;
	ЗаписьHTML.ОткрытьФайл("c:\temp\text_html2.html", "UTF-8");
	ЗаписьDOM = Новый ЗаписьDOM;
	ЗаписьDOM.Записать(ДокументHTML, ЗаписьHTML);
	ЗаписьHTML.Закрыть()

КонецПроцедуры
```

В этом обработчике сначала мы получаем объект `ДокументHTML` с помощью функции `ПолучитьДокументHTML()`, в которую передаем текст HTML-документа, содержащийся в реквизите Текст_HTML. Эта функция будет рассмотрена ниже, в листинге 6.98.

После этого мы создаем объект `ЗаписьHTML` и открываем файл для записи HTML-документа методом `ОткрытьФайл()` этого объекта. Затем создаем объект `ЗаписьDOM` и вызываем его метод `Записать()`, в который передаем полученный ранее `ДокументHTML` и объект записи `ЗаписьHTML`. По окончании записи файл закрывается.

Листинг 6.98. Функция «ПолучитьДокументHTML()»

```
&НаКлиенте
Функция ПолучитьДокументHTML(СтрокаHTML)

	ЧтениеHTML = Новый ЧтениеHTML;
	ЧтениеHTML.УстановитьСтроку(СтрокаHTML);

	ПостроительDOM = Новый ПостроительDOM;
	ДокументHTML = ПостроительDOM.Прочитать(ЧтениеHTML);
	ЧтениеHTML.Закрыть();

	Возврат ДокументHTML;

КонецФункции
```

В этой функции мы создаем объект ЧтениеHTML и методом этого объекта УстановитьСтроку() устанавливаем чтение данных из строки, полученной в параметре СтрокаHTML. Затем создаем объект ПостроительDOM и вызываем его метод Прочитать(), в который передаем объект чтения ЧтениеHTML. В результате на основе прочитанных данных будет создан объект ДокументHTML, который и возвращает функция ПолучитьДокументHTML().

Сформировать HTML-документ и записать в файл

Предположим, нам нужно полностью сформировать HTML-документ на основе произвольного текста и записать этот документ в файл.

Для решения этой задачи добавим команду СформироватьИЗаписатьHTML. Обработчик команды заполним следующим образом (листинг 6.99).

Листинг 6.99. Обработчик команды «СформироватьИЗаписатьHTML»

```
&НаКлиенте
Процедура СформироватьИЗаписатьHTML(Команда)

    СтрокаHTML = "<html><head><meta content='text/html; charset=windows-1251'>
                 |       <title>Пример страницы</title></head><body><h1>Заголовок</h1>
                 |       <b>Полужирный;<i> полужирный и наклонный;</i></b><i> наклонный текст
                 |                                                    </i><p>новый абзац</p>
                 |       <table><tr><td>Первое</td><td>Второе</td></tr>
                 |       <tr><td>Третье</td><td>Четвертое</td></tr></table>
                 |       <p><img src='C:/TEMP/pict_1.png' alt='Jardin'></img></p>
                 |       <p><a href='http://www.v8.1c.ru'>1C</a></p></body></html>";

    ДокументHTML = ПолучитьДокументHTML(СтрокаHTML);

    ЗаписьHTML = Новый ЗаписьHTML;
    ЗаписьHTML.ОткрытьФайл("c:\temp\text_html.html", "UTF-8");
    ЗаписьDOM = Новый ЗаписьDOM;
    ЗаписьDOM.Записать(ДокументHTML, ЗаписьHTML);
    ЗаписьHTML.Закрыть()

КонецПроцедуры
```

Этот обработчик подобен обработчику из предыдущего раздела (см. листинги 6.97, 6.98), с той лишь разницей, что текст HTML-документа устанавливается из произвольной строки.

Разобрать HTML

Предположим, нам нужно разобрать HTML-документ с целью дальнейшей обработки всех элементов документа, содержащих картинки, якоря, формы и т. д.

Для решения этой задачи добавим команду РазобратьHTML. Обработчик команды заполним следующим образом (листинг 6.100).

Листинг 6.100. Обработчик команды «РазобратьHTML»

```
&НаКлиенте
Процедура РазобратьHTML(Команда)

	ДокументHTML = ПолучитьДокументHTML(Текст_HTML);

	Сообщение = Новый СообщениеПользователю();
	ТекстСообщения = ДокументHTML.ЭлементДокумента.ТекстовоеСодержимое + Символы.ПС;

	ТекстСообщения = ТекстСообщения + "Картинки" + Символы.ПС;
	Для Каждого Картинка Из ДокументHTML.Картинки Цикл
		ТекстСообщения = ТекстСообщения + Картинка.АльтернативныйТекст + Символы.ПС;
		ТекстСообщения = ТекстСообщения + Картинка.Источник + Символы.ПС;
	КонецЦикла;

	ТекстСообщения = ТекстСообщения + "Якоря" + Символы.ПС;
	Для Каждого Якорь Из ДокументHTML.Якоря Цикл
		ТекстСообщения = ТекстСообщения + Якорь.ТекстовоеСодержимое + Символы.ПС;
		ТекстСообщения = ТекстСообщения + Якорь.Гиперссылка + Символы.ПС;
	КонецЦикла;

	Сообщение.Текст = ТекстСообщения;
	Сообщение.Сообщить();

КонецПроцедуры
```

В этом обработчике сначала мы получаем объект `ДокументHTML` с помощью функции `ПолучитьДокументHTML()`, в которую передаем текст HTML-документа, содержащийся в реквизите Текст_HTML. Эта функция была рассмотрена выше, в листинге 6.98.

Используя свойство `ЭлементДокумента` объекта `ДокументHTML`, мы получаем корневой узел HTML-документа и выводим все его текстовое содержимое в окно сообщений пользователю.

Затем обходим в цикле коллекцию элементов всех изображений HTML-документа (`ДокументHTML.Картинки`). По мере обхода объектов `ЭлементКартинкаHTML` выводим пользователю альтернативный текст и источник (URI-адрес) каждой картинки.

Затем обходим в цикле коллекцию элементов всех якорей (гиперссылок) HTML-документа (`ДокументHTML.Якоря`). По мере обхода объектов `ЭлементЯкорьHTML` выводим пользователю текстовое содержимое и гиперссылку (адрес другого ресурса) каждой гиперссылки.

Таким образом, после загрузки HTML-документа из файла (кнопка Прочитать HTML из файла) и нажатия кнопки Разобрать HTML будет показано следующее окно сообщений пользователю (рис. 6.17).

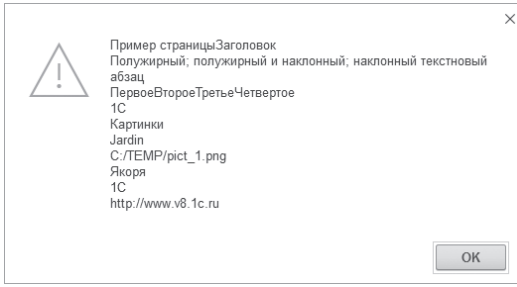


Рис. 6.17. Картинки и якоря HTML-документа

Показать структуру HTML

Предположим, нам нужно полностью обойти и показать всю иерархическую структуру узлов HTML-документа.

Для решения этой задачи добавим команду ПоказатьСтруктуруHTML. Обработчик команды заполним следующим образом (листинг 6.101).

Листинг 6.101. Обработчик команды «ПоказатьСтруктуруHTML»

```
&НаКлиенте
Процедура ПоказатьСтруктуруHTML(Команда)

	ДокументHTML = ПолучитьДокументHTML(Текст_HTML);

	Сообщение = Новый СообщениеПользователю();
	ТекстСообщения = "";

	ИтераторДерева = ДокументHTML.СоздатьОбходДерева(ДокументHTML.ЭлементДокумента);

	Пока ИтераторДерева.СледующийУзел() <> Неопределено Цикл
		Узел = ИтераторДерева.ТекущийУзел;

		ТекстСообщения = ТекстСообщения + "Имя узла: " + Узел.ИмяУзла + Символы.ПС;
		Если НЕ ПустаяСтрока(Узел.ЗначениеУзла) Тогда
			ТекстСообщения = ТекстСообщения + "Значение узла: " + Узел.ЗначениеУзла + Символы.ПС;
		КонецЕсли;
	КонецЦикла;

	Сообщение.Текст = ТекстСообщения;
	Сообщение.Сообщить();

КонецПроцедуры
```

В этом обработчике сначала мы получаем объект `ДокументHTML` с помощью функции `ПолучитьДокументHTML()`, в которую передаем текст HTML-документа, содержащийся в реквизите Текст_HTML. Эта функция была рассмотрена выше, в листинге 6.98.

Затем методом `СоздатьОбходДерева()` объекта `ДокументHTML` мы создаем иерархический итератор типа `ОбходДереваDOM` для обхода структуры дерева HTML-документа. В качестве стартового узла итератора передаем в этот метод корневой узел HTML-документа (`ДокументHTML.ЭлементДокумента`).

Методом итератора `СледующийУзел()` мы перебираем узлы HTML-документа в цикле. Используя свойство итератора `ТекущийУзел`, на каждом шаге цикла получаем текущий узел, на котором спозиционирован итератор, и выводим имя и значение узла в сообщение пользователю. В теле цикла можно проанализировать тип узла и исходя из этого проводить дальнейшую обработку его значений.

Таким образом, после загрузки HTML-документа из файла (кнопка Прочитать HTML из файла) и нажатия кнопки Показать структуру HTML будет показано следующее окно сообщений пользователю (рис. 6.18).

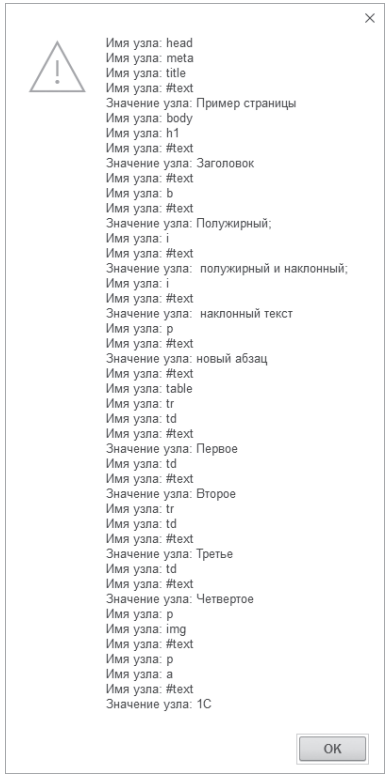


Рис. 6.18. Элементы структуры дерева HTML-документа

Изменить содержимое HTML

Предположим, нам нужно программно изменить содержимое HTML-документа, например добавить в уже существующую таблицу документа еще одну строку и заполнить значения ее столбцов.

Для решения этой задачи добавим команду ИзменитьСодержимоеHTML. Обработчик команды заполним следующим образом (листинг 6.102).

Листинг 6.102. Обработчик команды «ИзменитьСодержимоеHTML»

```
&НаКлиенте
Процедура ИзменитьСодержимоеHTML(Команда)

    ДокументHTML = ПолучитьДокументHTML(Текст_HTML);

    Узлы = ДокументHTML.ПолучитьЭлементыПоИмени("table");
    Таблица = Узлы[0];

    НоваяСтрока = ДокументHTML.СоздатьЭлемент("tr");
    Таблица.ДобавитьДочерний(НоваяСтрока);

    НоваяЯчейка = ДокументHTML.СоздатьЭлемент("td");
    НоваяЯчейка.ТекстовоеСодержимое = "Пятое";
    НоваяСтрока.ДобавитьДочерний(НоваяЯчейка);
    НоваяЯчейка = ДокументHTML.СоздатьЭлемент("td");
    НоваяЯчейка.ТекстовоеСодержимое = "Шестое";
    НоваяСтрока.ДобавитьДочерний(НоваяЯчейка);

    АдресHTML = ПолучитьТекстHTML(ДокументHTML);

КонецПроцедуры
```

В этом обработчике сначала мы получаем объект `ДокументHTML` с помощью функции `ПолучитьДокументHTML()`, в которую передаем текст HTML-документа, содержащийся в реквизите Текст_HTML. Эта функция была рассмотрена выше, в листинге 6.98.

После этого методом `ПолучитьЭлементыПоИмени(«table»)` объекта `ДокументHTML` мы получаем коллекцию элементов документа, содержащих таблицы, и из этой коллекции получаем первый узел типа `ЭлементТаблицаHTML`.

Затем методом `СоздатьЭлемент(«tr»)` объекта `ДокументHTML` мы создаем элемент документа типа `ЭлементСтрокаТаблицыHTML`. И добавляем этот узел в таблицу методом `ДобавитьДочерний()` объекта `ЭлементТаблицаHTML`. Таким образом, мы добавили новую строку в таблицу в качестве подчиненного узла.

Затем методом СоздатьЭлемент(«td») объекта ДокументHTML мы создаем элемент документа типа ЭлементЯчейкаТаблицыHTML. Устанавливаем текстовое содержимое этой ячейки и добавляем этот узел в новую строку таблицы методом ДобавитьДочерний() объекта ЭлементСтрокаТаблицыHTML. Таким образом, мы добавили новую ячейку в строку в качестве подчиненного узла.

Аналогичным образом заполняем и добавляем в строку вторую ячейку.

Теперь нам нужно получить HTML-текст результирующего документа, чтобы запомнить его реквизите АдресHTML. Для этого мы используем функцию ПолучитьТекстHTML(), в которую передаем результирующий документ. Функция была рассмотрена выше, в листинге 6.95.

Таким образом, после загрузки HTML-документа из файла (кнопка Прочитать HTML из файла) и нажатия кнопки Изменить содержимое HTML вид документа изменится следующим образом (рис. 6.19).

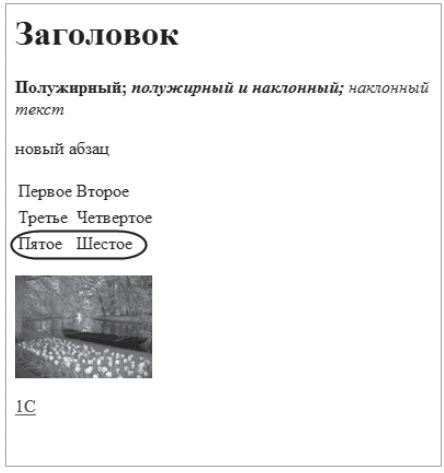


Рис. 6.19. Измененный HTML-документ

Вставить произвольный фрагмент HTML

Предположим, нам нужно добавить в HTML-документ произвольный фрагмент и отобразить результат в форме обработки.

Для решения этой задачи добавим команду ВставитьФрагментHTML. Обработчик команды заполним следующим образом (листинг 6.103).

Листинг 6.103. Обработчик команды «ВставитьФрагментHTML»

```
&НаКлиенте
Процедура ВставитьФрагментHTML(Команда)

	ДокументHTML = ПолучитьДокументHTML(Текст_HTML);

	//Узлы = ДокументHTML.ПолучитьЭлементыПоИмени("table");
	//Таблица = Узлы[0];

	Фрагмент = "<p><b>Добавленный фрагмент</b></p>
	|	<p><img src='C:/TEMP/pict.jpg' alt='Fleur'></img></p>
	|	<p><a href='http://www.its.1c.ru'>ИТС</a></p>";
	ФрагментHTML = ПолучитьДокументHTML(Фрагмент);

	Документ_HTML = ДокументHTML.ИмпортироватьУзел(ФрагментHTML.ЭлементДокумента, Истина);
	ДокументHTML.ДобавитьДочерний(Документ_HTML);
	//ДокументHTML.ВставитьПеред(Документ_HTML, Таблица);

	АдресHTML = ПолучитьТекстHTML(ДокументHTML);

КонецПроцедуры
```

В этом обработчике сначала мы получаем объект `ДокументHTML` с помощью функции `ПолучитьДокументHTML()`, в которую передаем текст HTML-документа, содержащийся в реквизите Текст_HTML. Эта функция была рассмотрена выше, в листинге 6.98.

Затем, используя ту же функцию, получаем объект `ДокументHTML` из произвольного фрагмента текста.

После этого импортируем получившийся узел в основной документ (в который мы хотим добавить фрагмент) с помощью метода `ИмпортироватьУзел()` объекта `ДокументHTML`, в который передаем корневой узел HTML-документа, созданного из фрагмента текста (`ФрагментHTML.ЭлементДокумента`). В результате основной документ устанавливается в качестве владельца этого узла, а также устанавливаются другие свойства узла.

Затем добавляем узел фрагмент в документ методом `ДобавитьДочерний()`.

Теперь нам нужно получить HTML-текст результирующего документа, чтобы запомнить его реквизите АдресHTML. Для этого мы используем функцию `ПолучитьТекстHTML()`, в которую передаем результирующий документ. Функция была рассмотрена выше, в листинге 6.95.

Таким образом, после загрузки HTML-документа из файла (кнопка Прочитать HTML из файла) и нажатия кнопки Вставить фрагмент HTML добавленный фрагмент появится в конце документа. Текст документа также изменится соответствующим образом (рис. 6.20).

Рис. 6.20. HTML-документ с добавленным фрагментом

Двоичные данные

Платформа «1С:Предприятие» предоставляет ряд низкоуровневых инструментов для манипулирования данными, представленными в виде объектов `ДвоичныеДанные`. Двоичные данные можно:

- хранить в базе данных в реквизитах вида `ХранилищеЗначения`;
- читать из файла и записывать в файл;
- передавать между клиентом и сервером при помощи временного хранилища;
- превратить в объект вида `Картинка`;
- хранить в макетах;
- шифровать и расшифровывать, подписывать и проверять подписи и др.

Работа с двоичными данными может понадобиться при реализации следующих задач:

- взаимодействие со специализированными устройствами по двоичному протоколу;
- разбор файлов и манипуляция файлами различных форматов;
- конвертация текстовых данных напрямую в двоичные данные (например, для отправки отчетов);
- работа с двоичными данными в памяти.

В рамках этой главы мы хотим лишь кратко описать основные понятия, а также показать конкретные примеры работы с двоичными данными.

ПОДРОБНЕЕ

Более подробно про работу с двоичными данными рассказано в документации «1С:Предприятия» в разделе «Глава 16. Работа с различными форматами данных – Работа с двоичными данными».

Подробнее познакомиться с примерами работы с двоичными данными можно в демонстрационной конфигурации «Примеры работы», которая прилагается к книге.

Общая информация

Для работы с двоичными данными используются такие объекты встроенного языка, как `Поток`, `ФайловыйПоток`, `ПотокВПамяти`, `ЧтениеДанных`, `ЗаписьДанных`, `БуферДвоичныхДанных` и др. С помощью этих объектов можно как организовывать последовательную работу с большими объемами двоичных данных, так и осуществлять произвольный доступ к относительно небольшим двоичным данным целиком в оперативной памяти.

Общая схема работы с двоичными данными такова: сначала данные из файла поступают в объект `ФайловыйПоток` (или `ПотокВПамяти`). Из потока с помощью объекта `ЧтениеДанных` можно либо получить объект `БуферДвоичныхДанных` (прочитать в него заданное количество байтов из потока), либо разделить поток на части одинакового размера (`РезультатЧтенияДанных`) и каждую из них обрабатывать некоторым образом.

В обратную сторону все работает аналогично: в буфер двоичных данных можно записать какие-то данные. В объект `ЗаписьДанных` можно записать либо `БуферДвоичныхДанных`, либо `РезультатЧтенияДанных`. А затем `ЗаписьДанных` можно сразу связать с потоком и с конечным файлом. Таким образом все будет записано в конечный файл.

Поясним назначение некоторых объектов.

Для работы с данными, расположенными на диске или в памяти, предназначены объекты `ФайловыйПоток` и `ПотокВПамяти`. Это специализированные версии типа `Поток`. Данный объект не имеет конструктора, а получить экземпляр объекта можно при помощи различных методов других объектов.

Потоки предназначены для последовательного чтения/записи больших объемов двоичных данных. Их преимущество заключается в том, что они позволяют работать с потоками данных произвольного объема. Но вместе с тем они предоставляют лишь базовые возможности работы, такие как чтение из потока, запись в поток и изменение текущей позиции.

Потоки можно сконструировать по имени файла или получить из других объектов, например из объектов `ДвоичныеДанные`, `HTTPЗапрос`, `ВложениеСистемыВзаимодействия` и др.

Объект ЧтениеДанных можно сконструировать на основании потока, двоичных данных или на основании имени файла. Этот объект позволяет уже читать отдельные байты, символы, числа. С его помощью можно прочитать строку с учетом кодировки или прочитать данные до некоторого известного заранее маркера. Объект ЗаписьДанных конструируется аналогичным образом, но занимается не чтением, а записью данных. Поскольку эти объекты читают/пишут данные из/в потоки, то они также делают это последовательно, что позволяет работать с потоками произвольного объема.

Объект БуферДвоичныхДанных можно получить из объекта ЧтениеДанных, записать в ЗаписьДанных. Глобальный контекст содержит также функции получения его из различных строк, из двоичных данных и др.

Главное отличие этого объекта от рассмотренных выше заключается в том, что он предоставляет не последовательный, а произвольный доступ к данным и позволяет изменять их по месту.

Все данные этого объекта полностью находятся в оперативной памяти. Поэтому, с одной стороны, он предназначен для анализа и редактирования не очень больших объемов двоичных данных. Но, с другой стороны, дает удобные возможности для произвольного чтения и записи байтов, представленных числами, для разделения буфера на несколько частей и объединения нескольких буферов в один, а также для получения части буфера указанного размера.

Также буфер двоичных данных позволяет выполнять побитовые логические операции И, ИЛИ, ИНЕ и другие. Аналогичные функции, которые позволяют удобно выполнять побитовые операции с целыми числами, есть и в глобальном контексте.

В заключение теоретической части необходимо сказать о том, что многие объекты: ДвоичныеДанные, Поток, ФайловыйПоток, ПотокВПамяти, ЧтениеДанных, ЗаписьДанных, РезультатЧтенияДанных – имеют пары синхронных и асинхронных методов. Например: Записать() – НачатьЗапись(), Закрыть() – НачатьЗакрытие() и т. п. Асинхронные методы нужны для обеспечения возможности одинаковой работы и в тонком клиенте, и в веб-клиенте. Потому что браузеры используют асинхронную модель работы.

Синхронные методы необходимы для работы в контексте сервера. Потому что на сервере используется только синхронная модель работы. Дальше в наших примерах мы будем выполнять все манипуляции с двоичными данными на сервере и, соответственно, будем использовать синхронные методы.

Примеры работы

Разбить WAV-файл на части

WAV-файл – это звуковой файл, как правило, большого размера. Наша задача – разделить этот файл на несколько частей меньшего размера. На этом примере мы наглядно покажем, как и зачем используются основные объекты для работы с двоичными данными.

Любой WAV-файл состоит из двух областей. Одна из них – заголовок файла, другая – область данных. В заголовке файла хранится информация о размере файла, количестве каналов, частоте дискретизации и др.

Чтобы один файл разделить на несколько частей, нужно разделить область данных на требуемое количество частей нужного размера, к каждой части дописать заголовок – такой же, как был у всего файла, но с другим размером данных – и затем каждую такую часть записать в отдельный WAV-файл.

Для решения этой задачи добавим команду РазделитьФайлНаФрагменты. Обработчик команды заполним следующим образом (листинг 6.104).

Листинг 6.104. Обработчик команды «РазделитьФайлНаФрагменты»

```
&НаКлиенте
Процедура РазделитьФайлНаФрагменты(Команда)

    ОповещениеОЗавершении = Новый ОписаниеОповещения(
                            "РазделитьФайлНаСервереЗавершение", ЭтотОбъект);
    НачатьПомещениеФайлаНаСервер(ОповещениеОЗавершении, , , "", , УникальныйИдентификатор);

КонецПроцедуры
```

В обработчике команды с помощью метода глобального контекста `НачатьПомещениеФайлаНаСервер()` мы начинаем помещение звукового файла из локальной файловой системы во временное хранилище. При этом пользователю предлагается диалог для выбора помещаемого файла.

В метод `НачатьПомещениеФайлаНаСервер()` первым параметром мы передаем описание оповещения, указывающее на экспортную процедуру `РазделитьФайлНаСервереЗавершение()`, которая будет выполнена, после того как выбранный файл будет помещен во временное хранилище (листинг 6.105).

Листинг 6.105. Процедура «РазделитьФайлНаСервереЗавершение()»

```
&НаКлиенте
Процедура РазделитьФайлНаСервереЗавершение(ОписаниеПомещенногоФайла, Дополнительно) Экспорт

    Если ОписаниеПомещенногоФайла = Неопределено Тогда
        Возврат;
    КонецЕсли;
```

```
    ОписаниеФайловФрагментов = РазделитьФайлНаФрагментыНаСервере(
                                ОписаниеПомещенногоФайла.Адрес);

    ФайлыФрагментов = Новый Массив;
    Для Каждого ФайлФрагмента Из ОписаниеФайловФрагментов Цикл
        ФайлыФрагментов.Добавить(
        Новый ОписаниеПередаваемогоФайла(ФайлФрагмента.ИмяФайла, ФайлФрагмента.АдресФайла));
    КонецЦикла;

    ПараметрыДиалога = Новый ПараметрыДиалогаПолученияФайлов(
                                "Выберите каталог для сохранения файлов фрагментов", Истина);

    НачатьПолучениеФайловССервера(ФайлыФрагментов, ПараметрыДиалога);

КонецПроцедуры
```

После того как файл будет помещен, адрес данных в хранилище (`ОписаниеПомещенногоФайла.Адрес`) мы передаем в серверную функцию `РазделитьФайлНаФрагментыНаСервере()`, в которой данные файла получаются из временного хранилища и производится нужная обработка и разбиение на части звукового файла (листинг 6.106).

Листинг 6.106. Функция «РазделитьФайлНаФрагментыНаСервере()»

```
&НаСервереБезКонтекста
Функция РазделитьФайлНаФрагментыНаСервере(АдресФайла)

    Сообщение = Новый СообщениеПользователю();
    РазделенныеФайлы = Новый Массив;

    ДанныеФайла = ПолучитьИзВременногоХранилища(АдресФайла);
    ЧтениеДанных = Новый ЧтениеДанных(ДанныеФайла);
    БуферЗаголовокИсходный = ЧтениеДанных.ПрочитатьВБуферДвоичныхДанных(44);

    // Декодировать заголовок файла.
    РазмерДанных        = БуферЗаголовокИсходный.ПрочитатьЦелое32(4);    // chunkSize
    КоличествоКаналов   = БуферЗаголовокИсходный.ПрочитатьЦелое16(22);   // numChannels
    ЧастотаДискретизации = БуферЗаголовокИсходный.ПрочитатьЦелое32(24);   // sampleRate
    БайтовВСекунду      = БуферЗаголовокИсходный.ПрочитатьЦелое32(28);   // byteRate

    Сообщение.Текст = "РазмерДанных: " + РазмерДанных;
    Сообщение.Сообщить();
    Сообщение.Текст = "КоличествоКаналов: " + КоличествоКаналов;
    Сообщение.Сообщить();
    Сообщение.Текст = "ЧастотаДискретизации: " + ЧастотаДискретизации;
    Сообщение.Сообщить();
    Сообщение.Текст = "БайтовВСекунду: " + БайтовВСекунду;
    Сообщение.Сообщить();

    // Получить массив фрагментов после разделения на части - массив значений типа РезультатЧтенияДанных.
    МассивФрагменты = ЧтениеДанных.РазделитьНаЧастиПо(1000);
```

```
    НомерФрагмента = 0;
    Для Каждого РезультатЧтенияФрагмент из МассивФрагменты Цикл
        БуферЗаголовокФрагмента = БуферЗаголовокИсходный.Скопировать();

        РазмерФрагмента = РезультатЧтенияФрагмент.Размер + 40;
        БуферЗаголовокФрагмента.ЗаписатьЦелое32(4, РазмерФрагмента);

        НомерФрагмента = НомерФрагмента + 1;
        ИмяФайлаФрагмента = Строка(НомерФрагмента) + ".wav";

        Поток = Новый ПотокВПамяти();
        ЗаписьДанных = Новый ЗаписьДанных(Поток);
        ЗаписьДанных.ЗаписатьБуферДвоичныхДанных(БуферЗаголовокФрагмента);
        ЗаписьДанных.Записать(РезультатЧтенияФрагмент);

        // АдресФайлаФрагмента = ПоместитьВоВременноеХранилище
        // (Новый ДвоичныеДанные(ИмяФайла));
        АдресФайлаФрагмента = ПоместитьВоВременноеХранилище(
                                Поток.ЗакрытьИПолучитьДвоичныеДанные());

        ДанныеФайлаФрагмента = Новый Структура("АдресФайла, ИмяФайла",);
        ДанныеФайлаФрагмента.АдресФайла = АдресФайлаФрагмента;
        ДанныеФайлаФрагмента.ИмяФайла = ИмяФайлаФрагмента;

        РазделенныеФайлы.Добавить(ДанныеФайлаФрагмента);

    КонецЦикла;

    Возврат РазделенныеФайлы;

КонецФункции
```

В этой функции мы сначала получаем двоичные данные по адресу во временном хранилище, переданном в функцию в параметре `АдресФайла`. И на основе этих данных создаем объект `ЧтениеДанных`. Затем, чтобы иметь возможность разобрать заголовок файла, методом этого объекта `ПрочитатьВБуферДвоичныхДанных()` читаем первые 44 байта в буфер двоичных данных.

Из получившегося буфера исходного заголовка (`БуферЗаголовокИсходный`) мы читаем параметры заголовка (размер данных, количество каналов и др.) с указанием конкретной позиции, с которой должно начинаться чтение.

Например, чтобы получить размер данных из буфера исходного заголовка, мы читаем из него 32-битное целое положительное число, начиная с четвертого байта – `БуферЗаголовокИсходный.ПрочитатьЦелое32(4)`.

На самом деле разбор заголовка файла нужен для того, чтобы вычислить, на фрагменты какого размера делить файл. Но, чтобы не усложнять пример вычислениями, мы разбираем заголовок и показываем его просто для инфор-

мации (рис. 6.21), а область данных делим на одинаковые части по 1000 байт методом `РазделитьНаЧастиПо()` объекта `ЧтениеДанных`.

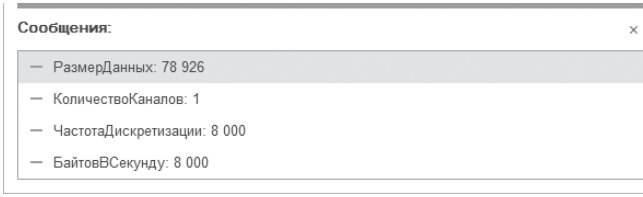


Рис. 6.21. Параметры заголовка звукового файла

Затем в цикле обхода массива значений типа `РезультатЧтенияДанных`, получившихся после деления области данных на части, мы копируем исходный заголовок методом `Скопировать()` объекта `БуферДвоичныхДанных` и записываем в него новый размер фрагмента – `БуферЗаголовокФрагмента.ЗаписатьЦелое32(4, РазмерФрагмента)`.

После этого создаем поток в памяти и объект `ЗаписьДанных` на основе этого потока. С помощью методов `ЗаписатьБуферДвоичныхДанных()` и `Записать()` этого объекта мы записываем в поток сначала заголовок фрагмента, затем часть данных. В результате получаются данные одного фрагмента файла, на который мы разделили исходный звуковой файл.

Методом `ЗакрытьИПолучитьДвоичныеДанные()` мы закрываем поток и получаем двоичные данные, которые сохраняем во временном хранилище в качестве адреса фрагмента файла.

В заключение на каждом шаге цикла мы создаем структуру `ДанныеФайлаФрагмента` с полями `ИмяФайла` и `АдресФайла` и заполняем их соответственно именем (порядковым номером фрагмента) и адресом во временном хранилище данных фрагмента файла. И затем добавляем эту структуру в массив `РазделенныеФайлы`, который и возвращает функция после завершения обхода всех фрагментов файла в процедуру `РазделитьФайлНаСервереЗавершение()`, приведенную выше (см. листинг 6.105).

После этого в этой процедуре мы обходим элементы массива `ОписаниеФайловФрагментов` и на каждом шаге цикла создаем объект `ОписаниеПередаваемогоФайла` на основе значений полей структуры, описывающей файлы фрагментов. Таким образом мы создаем массив `ФайлыФрагментов`, содержащий описания файлов, которые мы будем получать на клиенте по адресу ссылки во временном хранилище и сохранять в локальную файловую систему с именем, заданным в описании файла.

Затем мы создаем объект `ПараметрыДиалогаПолученияФайлов` и задаем заголовок и признак отображения диалога.

И передаем массив получаемых файлов и параметры диалога получения файлов в метод глобального контекста `НачатьПолучениеФайловССервера()`, с помощью которого начинается получение файлов-фрагментов и сохранение их в локальную файловую систему. При этом пользователю показывается диалог выбора каталога для сохранения файлов.

В результате в выбранном пользователем каталоге появятся звуковые файлы-фрагменты, которые можно запустить на клиенте (рис. 6.22).

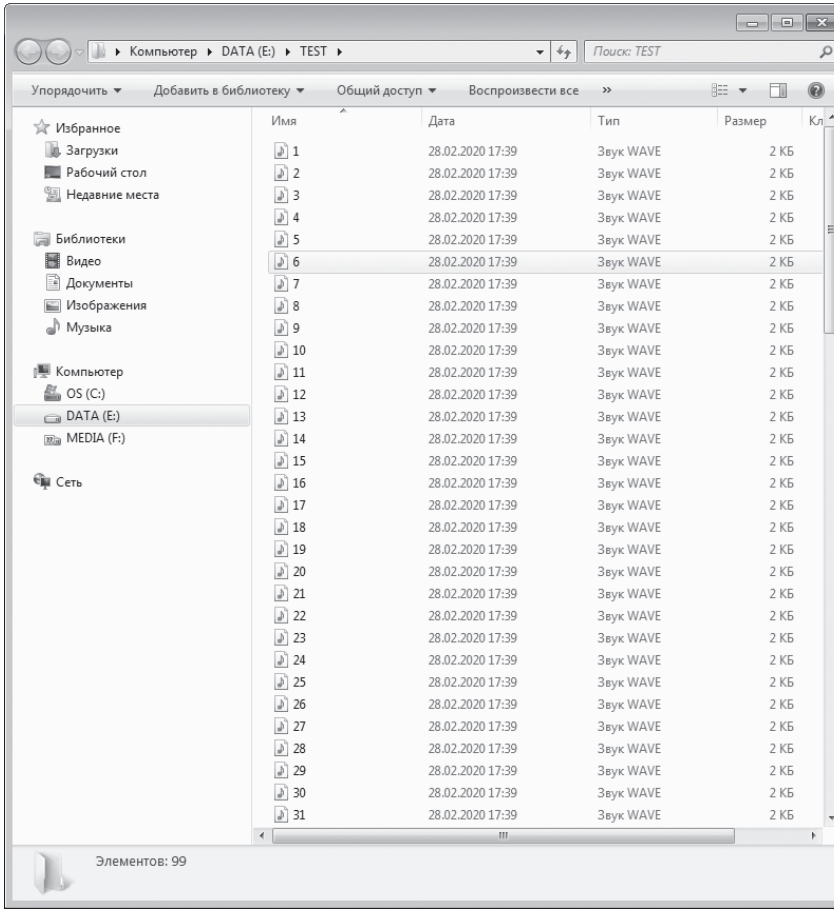


Рис. 6.22. Полученные фрагменты звукового файла

Работа с составными (multipart) HTTP-сообщениями

В данном примере мы создадим HTTP-сервис, который будет в ответ на запрос от клиента выдавать текстовое сообщение с вложенными картинками. Затем на клиенте мы отобразим полученный ответ.

Заголовок ответа сервиса Content-Type должен содержать значение «multipart/form-data». Первое слово «multipart» указывает на то, что HTTP-сообщение является составным, т. е. содержит внутри себя несколько вложенных сообщений. Второе слово – «form-data» указывает на конкретный стандарт составных сообщений, который часто используется для кодирования почтовых сообщений.

В любых составных сообщениях в заголовке Content-Type обязательно должен присутствовать атрибут «boundary», определяющий строку, которая отделяет друг от друга вложенные сообщения внутри составного сообщения.

В случае стандарта «multipart/form-data» каждое вложенное сообщение, в свою очередь, должно содержать заголовок Content-Disposition со значением «form-data» и атрибутом «name», который позволяет идентифицировать сообщения.

Создание сервиса для формирования составного сообщения

Работа с HTTP-сервисами подробно рассматривалась в первой главе в разделе «HTTP-сервисы». Поэтому в этом примере мы не будем еще раз на этом останавливаться. Поясним только моменты, касающиеся работы с двоичными данными.

Добавим в нашу демонстрационную конфигурацию новый HTTP-сервис и назовем его HTTPMultipart с корневым URL multipart. Для простоты будем считать, что наш сервис будет возвращать заданное составное сообщение в ответ на любой GET-запрос. Поэтому добавим шаблон URL с именем Тест и значением «/*». Т. е. данный шаблон соответствует любому запросу. Затем добавим в шаблон HTTP-метод GET с именем GET.

В качестве обработчика для метода создадим в модуле сервиса функцию `ТестGET()` и заполним ее следующим образом (листинг 6.107).

Листинг 6.107. Функция «ТестGET()»

```
Функция ТестGET(Запрос)

    Ответ = Новый HTTPСервисОтвет(200);

    HTTPСообщение = СоздатьСообщение();
    Ответ.Заголовки = HTTPСообщение.Заголовки;
    Ответ.УстановитьТелоИзДвоичныхДанных(HTTPСообщение.Тело);

    Возврат Ответ;

КонецФункции
```

Вся работа по созданию HTTP-сообщения выполняется в функции `СоздатьСообщение()`. Данную функцию заполним следующим образом (листинг 6.108).

Листинг 6.108. Функция «СоздатьСообщение()»

```
Функция СоздатьСообщение() Экспорт

    // Создать вложенные сообщения - текст и две картинки в виде двоичных данных.

    Текст = СоздатьСообщение_Текст("MessageText",
        "Привет, Василий!" + Символы.ПС +
        "Твой ручной лев, которого ты оставил
                                    у меня на прошлой неделе, разодрал весь мой диван." + Символы.ПС +
        "Пожалуйста забери его скорее!" + Символы.ПС +
        "Во вложении две фотки с последствиями.");

    Пингвины = СоздатьСообщение_Изображение("image1", "penguins.jpg", БиблиотекаКартинок.Пингвины);
    Коала = СоздатьСообщение_Изображение("image2", "coala.jpg", БиблиотекаКартинок.Коала);

    // Сформировать основное составное сообщение.
    Разделитель = "Asrf456BGe4h";

    Результат = Новый Структура();
    Заголовки = Новый Соответствие();

    Результат.Вставить("Заголовки", заголовки);
    Заголовки.Вставить("Content-Type", "multipart/form-data; boundary=" + Разделитель);

    Тело = Новый ПотокВПамяти();
    ЗаписьДанных = Новый ЗаписьДанных(Тело);

    ЗаписьДанных.ЗаписатьСтроку("==" + Разделитель);
    ЗаписьДанных.Записать(Текст);

    ЗаписьДанных.ЗаписатьСтроку("==" + Разделитель);
    ЗаписьДанных.Записать(Пингвины);

    ЗаписьДанных.ЗаписатьСтроку("==" + Разделитель);
    ЗаписьДанных.Записать(Коала);

    ЗаписьДанных.ЗаписатьСтроку("==" + Разделитель + "==");

    ЗаписьДанных.Закрыть();

    ДанныеТела = Тело.ЗакрытьИПолучитьДвоичныеДанные();

    Результат.Вставить("Тело", ДанныеТела);

    Возврат Результат;

КонецФункции
```

Из этой функции, формирующей составное HTTP-сообщение, возвращается структура Результат, содержащая заголовки и тело ответа сервиса, которое затем будет разобрано и показано в обработке на клиенте. Тело ответа сервиса (Результат.Тело) формируется в виде двоичных данных, разделенных разделителем, и содержит три части: текст и две картинки. Функции, возвращающие текст и картинки в виде двоичных данных: СоздатьСообщение_Текст(), СоздатьСообщение_Изображение(), – приведены ниже в листингах 6.109, 6.110.

Для записи тела сообщения создается поток в памяти, и с помощью объекта ЗаписьДанных в него записываются составные части HTTP-сообщения. Затем методом ЗакрытьИПолучитьДвоичныеДанные() поток закрывается, а двоичные данные из него сохраняются в поле Тело структуры Результат.

В заключение после возврата структуры сообщения в функцию ТестGET() (см. листинг 6.107) тело ответа сервиса устанавливается с помощью метода УстановитьТелоИзДвоичныхДанных объекта HTTPСервисОтвет.

Вспомогательную функцию СоздатьСообщение_Текст() заполним следующим образом (листинг 6.109).

Листинг 6.109. Функция «СоздатьСообщение_Текст()»

```
Функция СоздатьСообщение_Текст(ИмяСообщения, Текст)

    Поток = Новый ПотокВПамяти();
    ЗаписьДанных = Новый ЗаписьДанных(Поток);

    // Записать заголовки.
    ЗаписьДанных.ЗаписатьСтроку("Content-Disposition: form-data; name=" + ИмяСообщения);
    ЗаписьДанных.ЗаписатьСтроку("");

    // Записать тело.
    ЗаписьДанных.ЗаписатьСтроку(Текст);

    записьДанных.Закрыть();

    Возврат Поток.ЗакрытьИПолучитьДвоичныеДанные();

КонецФункции
```

Вспомогательную функцию СоздатьСообщение_Изображение() заполним следующим образом (листинг 6.110).

Листинг 6.110. Функция «СоздатьСообщение_Изображение()»

```
Функция СоздатьСообщение_Изображение(ИмяСообщения, ИмяФайла, Картинка)

	Поток = Новый ПотокВПамяти();
	ЗаписьДанных = Новый ЗаписьДанных(Поток);

	// Записать заголовки.
	ЗаписьДанных.ЗаписатьСтроку("Content-Disposition: form-data; name="
	                            + ИмяСообщения + "; filename=" + ИмяФайла);
	ЗаписьДанных.ЗаписатьСтроку("Content-Type: image/jpeg");
	ЗаписьДанных.ЗаписатьСтроку("");

	// Записать тело.
	ЗаписьДанных.Записать(Картинка.ПолучитьДвоичныеДанные());

	ЗаписьДанных.Закрыть();

	Возврат Поток.ЗакрытьИПолучитьДвоичныеДанные();

КонецФункции
```

Осталось только опубликовать наш HTTP-сервис на веб-сервере.

Разбор составного сообщения на стороне клиента

Теперь посмотрим, как мы можем работать с составными сообщениями на стороне клиента. Нам необходимо получить ответ HTTP-сервиса, распаковать вложенные в ответ сообщения и показать их содержимое.

В форму нашей демонстрационной обработки добавим реквизиты типа `Строка`:

- HTTPСообщение – текст сообщения;
- АдресКартинки1 – адрес первой картинки во временном хранилище;
- АдресКартинки2 – адрес второй картинки во временном хранилище.

Перетащим их в окно элемента формы и свяжем их с соответствующими элементами управления:

- Поле текстового документа (реквизит HTTPСообщение).
- Поле картинки (реквизит АдресКартинки1).
- Поле картинки (реквизит АдресКартинки2).

Также добавим в форму команду ОтправитьHTTPЗапрос. Из обработчика этой команды мы вызываем серверную процедуру `Выполнить _ HTTPЗапрос()`, в которой и происходят получение, разбор и отображение ответа HTTP-сервиса (листинг 6.111).

Листинг 6.111. Процедура «Выполнить_HTTPЗапрос()»

```
&НаСервере
Процедура Выполнить_HTTPЗапрос()

	HTTPСоединение = Новый HTTPСоединение("localhost");

	// Сформировать запрос, отправить на сервер и получить ответ.
	Запрос = Новый HTTPЗапрос("Web_1C/hs/multipart");
	Ответ = HTTPСоединение.Получить(Запрос);

	// Разобрать ответ на составные части.
	// Результат представляет собой структуру, содержащую текст
	// и картинки из составного HTTP-сообщения.
	Результат = ПрочитатьСообщение(Ответ.Заголовки, Ответ.ПолучитьТелоКакДвоичныеДанные());
	HTTPСообщение = Результат.Сообщение;

	// Создать объекты Картинка из двоичных данных изображений, полученных с сервера.
	Картинка1 = Новый Картинка(Результат.Картинка1);
	Картинка2 = Новый Картинка(Результат.Картинка2);

	// Поместить полученные картинки во временное хранилище.
	АдресКартинки1 = ПоместитьВоВременноеХранилище(Картинка1);
	АдресКартинки2 = ПоместитьВоВременноеХранилище(Картинка2);

КонецПроцедуры
```

Отправка запросов и получение ответов от HTTP-сервисов подробно рассматривались в первой главе в разделе «Обращение к HTTP-сервисам». Поэтому в данном примере мы не будем еще раз на этом останавливаться. Поясним только моменты, касающиеся работы с двоичными данными.

После отправки запроса к нашему HTTP-сервису, определенному ранее, мы вызываем функцию `ПрочитатьСообщение()` и передаем туда заголовки и тело ответа, полученное методом `ПолучитьТелоКакДвоичныеДанные()`. В этой функции и выполняется разбор полученного от сервиса составного сообщения (листинг 6.112).

Листинг 6.112. Функция «ПрочитатьСообщение()»

```
&НаСервереБезКонтекста
Функция ПрочитатьСообщение(Заголовки, Тело)

	Разделитель = ПолучитьРазделительСоставногоСообщения(Заголовки);

	Маркеры = Новый Массив();
	Маркеры.Добавить("==" + Разделитель);
	Маркеры.Добавить("==" + Разделитель + Символы.ПС);
	маркеры.Добавить("==" + Разделитель + Символы.ВК);
	Маркеры.Добавить("==" + Разделитель + Символы.ВК + Символы.ПС);
	Маркеры.Добавить("==" + Разделитель + "==");
```

```
    Текст = Неопределено;
    Изображение1 = Неопределено;
    Изображение2 = Неопределено;

    ЧтениеДанных = Новый ЧтениеДанных(Тело);

    // Перейти к началу первой части.
    ЧтениеДанных.ПропуститьДо(маркеры);

    // Прочитать в цикле все части тела сообщения.
    Пока Истина Цикл
        Часть = чтениеДанных.ПрочитатьДо(Маркеры);

        Если Не Часть.МаркерНайден Тогда
            // Неправильно сформированное сообщение
            Прервать;
        КонецЕсли;

        ЧтениеЧасти = Новый ЧтениеДанных(Часть.ОткрытьПотокДляЧтения());
        ЗаголовкиЧасти = ПрочитатьЗаголовки(ЧтениеЧасти);
        ИмяЧасти = ПолучитьИмяСообщения(ЗаголовкиЧасти);

        Если ИмяЧасти = "MessageText" Тогда
            Текст = ЧтениеЧасти.ПрочитатьСимволы();
        ИначеЕсли имяЧасти = "image1" Тогда
            Изображение1 = ЧтениеЧасти.Прочитать().ПолучитьДвоичныеДанные();
        ИначеЕсли имяЧасти = "image2" Тогда
            Изображение2 = ЧтениеЧасти.Прочитать().ПолучитьДвоичныеДанные();
        КонецЕсли;

        Если Часть.ИндексМаркера = 4 Тогда
            // Прочитана последняя часть
            Прервать;
        КонецЕсли;
    КонецЦикла;

    Возврат Новый Структура("Сообщение,Картинка1,Картинка2", Текст, Изображение1, Изображение2);

КонецФункции
```

В этой функции мы сначала получаем разделитель составного сообщения из заголовков сообщения с помощью функции `ПолучитьРазделительСоставногоСообщения()`, приведенной в листинге 6.113.

Затем формируем массив маркеров, описывающих все возможные маркеры, разделяющие составное сообщение на части.

После этого создаем объект `ЧтениеДанных` на основе тела ответа, переданного в функцию в виде двоичных данных, и методом `ПропуститьДо()` позиционируемся на чтении первой части сообщения. В цикле методом `ПрочитатьДо()` объекта `ЧтениеДанных` мы читаем сообщение по частям вплоть до последнего маркера.

Для каждой прочитанной части мы получаем заголовки этой части сообщения в виде объекта `Соответствие` с помощью функции `ПрочитатьЗаголовки()`, приведенной в листинге 6.114. А также получаем имя вложенного сообщения из заголовков этой части сообщения с помощью функции `ПолучитьИмяСообщения()`, приведенной в листинге 6.115.

Затем в зависимости от имени прочитанной части получаем либо текст, либо двоичные данные картинок. И после окончания чтения всех частей сообщения упаковываем эти значения в структуру, которую функция возвращает в процедуру `Выполнить _ HTTPЗапрос()` – см. листинг 6.111. После чего в этой процедуре текст и картинки распаковываются и отображаются в соответствующих элементах формы обработки.

Вспомогательную функцию `ПолучитьРазделительСоставногоСообщения()` заполним следующим образом (листинг 6.113).

Листинг 6.113. Функция «СоздатьСообщение_Изображение()»

```
&НаСервереБезКонтекста
Функция ПолучитьРазделительСоставногоСообщения(Заголовки)

	ТипСодержимого = Заголовки.Получить("Content-Type");

	Свойства = СтрРазделить(ТипСодержимого, ";", Ложь);
	Граница = Неопределено;

	Для Каждого Свойство Из Свойства Цикл
		Части = СтрРазделить(Свойство, "=", Ложь);
		ИмяСвойства = СокрЛП(Части[0]);

		Если ИмяСвойства <> "boundary" Тогда
			Продолжить;
		КонецЕсли;

		Граница = СокрЛП(Части[1]);
		Прервать;
	КонецЦикла;

	Возврат Граница;

КонецФункции
```

При поиске строки-разделителя составного сообщения из заголовков предполагается, что значение разделителя задается в заголовке Content-Type в виде `Content-Type: multipart/form-data; boundary=<Разделитель>`.

Для разбора строк во всех вспомогательных функциях используется функция глобального контекста `СтрРазделить()`.

Вспомогательную функцию `ПрочитатьЗаголовки()` заполним следующим образом (листинг 6.114).

Листинг 6.114. Функция «ПрочитатьЗаголовки()»

```
&НаСервереБезКонтекста
Функция ПрочитатьЗаголовки(Чтение)

	Заголовки = Новый Соответствие();

	Пока Истина Цикл
		Строка = Чтение.ПрочитатьСтроку();
		Если Строка = "" Тогда
			Прервать;
		КонецЕсли;

		Части = СтрРазделить(Строка, ":");
		ИмяЗаголовка = СокрЛП(Части[0]);
		Значение = СокрЛП(Части[1]);

		Заголовки.Вставить(ИмяЗаголовка, Значение);
	КонецЦикла;

	Возврат Заголовки;

КонецФункции
```

Вспомогательную функцию `ПолучитьИмяСообщения()` заполним следующим образом (листинг 6.115).

Листинг 6.115. Функция «ПолучитьИмяСообщения()»

```
&НаСервереБезКонтекста
Функция ПолучитьИмяСообщения(Заголовки)

	Описание = Заголовки.Получить("Content-Disposition");

	Свойства = СтрРазделить(Описание, ";", Ложь);
	Имя = Неопределено;

	Для Каждого Свойство Из Свойства Цикл
		Части = СтрРазделить(Свойство, "=", Ложь);
		ИмяСвойства = СокрЛП(Части[0]);

		Если ИмяСвойства <> "name" Тогда
			Продолжить;
		КонецЕсли;

		Имя = СокрЛП(Части[1]);
		Прервать;
	КонецЦикла;

	Возврат Имя;

КонецФункции
```

В этой функции имя сообщения получается из заголовка Content-Disposition в виде `Content-Disposition: form-data; name=<Имя сообщения>`.

XDTO-сериализация

Механизм XDTO является универсальным способом представления данных для интеграции с другими системами. Аббревиатура XDTO расшифровывается как XML Data Transfer Objects. XDTO является механизмом объектного моделирования данных, описываемых с помощью схемы XML.

Можно выделить несколько задач, для решения которых используется механизм XDTO:

- обмен данными между конфигурациями «1С:Предприятия» с разными структурами данных;
- обмен данными на основе схем XML, не привязанных к той или иной конфигурации (например, обмен с информационными системами, построенными не на основе системы «1С:Предприятие»);
- создание собственной системы типов и значений для обработки произвольных данных;
- описание типов параметров и возвращаемых значений Web-сервисов.

В данном разделе мы остановимся на вопросе сериализации данных на основе механизма XDTO. Использование механизма для описания типов параметров и возвращаемых значений Web-сервисов рассмотрено в главе «Использование XDTO для описания типов параметров и возвращаемых значений Web-сервисов».

В настоящее время обмен с различными программными системами реализуется с использованием XML. Механизм XDTO позволяет создавать требуемые для обмена схемы XML и формировать XML-документы, удовлетворяющие этим схемам.

Основным понятием, на котором строится механизм XDTO, является фабрика XDTO. Фабрика XDTO содержит описание всех типов, с которыми оперирует система. В частности, при создании новой информационной базы «1С:Предприятия» автоматически создается глобальная фабрика XDTO, которая описывает все типы, используемые в конфигурации. Эта фабрика доступна через свойство глобального контекста `ФабрикаXDTO`.

В общем случае фабрика XDTO создается на основании описаний всех типов, которые зарегистрированы в фабрике. Для создания фабрики XDTO средствами встроенного языка используется конструктор объекта `ФабрикаXDTO`, которому передается набор схем XML, содержащийся в объекте `НаборСхемXML`.

В системе «1С:Предприятие» реализована сериализация данных на основе механизма XDTO, которая позволяет сериализовать в/из XML все типы данных, хранящиеся в базе данных.

XDTO-сериализация предназначена для сохранения данных объекта в файл XML и создания объекта на основе данных, хранящихся в файле XML. Для этого используется объект `СериализаторXDTO`, который может быть получен с помощью конструктора на основе существующей фабрики XDTO.

Например, сериализация ссылки на элемент справочника Товары в файл XML может быть выполнена с помощью следующего программного кода (листинги 6.116, 6.117). При этом так же, как это подробно описано в разделе «Работа с простыми типами», передача XML-документа между клиентом и сервером происходит через временное хранилище, поэтому мы не будем еще раз на этом останавливаться.

Листинг 6.116. Сериализация ссылки на элемент справочника «Товары» в файл XML

```
&НаКлиенте
Процедура СериализацияВXML(Команда)

	АдресДокументаВХранилище = СериализацияВXMLНаСервере();
	НачатьПолучениеФайлаССервера(, АдресДокументаВХранилище, "c:\temp\exchange.xml");

КонецПроцедуры
```

Листинг 6.117. Функция «СериализацияВXMLНаСервере()»

```
&НаСервереБезКонтекста
Функция СериализацияВXMLНаСервере()

	// Создать сериализатор XDTO для глобальной фабрики XDTO.
	НовыйСериализаторXDTO = Новый СериализаторXDTO(ФабрикаXDTO);

	// Создать объект записи XML и открыть файл.
	ИмяФайлаXML = КаталогВременныхФайлов() + "temp.xml";

	НоваяЗаписьXML = Новый ЗаписьXML;
	НоваяЗаписьXML.ОткрытьФайл(ИмяФайлаXML);

	// Получить ссылку на элемент справочника Товары.
	ССылкаНаЭлементСправочника = Справочники.Товары.НайтиПоКоду("000000003");

	// Сериализовать ссылку в XML.
	НовыйСериализаторXDTO.ЗаписатьXML(НоваяЗаписьXML
	                                  , ССылкаНаЭлементСправочника, НазначениеТипаXML.Явное);
	НоваяЗаписьXML.Закрыть();

	Возврат ПоместитьВоВременноеХранилище(Новый ДвоичныеДанные(ИмяФайлаXML));

КонецФункции
```

В функции `СериализацияВXMLНаСервере()` создается объект `СериализаторXDTO` и методом этого объекта `ЗаписатьXML()` производится сериализация ссылки на элемент справочника Товары во временный файл, адрес которого во временном хранилище функция возвращает на клиент.

Сериализация ссылки на элемент справочника Товары из файла XML может быть выполнена с помощью следующего программного кода (листинги 6.118, 6.119).

Листинг 6.118. Сериализация ссылки на элемент справочника «Товары» из файла XML

```
&НаКлиенте
Процедура СериализацияИзXML(Команда)

    АдресВременногоХранилища = "";
    ОповещениеОЗавершении = Новый ОписаниеОповещения(
                                "СериализацияИзXMLЗавершение", ЭтотОбъект);
    НачатьПомещениеФайлаНаСервер(ОповещениеОЗавершении, , ,
                                АдресВременногоХранилища, "c:\temp\exchange.xml",
        УникальныйИдентификатор);

КонецПроцедуры
```

Листинг 6.119. Обработчик оповещения «СериализацияИзXMLЗавершение()»

```
&НаКлиенте
Процедура СериализацияИзXMLЗавершение(ОписаниеПомещенногоФайла, Дополнительно) Экспорт

    СериализацияИзXMLНаСервере(ОписаниеПомещенногоФайла.Адрес);

КонецПроцедуры
```

В обработчике оповещения `СериализацияИзXMLЗавершение()` вызывается серверная процедура `СериализацияИзXMLНаСервере()`, в которую передается адрес помещенного файла, содержащего сериализованное значение ссылки на элемент справочника Товары (листинг 6.120).

Листинг 6.120. Процедура «СериализацияИзXMLНаСервере()»

```
&НаСервереБезКонтекста
Процедура СериализацияИзXMLНаСервере(АдресДокументаВХранилище)

    // Создать сериализатор XDTO для глобальной фабрики XDTO.
    НовыйСериализаторXDTO = Новый СериализаторXDTO(ФабрикаXDTO);

    // Прочитать данные объекта XDTO из файла XML
    ДанныеДокумента = ПолучитьИзВременногоХранилища(АдресДокументаВХранилище);
    ИмяФайлаXML = КаталогВременныхФайлов() + "temp.xml";
    ДанныеДокумента.Записать(ИмяФайлаXML);
```

```
    НовоеЧтениеXML = Новый ЧтениеXML;
    НовоеЧтениеXML.ОткрытьФайл(ИмяФайлаXML);

    // Сериализовать ссылку из XML.
    ССылкаНаЭлементСправочника = НовыйСериализаторXDTO.ПрочитатьXML(НовоеЧтениеXML);

    ЭлементСправочника = ССылкаНаЭлементСправочника.ПолучитьОбъект();
    ЭлементСправочника.Наименование = ЭлементСправочника.Наименование
                        + " - сериализован из XML " + Строка(ТекущаяДата());
    ЭлементСправочника.Записать();

    НовоеЧтениеXML.Закрыть();

КонецПроцедуры
```

В этой процедуре данные файла получаются из временного хранилища, сохраняются во временный файл и происходит чтение сериализованного значения методом `ПрочитатьXML()` объекта `СериализаторXDTO`. После этого по прочитанной ссылке получается объект − элемент справочника. И затем этот элемент редактируется и записывается.

ZIP-архивы

При решении задач обмена может потребоваться передача довольно большого объема данных – как посредством каких-либо интернет-технологий, так и другими средствами. В связи с объемом этих данных может возникнуть необходимость в их сжатии перед передачей и распаковке после приема. Платформа «1С:Предприятие» предоставляет возможности работы с ZIP-архивами. Для этого в системе существуют объекты `ЗаписьZIPФайла` (отвечает за запись) и `ЧтениеZIPФайла` (отвечает за чтение архивов).

Создание архива

Для того чтобы записать файлы в ZIP-архив, необходимо выполнить несколько простых действий:

- создать архив с необходимыми параметрами, в который будут помещаться файлы;
- поместить в архив необходимые файлы;
- записать архив.

Рассмотрим эти действия на следующем примере (листинг 6.121).

Листинг 6.121. Пример записи файла в ZIP-архив

```
Архив = Новый ЗаписьZIPФайла("e:\test\архив.zip", "", "", МетодСжатияZIP.Сжатие
                                    , УровеньСжатияZIP.Максимальный,
        МетодШифрованияZIP.Zip20, КодировкаИменФайловВZipФайле.UTF8);

Архив.Добавить("c:\temp\document.xml", РежимСохраненияПутейZIP.НеСохранятьПути);
Архив.Добавить("c:\temp\index.html", РежимСохраненияПутейZIP.СохранятьОтносительныеПути);

Архив.Записать();
```

Создание объекта `ЗаписьZIPФайла` можно осуществить двумя путями:

- создать инициализированный объект (см. листинг 6.121);
- создать неинициализированный объект и вызвать у него метод `Открыть()` (файл архива не должен существовать в этом случае) – листинг 6.122.

Листинг 6.122. Пример использования неинициализированного объекта «ЗаписьZIPФайла»

```
Архив = Новый ЗаписьZIPФайла();
Архив.Открыть("e:\test\архив.zip", "", "", МетодСжатияZIP.Сжатие, УровеньСжатияZIP.Максимальный,
        МетодШифрованияZIP.Zip20, КодировкаИменФайловВZipФайле.UTF8);
```

При создании нового архива (либо конструктором, либо методом `Открыть()`) требуется указать:

- Имя файла, куда будет записан архив. Этот параметр является обязательным. Если такой файл уже существует на диске, он будет перезаписан (в момент вызова метода `Записать()`).
- Пароль доступа к архиву. Если этот параметр пропущен или равен пустой строке, то шифрование производиться не будет.
- Комментарий к архиву.
- Метод сжатия файлов в архиве. На выбор предоставляется возможность скопировать файлы в архив без сжатия или сжать их. По умолчанию файлы сжимаются.
- Уровень сжатия файлов в архиве. Можно выбирать между минимальным, оптимальным и максимальным сжатием. По умолчанию используется оптимальное сжатие.
- Метод шифрования. Можно защитить архив методом шифрования `ZIP20`, совместимым с большинством программ, или с помощью шифрования на основе новейшего стандарта AES с различной длиной ключа (128, 192 и 256 бит). Однако следует помнить, что данный метод может быть несовместим с некоторыми программами архивирования старых версий.

- Способ кодировки имен файлов внутри ZIP-файла. Указанный способ кодировки имеет приоритет над значением параметра `FileNamesEncodingInZipFile` файла conf.cfg.

После создания объекта необходимо добавить в него сжимаемые файлы. Для этой цели используется метод `Добавить()`. У него 3 параметра:

- Полное имя файла или маска.
- Режим сохранения путей к файлу. Можно сохранять полные пути, не сохранять пути совсем или сохранять пути относительно каталога.
- Режим обработки подкаталогов. Можно обрабатывать подкаталоги рекурсивно или не обрабатывать их. Параметр имеет смысл, если в качестве имени указана маска.

После того как все необходимые файлы добавлены, можно записать архив на диск, воспользовавшись методом `Записать()`.

Важно понимать, что до выполнения этого метода никаких реальных действий по созданию архива не происходит. После записи архива на диск объект закрывает его, и для работы со следующим архивом необходимо выполнить метод `Открыть()`.

Особенности упаковки файлов по маске

Остановимся подробнее на особенностях упаковки файлов по маске.

Предположим, что у нас есть следующие файлы:

```
c:\ZIP\file1.xls
c:\ZIP\file2.xls
c:\ZIP\file.doc
c:\ZIP\Test\file3.xls
c:\ZIP\Test\file4.xls
```

Для упаковки всех этих файлов в архив можно воспользоваться следующим кодом (листинг 6.123).

Листинг 6.123. Пример добавления файлов в ZIP-архив

```
Архив.Добавить("c:\ZIP\file1.xls", РежимСохраненияПутейZIP.СохранятьПолныеПути);
Архив.Добавить("c:\ZIP\Test\file3.xls", РежимСохраненияПутейZIP.СохранятьОтносительныеПути);
Архив.Добавить("c:\ZIP\Test\file4.xls", РежимСохраненияПутейZIP.НеСохранятьПути);
// и так далее...
```

Следует отметить, что относительные пути актуальны только при задании маски (значение `СохранятьОтносительныеПути` системного перечисления `РежимСохраненияПутейZIP`).

В архив файлы попадут следующим образом:

```
c:\ZIP\file1.xls
\Test\file3.xls
\file4.xls
```

Добавить файлы в архив можно и другим способом – по маске помещаемых файлов (листинг 6.124).

Листинг 6.124. Пример добавления файлов в ZIP-архив

```
Архив.Добавить("c:\ZIP\*.xls", РежимСохраненияПутейZIP.СохранятьОтносительныеПути,
        РежимОбработкиПодкаталоговZIP.НеОбрабатывать);
```

В результирующий архив попадут следующие файлы:

```
\file1.xls
\file2.xls
```

При создании архива можно включать файлы во вложенных каталогах (листинг 6.125).

Листинг 6.125. Пример добавления файлов из вложенных каталогов в ZIP-архив

```
Архив.Добавить("c:\ZIP\*.xls", РежимСохраненияПутейZIP.СохранятьОтносительныеПути,
        РежимОбработкиПодкаталоговZIP.ОбрабатыватьРекурсивно);
```

В результирующий архив попадут следующие файлы:

```
\file1.xls
\file2.xls
\Test\file3.xls
\Test\file4.xls
```

Особенности работы метода «Добавить()»

Упаковать файлы в архив можно несколькими способами в зависимости от того, какой результат необходим.

Упаковать некоторые файлы в папку ZIP с сохранением структуры каталогов.

Предположим, что у нас есть следующие файлы:

```
c:\ZIP\file1.xls
c:\ZIP\Test\file3.xls
```

Добавить их в архив можно, вызывая для каждого метод `Добавить()` с указанием требуемых параметров (листинг 6.126).

Листинг 6.126. Пример добавления файлов в ZIP-архив

```
Архив.Добавить("c:\ZIP\file1.xls", РежимСохраненияПутейZIP.СохранятьПолныеПути);
Архив.Добавить("c:\ZIP\Test\file3.xls", РежимСохраненияПутейZIP.СохранятьПолныеПути );
```

В архив файлы попадут следующим образом:

```
\ZIP\file1.xls
\ZIP\Test\file3.xls
```

Упаковать все файлы в папку ZIP без сохранения структуры каталогов.

Предположим, что у нас есть следующие файлы:

```
c:\ZIP\file1.xls
c:\ZIP\Test\file3.xls
```

Для добавления всех файлов передадим в метод `Добавить()` маску для поиска всех файлов (*.*), укажем, что нет необходимости сохранять пути для файлов и что требуется производить поиск в подпапках (листинг 6.127).

Листинг 6.127. Пример добавления файлов в ZIP-архив

```
Архив.Добавить("c:\ZIP\*.*", РежимСохраненияПутейZIP.НеСохранятьПути, РежимОбработкиПодкаталоговZIP.
        ОбрабатыватьРекурсивно);
```

В архив файлы попадут следующим образом:

```
\file1.xls
\file3.xls
```

При этом следует помнить, что если в разных папках будут файлы с одинаковыми именами, то операция выполнена не будет – из-за того что фактически мы попытаемся добавить в архив файлы с одинаковыми именами.

Упаковать все файлы в папку ZIP с сохранением относительных путей.

Предположим, что у нас есть следующие файлы:

```
c:\ZIP\file1.xls
c:\ZIP\Test\file3.xls
```

Для добавления всех файлов передадим в метод `Добавить()` маску для поиска всех файлов (*.*), укажем, что необходимо сохранять относительные пути для файлов и что требуется производить поиск в подпапках. При этом следует помнить, что относительные пути будут начинаться от папки, в которой начался поиск (листинг 6.128).

Листинг 6.128. Пример добавления файлов в ZIP-архив

```
Архив.Добавить("c:\ZIP\*.*", РежимСохраненияПутейZIP.СохранятьОтносительныеПути,
        РежимОбработкиПодкаталоговZIP.ОбрабатыватьРекурсивно);
```

В архив файлы попадут следующим образом:

```
\file1.xls
\Test\file3.xls
```

Чтение ZIP-архивов

На приемной стороне при получении архива возникает задача его чтения. С точки зрения объекта `ЧтениеZipФайла` необходимо выполнить два действия:

- открыть полученный архив;
- распаковать файлы (извлечь их из архива).

Создание объекта `ЧтениеZipФайла` может производиться двумя способами:

- создать инициализированный объект (листинг 6.129);
- создать неинициализированный объект (листинг 6.130).

Листинг 6.129. Пример использования объекта «ЧтениеZipФайла»

```
Архив = Новый ЧтениеZipФайла("c:\архив.zip", "Пароль");
```

Листинг 6.130. Пример использования объекта «ЧтениеZipФайла»

```
Архив=Новый ЧтениеZipФайла();
Архив.Открыть("c:\архив.zip", "Пароль");
```

Извлечение файлов из архива может осуществляться двумя методами:

- метод `ИзвлечьВсе()` (листинг 6.131);
- метод `Извлечь()` (каждый файл по отдельности) – листинг 6.132.

Листинг 6.131. Пример извлечения файлов из ZIP-архива

```
Архив.ИзвлечьВсе("c:\Temp", РежимВосстановленияПутейФайловZIP.НеВосстанавливать);
```

Листинг 6.132. Пример извлечения файлов из ZIP-архива

```
Для Каждого Элемент Из Архив.Элементы Цикл
        Архив.Извлечь(Элемент, "c:\Temp", РежимВосстановленияПутейФайловZIP.НеВосстанавливать,
                        ?(Элемент.Зашифрован, Пароль, ""));
КонецЦикла;
```

При взгляде на приведенный код может возникнуть вопрос: зачем же указывать пароль для расшифровки, если мы уже указали его при открытии? Суть в том, что формат ZIP-архива позволяет указать произвольный пароль для любого файла и многие популярные программы архивирования поддерживают такую возможность. При указании индивидуального пароля распаковки будет использоваться он. А если индивидуальный пароль не указан, то используется пароль, указанный при открытии архива.

После извлечения файлов для прекращения работы с архивом необходимо выполнить метод `Закрыть()`.

Работа с файлами большого объема

При работе с большими объемами данных могут возникнуть проблемы при пересылке архивных файлов. Типичной такой проблемой является ограничение некоторых почтовых серверов на размер сообщения: если письмо превышает некий заранее установленный размер, оно отбрасывается сервером. Применительно к «1С:Предприятию» такая ситуация может возникнуть при пересылке больших сообщений обмена данными.

Для решения этих проблем можно использовать функции глобального контекста `РазделитьФайл()` и `ОбъединитьФайлы()`.

Функция `РазделитьФайл()` предназначена для разбиения файла на несколько частей указанного размера. Функция `ОбъединитьФайлы()`, наоборот, позволяет из нескольких томов собрать один файл.

Пример использования функции `РазделитьФайл()` приведен в листинге 6.133.

Листинг 6.133. Пример использования функции «РазделитьФайл()»

```
Массив = РазделитьФайл("c:\архив.zip", 1024 * 1024, "c:\");
```

Первый параметр функции содержит имя архива, во второй записывается размер каждой части файла в байтах, в третьем параметре указывается путь, по которому будут размещаться создаваемые файлы. Функция возвращает массив имен созданных файлов.

Функцию `ОбъединитьФайлы()` можно использовать двумя способами:

- указывая в первом параметре массив файлов, которые необходимо объединить (листинг 6.134);
- указывая в первом параметре шаблон, в соответствии с которым будет производиться объединение (листинг 6.135).

Листинг 6.134. Пример использования функции «ОбъединитьФайлы()»

```
МассивИмен = Новый Массив(2);
МассивИмен.Добавить("c:\архив.zip.001");
МассивИмен.Добавить("c:\архив.zip.002");
ОбъединитьФайлы(МассивИмен, "c:\архив.zip");
```

Листинг 6.135. Пример использования функции «ОбъединитьФайлы()»

```
ОбъединитьФайлы("c:\архив.zip.*", "c:\архив.zip");
```

Во втором параметре функции указывается полное имя создаваемого в результате объединения файла.

Кроме этого способа существует также возможность объединить файлы в один с помощью команды операционной системы `COPY`. Для этого нужно воспользоваться ее возможностью конкатенации двоичных файлов (листинг 6.136).

Листинг 6.136. Пример использования команды COPY

```
C:\>COPY архив.zip.001 /B + архив.zip.002 /B архив.zip
```

Примеры работы

Ниже мы рассмотрим примеры записи и чтения файлов в/из архивов, использующие современные методы передачи файлов на сервер для их дальнейшей обработки и затем получения их с сервера. А также покажем возможность работы с архивами с помощью двоичных данных.

Поскольку обмен файлами между клиентом и сервером через временное хранилище, а также работа с двоичными данными отдельно подробно рассматривались в соответствующих разделах книги, в этих примерах мы будем пояснять только новые, не описанные ранее возможности работы.

Добавление в архив файлов по маске

Рассмотрим пример добавления в архив файлов из выбранного пользователем каталога и соответствующих маске (в данном случае – «*.jpg»). Данная задача может быть выполнена с помощью следующего программного кода (листинги 6.137, 6.138, 6.139, 6.140).

Листинг 6.137. Пример добавления файлов в ZIP-архив

```
&НаКлиенте
Процедура ДобавлениеФайловВАрхив(Команда)

	ВыбратьКаталог("Выбор каталога для поиска файлов",
		Новый ОписаниеОповещения("ДобавитьФайлыВАрхивЗавершение", ЭтотОбъект));

КонецПроцедуры
```

Листинг 6.138. Процедура «ВыбратьКаталог()»

```
&НаКлиенте
Процедура ВыбратьКаталог(ЗаголовокДиалога, Оповещение)

	Диалог = Новый ДиалогВыбораФайла(РежимДиалогаВыбораФайла.ВыборКаталога);
	Диалог.Заголовок = ЗаголовокДиалога;
	Диалог.Показать(Оповещение);

КонецПроцедуры
```

Листинг 6.139. Обработчик оповещения «ДобавитьФайлыВАрхивЗавершение()»

```
&НаКлиенте
Процедура ДобавитьФайлыВАрхивЗавершение(ВыбранныеФайлы, Дополнительно) Экспорт

	Если ВыбранныеФайлы = Неопределено Тогда
		Возврат;
	КонецЕсли;

	МаскаПоиска = ВыбранныеФайлы[0] + "\*.jpg";

	АдресАрхиваВХранилище = ДобавитьФайлыВАрхивНаСервере(МаскаПоиска);

	НачатьПолучениеФайлаССервера(, АдресАрхиваВХранилище, "c:\temp\pictures.zip");

КонецПроцедуры
```

Листинг 6.140. Функция «ДобавитьФайлыВАрхивНаСервере()»

```
&НаСервереБезКонтекста
Функция ДобавитьФайлыВАрхивНаСервере(МаскаПоиска)

	Архив = Новый ЗаписьZIPФайла();
```

```
    Архив.Добавить(МаскаПоиска,
        РежимСохраненияПутейZIP.СохранятьОтносительныеПути, РежимОбработкиПодкаталоговZIP.
            ОбрабатыватьРекурсивно);

    ДанныеАрхива = Архив.ПолучитьДвоичныеДанные("", "",
        МетодСжатияZIP.Сжатие, УровеньСжатияZIP.Максимальный, МетодШифрованияZIP.Zip20);

    Возврат ПоместитьВоВременноеХранилище(ДанныеАрхива);

КонецФункции
```

Сначала пользователю показывается диалог выбора каталога для поиска файлов, помещаемых в архив. Когда каталог выбран, на его основе формируется маска поиска и передается в функцию `ДобавитьФайлыВАрхивНаСервере()`, в которой новый архив создается как неинициализированный объект `ЗаписьZIPФайла`. Затем файлы по маске помещаются в архив и вызывается метод этого объекта `ПолучитьДвоичныеДанные()`, который получает ZIP-архив в виде двоичных данных и закрывает архив аналогично методу `Записать()`. Эти двоичные данные возвращаются на клиент в процедуру `ДобавитьФайлыВАрхивЗавершение()`, где файл архива получается в виде локального файла с явно заданным именем.

Добавление выбранных файлов в архив

Рассмотрим пример добавления в архив произвольных файлов, выбранных пользователем. Данная задача может быть выполнена с помощью следующего программного кода (листинги 6.141, 6.142, 6.143, 6.144).

Листинг 6.141. Пример добавления файлов в ZIP-архив

```
&НаКлиенте
Процедура ЗаписатьФайлыВАрхив(Команда)

    ПоместитьФайлыВАрхив(Новый ОписаниеОповещения(
            "ПоместитьФайлыВАрхивЗавершение", ЭтотОбъект));

КонецПроцедуры
```

Листинг 6.142. Процедура «ПоместитьФайлыВАрхив()»

```
&НаКлиенте
Процедура ПоместитьФайлыВАрхив(ОповещениеОЗавершении)

    ПараметрыДиалога = Новый ПараметрыДиалогаПомещенияФайлов(
            "Выберите файлы для помещения в архив", Истина);
    НачатьПомещениеФайловНаСервер(ОповещениеОЗавершении, , , ПараметрыДиалога,
            УникальныйИдентификатор);

КонецПроцедуры
```

Листинг 6.143. Обработчик оповещения «ПоместитьФайлыВАрхивЗавершение()»

```
&НаКлиенте
Процедура ПоместитьФайлыВАрхивЗавершение(ПомещенныеФайлы, Дополнительно) Экспорт

	Если ПомещенныеФайлы = Неопределено Тогда
		Возврат;
	КонецЕсли;

	АдресаДобавляемыхФайлов = Новый Массив;
	ИменаДобавляемыхФайлов = Новый Массив;

	Для Каждого ДобавляемыйФайл Из ПомещенныеФайлы Цикл
		АдресаДобавляемыхФайлов.Добавить(ДобавляемыйФайл.Адрес);
		ИменаДобавляемыхФайлов.Добавить(ДобавляемыйФайл.ССылкаНаФайл.Имя);
	КонецЦикла;

	АдресФайлаВХранилище = ЗаписатьФайлыВАрхивНаСервере(
				АдресаДобавляемыхФайлов, ИменаДобавляемыхФайлов);

	НачатьПолучениеФайлаССервера(, АдресФайлаВХранилище, "e:\test\arc.zip");

КонецПроцедуры
```

Листинг 6.144. Функция «ЗаписатьФайлыВАрхивНаСервере()»

```
&НаСервереБезКонтекста
Функция ЗаписатьФайлыВАрхивНаСервере(МассивАдресовФайлов, МассивИменФайлов)

	Архив = Новый ЗаписьZIPФайла();

	Индекс = 0;
	Для Каждого АдресДобавляемогоФайла Из МассивАдресовФайлов Цикл
		ДанныеФайла = ПолучитьИзВременногоХранилища(АдресДобавляемогоФайла);
		ИмяФайла = КаталогВременныхФайлов() + МассивИменФайлов[Индекс];
		ДанныеФайла.Записать(ИмяФайла);

		Архив.Добавить(ИмяФайла,
				РежимСохраненияПутейZIP.НеСохранятьПути,
				РежимОбработкиПодкаталоговZIP.НеОбрабатывать);

		Индекс = Индекс + 1;

	КонецЦикла;

	ДанныеАрхива = Архив.ПолучитьДвоичныеДанные("", "",
		МетодСжатияZIP.Сжатие, УровеньСжатияZIP.Максимальный, МетодШифрованияZIP.Zip20);

	Возврат ПоместитьВоВременноеХранилище(ДанныеАрхива);

КонецФункции
```

Сначала пользователю показывается диалог множественного выбора файлов, добавляемых в архив, и начинается помещение выбранных файлов на сервер. Когда все файлы помещены, адреса во временном хранилище и имена помещенных файлов в виде двух отдельных массивов передаются в функцию `ЗаписатьФайлыВАрхивНаСервере()`.

В этой функции элементы массива помещенных файлов обходятся в цикле, данные каждого файла записываются во временный файл и этот файл добавляется в неинициализированный архив. Затем методом `ПолучитьДвоичныеДанные()` этот архив получается в виде двоичных данных, которые функция возвращает на клиент в процедуру `ПоместитьФайлыВАрхивЗавершение()`, где файл архива получается в виде локального zip-файла.

Распаковка файлов из архива

Рассмотрим пример распаковки файлов из архива с паролем, указанным пользователем. Данная задача может быть выполнена с помощью следующего программного кода (листинги 6.145, 6.146, 6.147, 6.148, 6.149).

Листинг 6.145. Пример распаковки файлов из ZIP-архива

```
&НаКлиенте
Процедура РаспаковкаФайловИзАрхива(Команда)

    // Показать ввод пароля на архив
    Подсказка = "Введите пароль на архив";
    Оповещение = Новый ОписаниеОповещения("ПослеВводаСтроки", ЭтотОбъект);
    ПоказатьВводСтроки(Оповещение, "", Подсказка, 0, Ложь);

КонецПроцедуры
```

Листинг 6.146. Процедура «ПослеВводаСтроки()»

```
&НаКлиенте
Процедура ПослеВводаСтроки(Пароль, Параметры) Экспорт

    Если НЕ Пароль = Неопределено Тогда
        ПоместитьФайлАрхива(Новый ОписаниеОповещения(
                "РаспаковкаФайловИзАрхиваЗавершение", ЭтотОбъект, Пароль));
    КонецЕсли;

КонецПроцедуры
```

Листинг 6.147. Процедура «ПоместитьФайлАрхива()»

```
&НаКлиенте
Процедура ПоместитьФайлАрхива(ОповещениеОЗавершении)

    НачатьПомещениеФайлаНаСервер(ОповещениеОЗавершении, , , "", , УникальныйИдентификатор);

КонецПроцедуры
```

Листинг 6.148. Обработчик оповещения «РаспаковкаФайловИзАрхиваЗавершение()»

```
&НаКлиенте
Процедура РаспаковкаФайловИзАрхиваЗавершение(ОписаниеПомещенногоФайла, Пароль) Экспорт

	Если ОписаниеПомещенногоФайла = Неопределено Тогда
		Возврат;
	КонецЕсли;

	ОписаниеИзвлеченныхФайлов = РаспаковкаФайловИзАрхиваНаСервере(
				ОписаниеПомещенногоФайла.Адрес, Пароль);

	Сообщение = Новый СообщениеПользователю();
	ИзвлеченныеФайлы = Новый Массив;

	Для Каждого Файл Из ОписаниеИзвлеченныхФайлов Цикл
		ИзвлеченныеФайлы.Добавить(Новый ОписаниеПередаваемогоФайла(
				Файл.ИмяФайла, Файл.АдресФайла));
		Сообщение.Текст = "Распаковывается файл: " + Файл.ИмяФайла;
		Сообщение.Сообщить();
	КонецЦикла;

	ПараметрыДиалога = Новый ПараметрыДиалогаПолученияФайлов(
				"Выберите каталог для сохранения файлов", Истина);

	НачатьПолучениеФайловССервера(ИзвлеченныеФайлы, ПараметрыДиалога);

КонецПроцедуры
```

Листинг 6.149. Функция «РаспаковкаФайловИзАрхиваНаСервере()»

```
&НаСервереБезКонтекста
Функция РаспаковкаФайловИзАрхиваНаСервере(АдресФайлаАрхива, Пароль)

	ДанныеФайла = ПолучитьИзВременногоХранилища(АдресФайлаАрхива);

	МассивИзвлеченныхФайлов = Новый Массив();
	ВременныйКаталог = КаталогВременныхФайлов();

	Архив = Новый ЧтениеZipФайла(ДанныеФайла.ОткрытьПотокДляЧтения(), Пароль);

	Для каждого Элемент из Архив.Элементы Цикл

		Архив.Извлечь(Элемент, ВременныйКаталог,РежимВосстановленияПутейФайловZIP.
				НеВосстанавливать,
			?(Элемент.Зашифрован, Пароль, ""));

		ПолныйПутьКИзвлеченномуФайлу = ВременныйКаталог + Элемент.Имя;
		АдресИзвлеченногоФайла = ПоместитьВоВременноеХранилище(Новый ДвоичныеДанные(
				ПолныйПутьКИзвлеченномуФайлу));

		ДанныеИзвлеченногоФайла = Новый Структура("АдресФайла, ИмяФайла",);
		ДанныеИзвлеченногоФайла.АдресФайла = АдресИзвлеченногоФайла;
		ДанныеИзвлеченногоФайла.ИмяФайла = Элемент.Имя;
```

```
            МассивИзвлеченныхФайлов.Добавить(ДанныеИзвлеченногоФайла);
            УдалитьФайлы(ПолныйПутьКИзвлеченномуФайлу);
        КонецЦикла;
        Архив.Закрыть();
        Возврат МассивИзвлеченныхФайлов;
КонецФункции
```

Сначала пользователю показывается диалог с запросом пароля на чтение архива. Затем пользователю предлагается выбрать файл архива, и начинается помещение выбранного файла на сервер. Когда файл помещен, вызывается функция `РаспаковкаФайловИзАрхиваНаСервере()`, в которую передаются адрес помещенного файла и введенный пользователем пароль.

В этой функции данные файла-архива получаются из временного хранилища и с помощью метода `ОткрытьПотокДляЧтения()` поток двоичных данных открывается для чтения. Затем на основе этого потока создается объект `ЧтениеZipФайла`.

После этого элементы архива обходятся в цикле, каждый файл извлекается из архива во временный каталог методом `Извлечь()`. На каждом шаге цикла заполняется массив структур, содержащих адреса во временном хранилище и имена извлеченных из архива файлов. Этот массив `МассивИзвлеченныхФайлов` функция возвращает в процедуру `РаспаковкаФайловИзАрхиваЗавершение()`, в которой переданные разархивированные файлы получаются на клиенте согласно их описаниям.

Разделение и объединение файлов архива

Как уже говорилось, для работы с файлами архивов большого объема можно использовать функции глобального контекста `РазделитьФайл()` и `ОбъединитьФайлы()`. Но во многих случаях вместо них более эффективно использовать методы `РазделитьДвоичныеДанные()` и `СоединитьДвоичныеДанные()`, потому что в этом случае нет необходимости предварительно сохранять двоичные данные в файл. Кроме того, при обмене с сервером через временное хранилище файлы передаются как раз в виде двоичных данных.

Рассмотрим пример разделения на части файла архива. Данная задача может быть выполнена с помощью следующего программного кода (листинги 6.150, 6.151).

Листинг 6.150. Пример разделения ZIP-архива

```
&НаКлиенте
Процедура РазделитьФайлыАрхива(Команда)

	ПоместитьФайлАрхива(Новый ОписаниеОповещения(
				"РазделитьФайлыАрхиваЗавершение", ЭтотОбъект));

КонецПроцедуры
```

Листинг 6.151. Обработчик оповещения «РазделитьФайлыАрхиваЗавершение()»

```
&НаКлиенте
Процедура РазделитьФайлыАрхиваЗавершение(ОписаниеПомещенногоФайла, Дополнительно) Экспорт

	Если ОписаниеПомещенногоФайла = Неопределено Тогда
		Возврат;
	КонецЕсли;

	ДанныеФайлаАрхива = ПолучитьИзВременногоХранилища(ОписаниеПомещенногоФайла.Адрес);
	МассивДанныхЧастей = РазделитьДвоичныеДанные(ДанныеФайлаАрхива, 1024 * 1024);

	МассивЧастичныхФайлов = Новый Массив;

	Индекс = 0;
	Для Каждого ДанныеЧасти Из МассивДанныхЧастей Цикл

		АдресФайла = ПоместитьВоВременноеХранилище(ДанныеЧасти);
		ИмяФайла = ОписаниеПомещенногоФайла.ССылкаНаФайл.Имя + "_" + СокрЛП(Строка(Индекс));
		МассивЧастичныхФайлов.Добавить(
			Новый ОписаниеПередаваемогоФайла(ИмяФайла, АдресФайла));
		Индекс = Индекс + 1;

	КонецЦикла;

	ПараметрыДиалога = Новый ПараметрыДиалогаПолученияФайлов(
				"Выберите каталог для сохранения файлов", Истина);

	НачатьПолучениеФайловССервера(МассивЧастичныхФайлов, ПараметрыДиалога);

КонецПроцедуры
```

Сначала пользователю показывается диалог для выбора файла архива, и начинается помещение выбранного файла архива на сервер. Когда файл помещен, данные архива получаются из временного хранилища и с помощью метода `РазделитьДвоичныеДанные()` двоичные данные файла делятся на части заданного в байтах размера.

После этого в цикле обхода получившегося массива двоичных данных на основе адреса во временном хранилище и имени каждой части создается массив описаний передаваемых файлов, согласно которому файлы частей сохраняются в выбранный пользователем каталог локальной файловой системы. Имя файла части формируется на основе имени файла архива с добавлением постфикса с порядковым номером части.

Процедура ПоместитьФайлАрхива() была приведена в листинге 6.147.

Рассмотрим теперь обратный пример объединения файлов архива. Данная задача может быть выполнена с помощью следующего программного кода (листинги 6.152, 6.153).

Листинг 6.152. Пример объединения файлов из ZIP-архива

```
&НаКлиенте
Процедура ОбъединитьФайлыАрхива(Команда)

    ПоместитьФайлыВАрхив(Новый ОписаниеОповещения(
                        "ОбъединитьФайлыАрхиваЗавершение", ЭтотОбъект));

КонецПроцедуры
```

Листинг 6.153. Обработчик оповещения «ОбъединитьФайлыАрхиваЗавершение()»

```
&НаКлиенте
Процедура ОбъединитьФайлыАрхиваЗавершение(ПомещенныеФайлы, Дополнительно) Экспорт

    Если ПомещенныеФайлы = Неопределено Тогда
        Возврат;
    КонецЕсли;

    ДанныеОбъединяемыхФайлов = Новый Массив;

    Для Каждого ЧастичныйФайл Из ПомещенныеФайлы Цикл
        ДанныеОбъединяемыхФайлов.Добавить(
                    ПолучитьИзВременногоХранилища(ЧастичныйФайл.Адрес));
    КонецЦикла;

    ОбъединенныеДанные = СоединитьДвоичныеДанные(ДанныеОбъединяемыхФайлов);
    АдресФайлаВХранилище = ПоместитьВоВременноеХранилище(ОбъединенныеДанные);
    НачатьПолучениеФайлаССервера(, АдресФайлаВХранилище, "c:\temp\pict.zip");/

КонецПроцедуры
```

Сначала пользователю показывается диалог множественного выбора частичных файлов архива, которые требуется объединить, и начинается помещение выбранных файлов на сервер. Когда все файлы помещены, данные частичных файлов получаются из временного хранилища и добавляются в массив ДанныеОбъединяемыхФайлов.

На основе этого массива с помощью метода СоединитьДвоичныеДанные() двоичные данные объединяются, помещаются во временное хранилище и затем сохраняются в локальную файловую систему в виде целого zip-архива.

Процедура ПоместитьФайлыВАрхив() была приведена в листинге 6.142.

DBF-файлы

Для работы с базами данных формата DBF в системе имеется специальный объект – `XBase`. Механизм работы с базами данных формата DBF предназначен для обеспечения возможности манипулирования ими непосредственно из встроенного языка системы «1С:Предприятие». Возможно практически любое манипулирование данными.

Помимо работы с существующими базами данных, объект `XBase` имеет набор методов, позволяющих создать новую базу данных произвольной структуры, новые индексы и новый индексный файл. Следует отметить, что если использование методов, изменяющих структуру базы данных, возможно только для объектов, не связанных с существующей базой данных (т. е. для вновь создаваемых баз данных), то создание новых индексов и индексного файла возможно как для создаваемых баз данных, так и для уже существующих и открытых.

Для записи данных в файл DBF может использоваться следующий фрагмент кода (листинг 6.154).

Листинг 6.154. Пример записи данных в файл DBF

```
&НаКлиенте
Процедура ЗаписьДанных(Команда)

	Путь = "c:\temp\";
	БД = Новый XBase;
	БД.Поля.Добавить("CODE", "S", 9);
	БД.Поля.Добавить("NAME", "S", 40);
	БД.Поля.Добавить("DATA_B", "S", 10);
	БД.Поля.Добавить("CHILD", "S", 1);

	БД.СоздатьФайл(Путь + "list.dbf", Путь + "index.cdx");
	БД.Индексы.Добавить("IDXCODE", "CODE");
	ИБД = БД.СоздатьИндексныйФайл(Путь + "index.cdx");
	БД.АвтоСохранение = Истина;

	// Получить данные сотрудников в виде одной большой строки.
	СтрокаРазделителя = "*";
	СтрокаСотрудников = ПолучитьСтрокиСотрудников(СтрокаРазделителя);
	// Получить массив строк для каждого сотрудника.
	СтрокиСотрудников = СтрРазделить(СтрокаСотрудников, Символы.ПС);

	Для ТекущаяСтрока = 0 По СтрокиСотрудников.Количество() - 1 Цикл
		// Получить данные каждого сотрудника.
		Данные = СтрРазделить(СтрокиСотрудников[ТекущаяСтрока], СтрокаРазделителя);
		БД.Добавить();

		БД.CODE = Данные[0];
		БД.NAME = Данные[1];
		БД.DATA_B = Данные[2];
```

```
            БД.CHILD = Данные[3];
        КонецЦикла;

        БД.ЗакрытьФайл();

КонецПроцедуры
```

Создается файл, у которого определяются четыре поля (CODE, NAME, DATA_B, CHILD). Все поля имеют строковый тип.

У создаваемого DBF-файла определяется один индексный файл. Следует отметить, что у DBF-файла может быть определено несколько индексов, но создаваемый при этом индексный файл всегда один.

При записи данных следует обращать внимание на свойство АвтоСохранение объекта `XBase`. Если оно установлено в значение `Истина`, то добавление записи в файл производится при вызове следующего метода `Добавить()`, создающего новую запись.

Из процедуры `ЗаписьДанных()` вызывается серверная функция `ПолучитьСтрокиСотрудников()`, в которой в цикле обхода выборки элементов справочника Сотрудники формируются строки с данными сотрудников и в виде одной большой строки возвращаются на клиент (см. листинг 6.155).

Затем из этой строки получается массив строк с данными для каждого сотрудника и каждая строка разбирается на структуры с помощью функции глобального контекста `СтрРазделить()`. Полученные структуры с данными каждого сотрудника записываются в соответствующие поля новой записи DBF-файла.

Листинг 6.155. Функция «ПолучитьСтрокиСотрудников()»

```
&НаСервере
Функция ПолучитьСтрокиСотрудников(Разделитель)

        СтрокаСотрудника = Новый Массив();
        МассивСтрокСотрудников = Новый Массив();

        Выборка = Справочники.Сотрудники.Выбрать();
        Пока Выборка.Следующий() Цикл
                СтрокаСотрудника.Очистить();
                СтрокаСотрудника.Добавить(Выборка.Код);
                СтрокаСотрудника.Добавить(Выборка.Наименование);
                СтрокаСотрудника.Добавить(Формат(Выборка.ДатаРождения, "ДЛФ=D"));
                СтрокаСотрудника.Добавить(Выборка.КоличествоДетей);

                Данные = СтрСоединить(СтрокаСотрудника, Разделитель);
                МассивСтрокСотрудников.Добавить(Данные);
        КонецЦикла;

        Возврат СтрСоединить(МассивСтрокСотрудников, Символы.ПС);

КонецФункции
```

Для чтения данных может использоваться следующий фрагмент кода (листинг 6.156).

Листинг 6.156. Пример чтения данных из файла DBF

```
&НаКлиенте
Процедура ЧтениеДанных(Команда)

	Путь = "c:\temp\";
	БД = Новый XBase;
	БД.ОткрытьФайл(Путь + "list.dbf", Путь + "index.cdx");

	Сообщение = Новый СообщениеПользователю();

	Пока Истина Цикл
	Сообщение.Текст = БД.CODE + " " + БД.Name;
		Сообщение.Сообщить();
		Если Не БД.Следующая() Тогда
		Прервать;
		КонецЕсли;
	КонецЦикла;

	БД.ЗакрытьФайл();

КонецПроцедуры
```

В приведенном фрагменте не используется определенный при записи индекс, порядок отображения данных может быть следующий (фактически это соответствует физическому положению записей в файле) – листинг 6.157.

Листинг 6.157. Порядок отображения данных

```
000000001   Алексеев Сергей Иванович
REST-0003   Артемов Игорь Владимирович
000000002   Смирнова Светлана Ивановна
```

Изменим код, добавив в него использование индекса (листинг 6.158).

Листинг 6.158. Пример чтения данных из файла DBF

```
&НаКлиенте
Процедура ЧтениеДанных(Команда)

	Путь = "c:\temp\";
	БД = Новый XBase;
	БД.ОткрытьФайл(Путь + "list.dbf", Путь + "index.cdx");
	БД.ТекущийИндекс = БД.Индексы.IDXCODE;

	Сообщение = Новый СообщениеПользователю();

	Пока Истина Цикл
	Сообщение.Текст = БД.CODE + " " + БД.Name;
```

```
            Сообщение.Сообщить();
            Если Не БД.Следующая() Тогда
            Прервать;
            КонецЕсли;
    КонецЦикла;

    БД.ЗакрытьФайл();

КонецПроцедуры
```

В этом случае в результате будут получены следующие записи, отсортированные в порядке возрастания кодов сотрудников (листинг 6.159).

Листинг 6.159. Порядок отображения данных

```
000000001   Алексеев Сергей Иванович
000000002   Смирнова Светлана Ивановна
REST-0003   Артемов Игорь Владимирович
```

При этом надо иметь в виду, что при открытии файла объект `XBase` позиционируется на записи, которая является физически первой в файле. Однако с точки зрения используемого индекса она может быть далеко не первой. При выполнении метода `Следующая()` выбирается следующая запись в индексном файле. Поэтому, для того чтобы обойти все записи, необходимо явно использовать метод `Первая()`, как показано ниже (листинг 6.160).

Листинг 6.160. Пример использования метода «Первая()»

```
&НаКлиенте
Процедура ЧтениеДанных(Команда)

    Путь = "c:\temp\";
    БД = Новый XBase;
    БД.ОткрытьФайл(Путь + "list.dbf", Путь + "index.cdx");
    БД.ТекущийИндекс = БД.Индексы.IDXCODE;

    Сообщение = Новый СообщениеПользователю();

    БД.Первая();
    Пока Истина Цикл
    Сообщение.Текст = БД.CODE + " " + БД.Name;
            Сообщение.Сообщить();
            Если Не БД.Следующая() Тогда
            Прервать;
            КонецЕсли;
    КонецЦикла;

    БД.ЗакрытьФайл();

КонецПроцедуры
```

Индексы полезно использовать не только для упорядочивания выборок записей по какому-либо полю, но и для реализации каких-либо «поисковых механизмов». Рассмотрим следующий пример (листинг 6.161).

Листинг 6.161. Пример поиска данных в файле DBF

```
&НаКлиенте
Процедура ПоискДанных(Команда)

	Путь = "c:\temp\";
	БД = Новый XBase;
	БД.ОткрытьФайл(Путь + "list.dbf", Путь + "index.cdx");
	БД.ТекущийИндекс = БД.Индексы.IDXCODE;

	Сообщение = Новый СообщениеПользователю();
	Если БД.Найти("000000002", "=") Тогда

		Пока Истина Цикл
		Сообщение.Текст = БД.CODE + " " + БД.Name;
			Сообщение.Сообщить();
			Если Не БД.Следующая() Тогда
		Прервать;
		КонецЕсли;
		КонецЦикла;

	КонецЕсли;

	БД.ЗакрытьФайл();

КонецПроцедуры
```

В результате будут получены только две записи (листинг 6.162).

Листинг 6.162. Порядок отображения данных

```
000000002   Смирнова Светлана Ивановна
REST-0003   Артемов Игорь Владимирович
```

Менять текущий индекс можно и во время выборки (однако следует помнить, что при «умелой» манипуляции индексами можно никогда не достигнуть конца DBF-файла).

Фирма «1С»
123056, Москва, а/я 64, Селезневская ул., 21
Тел.: (495) 737-92-57
1c@1c.ru, http://www.1c.ru/

Издательство ООО «1С-Паблишинг»
127434, Москва, Дмитровское ш., 9
Тел.: (495) 681-02-21
publishing@1c.ru, http://books.1c.ru

Об опечатках просьба сообщать по адресу books.v8@1c.ru.