

ЮРИЙ ШУТЕНКО



Visual FoxPro

ДЛЯ ПРОФЕССИОНАЛОВ



Visual FoxPro +
Web-ТЕХНОЛОГИИ:
ASP.NET, LINQ, Silverlight,
JavaScript, AJAX, JSON

Visual FoxPro +
ТЕХНОЛОГИИ WINDOWS:
COM, DCOM и COM+,
WINDOWS Shell, ActiveX

ОБМЕН ДАННЫМИ
С ДРУГИМИ
ПРИЛОЖЕНИЯМИ

РАСШИРЕНИЕ
ВОЗМОЖНОСТЕЙ:
НАСТРОЙКА ИНТЕРФЕЙСА,
ТРЮКИ И СЕКРЕТЫ

PRO
ПРОФЕССИОНАЛЬНОЕ
ПРОГРАММИРОВАНИЕ

+ CD

УДК 681.3.068
ББК 32.973.26-018.1
Ш97

Шутенко Ю. Т.

Ш97 Visual FoxPro для профессионалов. — СПб.: БХВ-Петербург, 2009. — 576 с.: ил. + CD-ROM — (Профессиональное программирование)

ISBN 978-5-9775-0307-5

Книга посвящена расширению возможностей приложений Visual FoxPro за счет использования современных технологий. Показано применение различных Web-технологий, таких как ASP.NET, LINQ, Silverlight, JavaScript, AJAX, JSON и др. Описаны способы размещения и получения данных в Интернете. Рассмотрено применение Windows-технологий: COM, DCOM и COM+, Windows Shell, ActiveX и др. Показана организация обмена данными с различными СУБД (MySQL, SQL Server) и другими приложениями. Уделено внимание вопросам расширения возможностей VFP за счет настроек интерфейса и применения различных трюков при программировании. Компакт-диск содержит исходные тексты программ, классов и демонстрационных примеров, описанных в книге.

Для разработчиков

УДК 681.3.068
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Екатерина Капалыгина</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Корректор	<i>Зинаида Дмитриева</i>
Дизайн серии	<i>Инны Тачиной</i>
Оформление обложки	<i>Елены Беляевой</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 29.10.08.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 46,44.

Тираж 2000 экз. Заказ №

"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0307-5

© Шутенко Ю. Т., 2008

© Оформление, издательство "БХВ-Петербург", 2008

Оглавление

Благодарности	11
Введение	13
Глава 1. HTML.....	17
Что представляет собой документ HTML?	17
Кто в DOM-ике живет?	19
О пользе первичного ключа	30
Работа с объектами. Сборка произвольного HTML-файла	31
Принципы создания элементов документа	32
Сказ про овечку Долли в доме HTML	36
Практическое использование	42
Отчеты в формате HTML/XHTML	42
Некоторые нюансы использования HTML DOM с VFP	44
Глава 2. XML и другие X	47
Терминология	47
Well-formed против valid	48
Структура XML-документа	49
Пролог	50
Корневой элемент	52
Элементы	53
Типы структуры XML-документов (или смерть буриданова осла)	55
Экспорт из Visual FoxPro в XML	58
Общие правила создания объектов объектной модели XML-документа	58
Создание инструкции по обработке (processing instruction)	59
Особенности использования декларации <i>standalone</i>	61
XSLT-преобразование	64

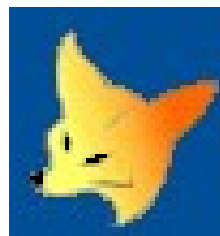
Глава 3. JavaScript, AJAX, JSON	77
JavaScript — давайте познакомимся!	77
Элемент <i>script</i>	78
Где место JavaScript в документах?.....	80
Эти "переменчивые" переменные.....	83
Двуликий оператор <i>var</i>	85
Массивы.....	87
Функции.....	90
Объекты	94
JSON и AJAX.....	99
 Глава 4. ASP.NET + Visual FoxPro	 105
Терминология.....	105
Работа с базами данных Visual FoxPro.....	106
Web-сайты на ASP.NET.....	107
Создание простейшего сайта.....	108
Работа с локальными данными	122
Работа с данными, размещенными на публичном сервере.....	139
 Глава 5. Web-службы	 147
Использование Web-служб.....	149
Получение информации через сервисные приложения VFP.....	149
Получение информации с помощью XML DOM	160
Получение информации с помощью <i>WSDLReader</i>	165
Создание собственной Web-службы	169
 Глава 6. Введение в LINQ.....	 181
Первые шаги.....	182
Что требуется для работы с LINQ?	182
Первый проект с LINQ	183
Подготовка.....	183
Работа с ORD.....	187
Создание страницы с данными из таблиц	190
 Глава 7. Введение в SilverLight	 203
Что такое SilverLight?	203
Что требуется для введения SilverLight в приложения Visual FoxPro?	204
Первые шаги.....	204
Где разместить вызов метода создания плагина?.....	210
Где разместить вызов функции создания плагина?.....	211
Файл содержания SilverLight	212
Объект <i>Canvas</i>	212
Размещение объекта <i>Canvas</i>	213
Первые результаты.....	214

Глава 8. COM, DCOM и COM+	221
Краткий экскурс в историю.....	221
Что такое COM?	222
Что такое DCOM?	223
Что такое COM+?	224
"Состояние без состояния"	224
Потоки и апартаменты.....	225
COM+ и транзакции	226
Как выполнять транзакции под COM+ для данных VFP	228
Создание классов <i>CRMWorker</i> и <i>CRMCompensator</i>	229
Глава 9. <i>FileSystemObject</i>.....	237
Объект <i>FileSystemObject</i>	238
Свойство <i>Drives</i>	238
Методы <i>FileSystemObject</i>	239
Объект <i>File</i>	240
Свойства объекта <i>File</i>	240
Методы объекта <i>File</i>	242
Объект <i>Folder</i>	243
Свойства объекта <i>Folder</i>	243
Методы объекта <i>Folder</i>	247
Объект <i>Drive</i>	248
Свойства объекта <i>Drive</i>	248
Объект <i>TextStream</i>	248
Свойства объекта <i>TextStream</i>	249
Методы объекта <i>TextStream</i>	250
Глава 10. Объекты <i>WshShell</i> и <i>WshNetwork</i>.....	253
Что такое WSH?	253
Объект <i>WshShell</i>	255
Свойства.....	255
Методы.....	262
Объект <i>WshNetwork</i>	268
Свойства.....	268
Методы.....	268
Глава 11. Использование Windows Management Instrumentation	273
Структура WMI	275
Создание объектов	276
Подключение с помощью WMI-моникера.....	277
Извлечение информации из объектов WMI.....	280
Провайдеры	280
Работа с классами.....	284
Особенности работы с WMI на Windows Vista	297

Глава 12. Использование ActiveX	301
Использование WEB Browser Control	301
Планировщик маршрута на базе Microsoft Virtual Earth.....	302
Как создать объект <i>VEMap</i>	303
Реализация	305
Использование Windows Image Acquisition	314
Глава 13. Сервер MySQL и Visual FoxPro	329
Краткие характеристики версии 5.0	330
Установка и конфигурирование MySQL сервера.....	332
Работа с мастером конфигурации MySQL сервера	334
Инструменты для работы с базами данных MySQL.....	346
Создание и работа с базой данных MySQL.....	347
Разметка типов данных MySQL в типы данных Visual FoxPro	353
Глава 14. Advantage Database Server 9.0	357
Что пишут разработчики сервера о своем продукте?	357
Чем он может быть интересен для программистов Visual FoxPro?.....	359
Установка Advantage Database Server.....	361
Начало работы	366
Утилита конфигурации	366
Утилита Advantage Data Architect (ARC)	367
Конфигурация Advantage серверов и клиентов	370
"Кто есть кто" в Advantage?	370
Как сконфигурировать?	371
Конфигурация Advantage Local Server	371
Конфигурация Advantage Database Server	373
Конфигурация клиента	374
Проба пера	381
Работа со свободными таблицами Visual FoxPro	381
Импорт базы данных Northwind в словарь данных Advantage	384
Использование сквозных запросов	389
Полнотекстовый поиск (FTS).....	391
Использование утилиты DBCCConvert.prg	393
Заключение	397
Глава 15. Использование CursorAdapter	399
Класс <i>CursorAdapter</i>	400
Создание настраиваемого класса <i>CursorAdapter</i>	401
Настройка класса для работы с MySQL через ODBC	405
Как насчет модификации команды <i>SelectCmd</i> ?	414
Глава 16. XMLAdapter	423
Как извлечь данные из DataSet Web-службы?.....	426
Особенности работы с вложенными таблицами	433

Извлечение XML-данных из MS SQL Server.....	434
MS SQL Server 2000	434
Как извлечь данные?.....	435
MS SQL Server 2005	441
Особенности формирования XML MS SQL Server	449
Глава 17. Расширение Visual FoxPro с помощью Visual FoxPro	453
IntelliSense.....	456
Объекты-компаньоны	464
Глава 18. Инструментарий Visual FoxPro	469
Task List (Список задач)	469
Environment Manager (Диспетчер среды).....	476
Data Explorer (Проводник к данным)	482
Работа с природными базами Visual FoxPro.....	482
Глава 19. Полезные решения	493
Предотвращение повторного запуска приложения.....	493
Поиск окна приложения	493
Использование таблицы атомов.....	505
Использование <i>Mutex</i>	511
ПРИЛОЖЕНИЯ	517
Приложение 1. <i>FileSystemObject</i> и WMI	519
Объект <i>Scripting.FileSystemObject</i>	519
Настройки безопасности WMI.....	542
Язык запросов WMI Query Language.....	546
Приложение 2. Свойства и методы объекта <i>WshShell</i>	549
Набор переменных среды — <i>PROCESS</i>	549
Методы объекта <i>WshShell</i>	550
Специальные каталоги.....	555
База данных foxdevcons (MySQL).....	558
Приложение 3. Описание компакт-диска.....	561
Предметный указатель	565

ГЛАВА 2



XML и другие X

Пожалуй, не найдется другого языка с такой же витиеватой историей, какая происходит по сей день с XML. Чего только не писалось о нем журналистами и околопрограммными авторами за десятилетие его развития. От невероятной по "глубине понимания предмета" фразы "SGML умер, да здравствует XML!", до сверхоптимистических надежд, вроде "XML — безальтернативное хранилище данных". Не вдаваясь в суть, да часто даже и не пытаясь понять, что же это такое, нас убеждали то в очередной революции, сметающей все на своем пути, то в очередном тупике, то в... — перечислять можно до бесконечности. Не отставал от журналистов и W3-консорциум. За неполных 10 лет консорциумом было издано четыре редакции одной и той же спецификации. А что же программисты? А программисты, следя за путями его развития, похожими скорее на запутанные схемы движения метро, гадали, куда же кривая выведет? Кривая вывела. Мы имеем стандарт, а также многочисленные его вольные трактовки. А что вы хотели от кривой?

Терминология

Погружение в изучение XML мне, лично, очень напоминало путешествие Алисы в страну чудес. "Сплошная путаница", — сказала же она, почитав труды многих авторов. Так что же представляет собой XML?

Если исходить из его названия, то это сочетание одиночных символов из трех слов его определения — eXtensible Markup Language — расширяемый язык разметки.

Если рассматривать его физическую сущность, то XML, с одной стороны, может представлять собой обычный ASCII-файл, информация в котором оформлена определенным образом, с другой стороны — содержимое какого-

то поля базы данных. В первом случае мы говорим о XML-файле, во втором — о XML-документе.

В чем отличие XML от HTML? В первую очередь различие между этими двумя языками заключается в целях, ради которых эти языки были созданы. Если HTML изначально создавался для представления информации, то XML — для публикации больших массивов информации. Если HTML имеет фиксированный набор тэгов, составляющий словарь языка, то XML такого словаря не имеет, что позволяет разработчикам создавать свои собственные словари и использовать эти словари, когда требуется описать какие-то данные. Однако главное различие, и это различие является для нас наиболее важным, заключается в том, что HTML является языком *представления* данных, тогда как XML является языком *описания* данных.

Поскольку, как уже было сказано ранее, XML может быть представлен в виде физического, текстового ASCII-файла, то для нас крайне важен тот факт, что в таком случае можно говорить о кроссплатформенности XML. По своей природе XML является полностью интерпретируемым языком, а значит, может представлять и представляет собой средство обмена информацией между приложениями, вне зависимости от того, на какой платформе эти приложения были созданы.

Совет

Несмотря на сложность восприятия спецификации W3-консорциума из-за особенности изложения, именно она должна служить основой для понимания и изучения XML.

Well-formed против valid

Вероятно вы слышали такие определения XML-документов, как "well-formed" (хорошо сформированный) и "valid" (правильный). Многие уважаемые мной авторы иногда прямо делят XML-документы на указанные типы. Квинтэссенцией такого деления могут служить определения, приведенные в Википедии.

Хорошо сформированный (well-formed) XML-документ

"Хорошо сформированный XML документ" на http://en.wikipedia.org/wiki/Well-formed_XML_document определяют как XML-документ, который имеет корректный XML-синтаксис. В соответствии с рекомендацией W3-консорциума, это означает:

- ☐ XML-документы должны иметь корневой элемент;
- ☐ XML-элементы должны иметь закрывающий тэг;

- ❑ XML-тэги являются регистро-чувствительными;
- ❑ XML-элементы должны быть вложены должным образом;
- ❑ значения атрибутов XML должны быть всегда заключены в кавычки.

Далее следует утверждение, что не нужно путать этот тип с "правильным" XML-документом, который определен как "хорошо сформированный" XML-документ и который, кроме того, соответствует правилам определения типа документа (Document Type Definition (DTD)) или XML-схеме (XSD), которая поддерживается W3-консорциумом в качестве альтернативы определению типа документа.

Правильный (valid) XML-документ

Правильный XML-документ определен W3-консорциумом (http://en.wikipedia.org/wiki/Valid_XML_document), как "хорошо сформированный" документ, который, кроме того, соответствует правилам определения типа документа (Document Type Definition (DTD)) или XML-схеме (XSD), которая поддерживается W3-консорциумом в качестве альтернативы DTD.

Далее следует утверждение, что не нужно путать этот тип с "хорошо сформированным" XML-документом, который определен как имеющий корректный XML-синтаксис в соответствии со стандартами W3-консорциума.

Мне такие определения напоминают анекдот про две половинки таблетки от головы и живота — "и смотри, не перепутай!".

Что ясно и недвусмысленно сказано в спецификации W3-консорциума?

"Объект данных представляет собой XML-документ, если он хорошо сформирован так, как определено в этой спецификации. Дополнительно XML-документ является правильным, если он отвечает определенным дополнительным ограничениям".

Следовательно, "правильность" является не более чем свойством XML-документа, но уж никак не может быть отдельным типом. А плохо сформированный XML-документ вообще не является XML.

ПРИМЕЧАНИЕ

На самом деле, чтобы быть "хорошо сформированным" документ должен отвечать большему числу правил и ограничений. Мы разберем правила создания "хорошо сформированного документа" чуть позже, а пока будем считать приведенные ранее базовыми правилами.

Структура XML-документа

Несмотря на то, что структура документа, как логическая, так и физическая и их составляющие однозначно определены в спецификации W3-консорциума,

здесь тоже нашлось немало мест для разночтений и неадекватных толкований, чему, возможно, способствует и сама спецификация XML.

Что говорит нам спецификация консорциума?

"Физически, документ составляется из единиц, называемых объектами. Объект может ссылаться на другие объекты, что вызывает их включение в документ. Документ начинается с "корня" или с объекта документа, определение которого приведено ниже:

[Определение: объект документа служит в качестве корня дерева объектов и является стартовой точкой для XML-обработчика]".

В отличие от других объектов, объект документа не имеет имени и может показаться во входном потоке обработчика вовсе без какой-либо идентификации.

Логически документ состоит из деклараций, элементов, комментариев, символьных ссылок и инструкций по обработке, которые все без исключения должны быть обозначены в документе явной разметкой. Логическая и физическая структуры ДОЛЖНЫ БЫТЬ вложены должным образом, т. к. это описано в данной спецификации.

Следуя спецификации консорциума, общую структуру документа можно представить в виде трех обязательных элементов: пролога, корневого элемента и содержания, являющегося составной частью корневого элемента.

Пролог

Пролог размещается в документе до корневого элемента — вот здесь действительно возможна путаница в представлении структуры. Корневой элемент (Root element) не нужно путать с корнем документа (Document root)! Пролог включает в себя, по меньшей мере, одну инструкцию по обработке (processing instruction). В пролог можно включить дополнительные инструкции по обработке, декларацию типа документа, а также комментарии.

И здесь не обошлось без путаницы. Пролог, а по сути "видимая" часть документа, начинается с инструкции по обработке, содержащей XML-декларацию:

```
<?xml version="1.0" [encoding="значение"] [standalone="yes/no"]>
```

Однако в Интернете, в том числе и на сайте www.xml.com, претендующем на звание "XML-коммуны", вы найдете утверждения, подобные приведенному далее:

"XML-декларация не является инструкцией по обработке и не является обязательной в XML-документе". Как говорится, приехали!

Что есть истина в этом случае? Читаем спецификацию:

"Инструкция по обработке начинается с целевого указателя, используемого для идентификации приложением, которому эта инструкция направляется. Целевые имена "XML", "xml" и им подобные резервированы для стандартизации этой и последующих будущих версий этой спецификации. ... XML-документы ДОЛЖНЫ начинаться с XML-декларации, которая указывает используемую версию XML".

В данном случае в спецификации все определено однозначно и предельно ясно. Неясно одно — на чем основаны приведенные ранее утверждения.

Кроме указания версии XML декларация может содержать две необязательных декларации: `encoding` (кодировка) и `standalone` (автономный).

С кодировкой и автономностью мы разберемся позже, когда будем работать с реальными примерами.

Кроме инструкций по обработке, пролог может содержать декларацию типа документа. Декларация типа документа, если она включена в документ, определяет его корневой элемент, а также правила формирования документа. И здесь "имеет место быть" путаница в материалах, представленных в Интернете. Очень часто декларацию типа документа (Document Type Declaration) сокращают, в силу ее наименования на английском языке, до DTD, что в корне неверно! Декларация типа документа может содержать внутренний поднабор деклараций разметки, представляющий собой коллекцию деклараций `ELEMENT`, `ATTLIST`, `ENTITY` и `NOTATION`, может указывать на внешний поднабор деклараций или включать в себя оба поднабора, по которым выстраивается грамматика класса документа. Эта грамматика и известна под именем "DTD" и представляет собой сокращение от Document Type Definition (определение типа документа). Декларация типа документа и определение типа документа очень тесно связаны, но все-таки это разные вещи. Чтобы различать их, приведу два простых примера.

Листинг 2.1. DTD как внутренний поднабор декларации типа документа

```
<?xml version="1.0"?>
<!DOCTYPE item [
  <!ELEMENT item (id,itemname,price,instore,description)>
  <!ELEMENT id (#PCDATA)>
  <!ELEMENT itemname (#PCDATA)>
  <!ELEMENT price (#PCDATA)>
  <!ELEMENT instore (#PCDATA)>
  <!ELEMENT description (#PCDATA)>
]>
```

```

<item>
<id>SI_2CA0MAVOE</id>
<itemname>iPod Nano</itemname>
<price>230.0000</price>
<instore>10</instore>
<description>Apple Multimedia player</description>
</item>

```

Листинг 2.2. DTD как внешний поднабор декларации типа документа

```

<?xml version="1.0"?>
<!DOCTYPE item SYSTEM "http://www.foxhelp.ru/simple.dtd"> [
<item>
<id>SI_2CA0MAVOE</id>
<itemname>iPod Nano</itemname>
<price>230.0000</price>
<instore>10</instore>
<description>Apple Multimedia player</description>
</item>

```

И в первом, и во втором примерах `<!DOCTYPE item` представляет собой декларацию типа документа класса `item`. В первом примере все, что находится внутри квадратных скобок, представляет собой DTD — определение типа документа. Во втором примере декларация типа документа ссылается на определение типа документа, размещенное во внешнем файле. В таком случае приложение-обработчик XML должно загрузить этот файл с указанного ресурса, чтобы определить состоятельность XML-документа. Как видно из примеров, декларация типа документа определяет корневой элемент XML `item`, но это не сделано в DTD. В свою очередь DTD определяет модель содержания — что и в каком порядке должно следовать в документе, а также список атрибутов элемента (в приведенных примерах такого списка нет). Декларация типа документа этого не делает.

Корневой элемент

Корневой элемент следует за прологом и может быть представлен в XML только один раз. Если вы использовали функцию `Visaul FoxPro` для экспорта данных из таблицы в XML, то в качестве корневого элемента `Visaul FoxPro` использует элемент с именем `<VFPData>`. Для понимания сути этого элемента считайте его контейнером на вершине иерархии вложенных в него объектов. В первом приближении его можно сравнить с формой `Visual FoxPro`.

Все, что вложено в корневой документ, представляет собой содержание документа, составленное из элементов.

Элементы

Разберем сущность термина "элемент", т. к. иногда в Интернете смешиваются два термина — элемент и тэг. Однако это совершенно разные сущности.

- ☐ Элемент является структурной единицей XML.
- ☐ Тэг является элементом разметки.
- ☐ Элемент не является тэгом.
- ☐ Тэг, в свою очередь, не является элементом, но представляет собой его часть.
- ☐ Тэги обрамляют элементы и разделяют их.
- ☐ Элемент может содержать другие элементы.
- ☐ Тэг не может содержать других тэгов.

В общем случае элемент начинается с открывающего тэга и заканчивается закрывающим тэгом:

```
<id>SI_2CA0MAVOE</id>
```

Исключением из этого правила является либо одиночный пустой элемент, либо элемент, описываемый атрибутами и не имеющий вложенных элементов. В этом случае тэг приводится к виду:

```
<id />
```

или

```
<id value="SI_2CA0MAVOE" />
```

Теперь самое время проверить XML-документ, приведенный в листинге 2.1, на предмет его хорошего формирования и правильности.

Первые два элемента этого документа представляют собой пролог документа, в который входит декларация документа, указывающая версию XML, и декларация типа документа, представляющая собой вложенное определение типа документа (DTD). Элементы документа имеют открывающий и закрывающий тэги, не перекрывают друг друга и вложены должным образом. Не имеется различий в регистре символов, составляющих имена открывающего и закрывающего тэгов одного и того же элемента. В определении типа документа указано, что элемент *item* может иметь пять вложенных элементов, следующих в указанном порядке. Для всех элементов определены типы их значений, как разбираемые строчные данные.

Является ли этот документ хорошо сформированным? Несомненно!

Является ли этот документ действительным? Разумеется!

А что произойдет, если вы вставите в действительный документ элемент, не описанный в определении типа документа? Это всенепременнейше вызовет ошибку синтаксического анализатора (parser), — "Нет у меня в списке этого самозванца, вы мне подсовываете недействительный документ!" Но! Если вы при создании действительного документа не будете соблюдать синтаксические правила, например, забудете сопроводить какой-либо элемент закрывающим тэгом, что будет подразумевать наличие у него дочерних элементов, то тот же синтаксический анализатор вполне правомочно опять-таки выдаст ошибку, т. к. не сможет найти детишек элемента с отсутствующим закрывающим тэгом. То есть документ не будет являться "хорошо сформированным". Проверим это на практике, написав простейший синтаксический анализатор XML-документа (листинг 2.3).

Листинг 2.3. Программа простейшего анализатора XML

```
Lparameters lpFileName
CLEAR

!*      LOCAL loXMLDocument As MSXML2.DOMDocument.6.0
!*      loXMLDocument=Createobject("MSXML2.DOMDocument.6.0")
LOCAL loXMLDocument As Microsoft.XMLDOM
loXMLDocument=Createobject("Microsoft.XMLDOM")
local lcResult As String
loXMLDocument.Async=.F.
loXMLDocument.validateOnParse=.T.
loXMLDocument.Load(lpFileName)
? "Parsing XML document..."
If !Empty(loXMLDocument.parseError.Line)
    ? "Error Code: " + ;
        Transform(
            loXMLDocument.parseError.errorCode, "9999999999999999";
        )
    ? "Error Reason: " + ;
        Transform(loXMLDocument.parseError.reason)
    ? "Error Line: " + ;
        Transform(loXMLDocument.parseError.Line)
Else
    ? "Congratulations! You have a valid or well-formed XML document!"
Endif
Release loXMLDocument
```

ПРИМЕЧАНИЕ

Использование в примере более старого класса `Microsoft.XMLDOM` определено его возможностями по более внятной детализации ошибок, которая, увы, исчезла в `MSXML2.DOMDocument.6.0`. Так, например, если поменять местами два элемента, то синтаксический анализатор на основе `MSXML2.DOMDocument.6.0` сообщит, что структура запрещена DTD, тогда как синтаксический анализатор на основе `Microsoft.XMLDOM`, сообщит, что структура неверна и вместо одного элемента предполагался другой с указанием ожидаемого имени.

Для начала произведите синтаксический разбор приведенного ранее XML-документа, выполнив эту программу и передав ей в качестве параметра ссылку на файл `\\SVFPBook\\Eexamples\\Glava2\\valid_xml.xml`.

Вы получите сообщение, что у вас имеется действительный и хорошо сформированный документ. Далее сделайте то же самое с файлом `\\SVFPBook\\Eexamples\\Glava2\\well-formed_xml.xml`. Затем проведите разбор файлов `\\SVFPBook\\Eexamples\\Glava2\\non_valid_xml.xml` и `\\non_well-formed_xml.xml`.

Отметьте для себя типы ошибок при разборе "нехорошо сформированного" и "недействительного" документов.

Типы структуры XML-документов (или смерть буриданова осла)

XML-документы различаются по структуре, и это различие для нас принципиально важно. XML-документы могут быть двух структурных типов: "element-centric" и "attribute-centric". Из названий типов легко определить различие между ними. Структура документов первого типа основана на элементах, а структура документов второго типа — на атрибутах элементов. В листингах 2.4 и 2.5 приведены примеры файлов обоих типов.

Листинг 2.4. XML-документ типа "element-centric"

```
<?xml version = "1.0" encoding="Windows-1252" standalone="yes"?>
<VFPData>
  <simple>
    <id>SI_2CA0MAVOE</id>
    <itemname>iPod Nano</itemname>
    <price>230.0000</price>
    <instore>10</instore>
    <description>Apple Multimedia player</description>
  </simple>
</VFPData>
```

Листинг 2.5. XML-документ типа "attribute-centric"

```
<?xml version = "1.0" encoding="Windows-1252" standalone="yes"?>
<VFPDData>
  <simple id="SI_2CA0MAVOE"
    itemname="iPod Nano"
    price="230.0000"
    instore="10"
    description="Apple Multimedia player"
  />
</VFPDData>
```

В первом примере элемент `simple` описывается пятью дочерними элементами: `id`, `itemname`, `price`, `instore` и `description`. Во втором примере этот же элемент описывается атрибутами с именами `id`, `itemname`, `price`, `instore` и `description`, размещенными внутри тэга элемента `simple`. Если открыть оба элемента в Microsoft Excel или импортировать в курсор Visual FoxPro, то мы получим одинаковый результат. Забегая немного вперед, отметим, что функция `CursorToXML()` позволяет получить оба типа документа.

Какой же тип выбрать? Как говорится, — "it depends"! Однако если иметь в виду конкретное использование XML-документа, то выбор типа имеет принципиальное значение.

Например, в Microsoft Access при экспорте данных из документа типа "attribute-centric" создается пустая таблица. Причина проста — Microsoft Access не может импортировать данные из документов этого типа (см. описание в базе знаний Microsoft по адресу <http://support.microsoft.com/kb/285329>).

Разберемся с этим типами поподробнее, т. к. непонимание различий между структурными типами может породить ряд проблем.

Элементы или атрибуты? Вот в чем вопрос! В одном эта парочка одинакова: и элементы, и атрибуты являются контейнерами для информации. А дальше сплошные различия.

- ☐ Элемент является логическим элементом структуры.
- ☐ Атрибут является атомарным элементом структуры.
 - Если проводить аналогии с точки зрения мудрого Лиса, то элемент можно сравнить с объектом, а атрибут со свойством.
- ☐ Элемент может содержать значение или заключать в себе структуру дочерних элементов.

□ Атрибут может содержать только значения в силу своей атомарной природы.

Отсюда можно вывести первое правило для выбора структурного типа XML-документа:

"Element-centric" XML-документы предпочтительнее в случае оформления структурно-значимой информации.

Элементы могут содержать значительные объемы данных, включая тексты и значительные бинарные величины, оформленные в виде CDATA. Атрибуты ограничены в передаче больших объемов информации. Отсюда можно вывести второе правило:

"Element-centric" XML-документы предпочтительнее в случае подготовки документов с большими массивами данных, оформленных как содержание одного элемента.

Предположим, что вы собираетесь обеспечить обмен данными с филиалом или дочерней компанией посредством XML-документов и объем передаваемых данных может составить десятки тысяч записей из таблиц баз данных. Возможно, что вам потребуется обеспечить быстрый поиск необходимой информации непосредственно в XML-документе. Если документ сформирован как "element-centric", то организовать поиск достаточно просто, но если документ сформирован как "attribute-centric", вам придется перебирать поочередно все "строки" документа. Отсюда можно вывести третье правило:

Если планируется организация поиска внутри XML-документа, то лучше формировать "element-centric" XML-документы.

Порядок следования элементов в документе должен строго соответствовать DTD или схеме. Порядок следования атрибутов может быть произвольным. Отсюда можно вывести четвертое правило:

Если порядок следования данных является критически важным, то следует формировать "element-centric" XML-документы. Если порядок следования данных не имеет большого значения, то можно формировать "attribute-centric" документы.

Предположим, что у вас имеется каталог примерно такого содержания, как у приведенного ранее XML-документа, и число записей каталога составляет несколько сотен или тысяч. Для описания каталожного элемента, который характеризуется пятью параметрами, с учетом самого описываемого элемента при формировании "element-centric" XML-документа вам потребуется число тэгов, равное $6 \times 2 \times N$, где N — число записей. Соответственно, для формирования "attribute-centric" XML-документа вам потребуется N тэгов, правда, нужно учесть длины имен атрибутов, но все равно такой документ будет значительно меньше по размеру.

Отсюда вытекает следующее правило:

При передаче однородных компактных данных, составляющих в целом большой объем записей, следует формировать "attribute-centric" XML-документы.

ЗАМЕЧАНИЕ

Несмотря на все недостатки "attribute-centric" XML-документов, у атрибутов по сравнению с элементами есть одно, но очень серьезное преимущество — в версии XML 1.0 и XML-схеме только атрибуты могут иметь значение по умолчанию, присваиваемое им DTD или схемой XML-документа.

Я думаю, что вы и сами в состоянии продолжить составлять правила, по которым следует формировать тот или иной структурный тип документа, учитывая и то, что элементы, кроме наличия вложенной структуры, могут содержать атрибуты. Короче говоря, эта тема может быть бесконечной.

Можно сказать однозначно, что без человеческого фактора в выборе структурного типа XML-документа не обойтись. А значит, как сказал бы Лейбниц, буриданов осел мертв.

Экспорт из Visual FoxPro в XML

Для экспорта данных из баз данных и таблиц в Visual FoxPro предусмотрены одна функция — `CursorToXML()` и два объекта — `CursorAdapter` и `XMLAdapter`. Все эти средства отлично справляются с теми задачами, для которых они и были созданы. Однако иногда может возникнуть ситуация, когда требуется передать такую иерархическую структуру, которая не может быть создана ни одним из указанных средств. В таком случае необходимые структура и данные могут быть легко экспортированы в XML с помощью COM-объектов, таких как `Microsoft.XMLDOM` и `MSXML2.Document.<номер_версии>.0`, которые описывают объектные модели XML-документов.

Общие правила создания объектов объектной модели XML-документа

В принципе эти правила почти тождественны правилам создания объектов объектной модели HTML-документов. Да и оба DOM'a имеют множество одинаковых свойств и методов. Вне зависимости от того, что мы желаем создать — инструкцию по обработке, элемент, атрибут или комментарий.

Элемент создается в корне документа. Далее объект элемента передается тому элементу, который должен либо добавить его в качестве дочернего элемента, либо вставить перед уже существующими дочерними элементами.

Поскольку XML не имеет своего собственного словаря имен элементов, то отсутствуют и специальные методы их создания. Единственная проблема может возникнуть при создании в Visual FoxPro инструкций по обработке и деклараций.

Создание инструкции по обработке (processing instruction)

Для создания инструкции по обработке используется общее правило создания любого структурного элемента XML-документа. Однако действие по созданию инструкции по обработке зависит от того, производили вы загрузку шаблона документа или нет. Если сразу после создания документа вы загрузили в него шаблон, то используется XML DOM-метод `insertBefore()`:

```
insertBefore(  
    объект_созданного_элемента,  
    объект_элемента_перед_которым_будет_вставлен_созданный_элемент  
)
```

Если же документ создается с нуля, то используется XML DOM-метод `appendChild()`.

```
appendChild(  
    объект_созданного_элемента  
)
```

Тексты программного кода создания инструкции по обработке для обоих вариантов приведены в листингах 2.6 и 2.7.

ПРИМЕЧАНИЕ

По неизвестным причинам Visual FoxPro не желает добавлять декларацию кодировки в инструкцию по обработке при формировании выходного файла.

Листинг 2.6. Создание инструкции по обработке в пустом документе

```
Local ;  
    loXMLDOM As MSXML2.DOMDocument60, loPI  
loXMLDOM=Createobject("MSXML2.DOMDocument.6.0")  
loPI=loXMLDOM.createProcessingInstruction("xml",[version="1.0"])  
loXMLDOM.appendChild(loPI)  
? loPI.nodeType  
? loXMLDOM.XML  
Release All
```