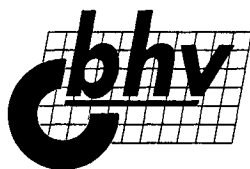


Денис Русеев

Технологии беспроводного доступа

Справочник



Санкт-Петербург

Дюссельдорф ♦ Киев ♦ Москва ♦ Санкт-Петербург

Книга посвящена технологиям, которые позволяют организовать беспроводной доступ в Интернет посредством мобильных терминалов. Рассматриваются способы управления диалогом в режиме передачи данных и в режиме голосового сообщения. Приведены описания языков WLM, WMLScript, CallXML, VoiceXML, HDML, позволяющих создавать не только WAP-совместимые гипертекстовые страницы, размещаемые на малых экранах сотовых телефонов, но и приложения, дающие возможность управления голосовым сообщением. На основе примеров показаны возможность реализации команд, представленных языков и их ограничения, текущие и перспективные области применения новых технологий.

*Для широкого круга технических специалистов,
работающих над созданием беспроводных приложений*

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зав. редакцией	<i>Анна Кузьмина</i>
Редактор	<i>Марина Чуканова</i>
Компьютерная верстка	<i>Юлии Серебрянниковой</i>
Корректор	<i>Татьяна Звертановская</i>
Дизайн обложки	<i>Игоря Цырульников</i>
Зав. производством	<i>Николай Тверских</i>

Русеев Д. С.

Технологии беспроводного доступа. Справочник. — СПб.: БХВ-Петербург, 2002. — 352 с.: ил.

ISBN 5-94157-132-1

© Д. С. Русеев, 2002

© Оформление, издательство "БХВ-Петербург", 2002

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.12.01

Формат 70×100¹/₁₆. Печать офсетная. Усл. печ. л. 28,38.

Тираж 3000 экз. Заказ № 1394

"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29

Гигиеническое заключение на продукцию, товар, № 77 99.1 953 П.950.3.99
от 01.03.1999 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН.
199034, Санкт-Петербург, 9 линия, 12

Содержание

Предисловие.....	1
Глава 1. Конвергентные сети и услуги.....	3
Введение.....	3
Процесс интерпретации.....	7
Новые возможности.....	9
Появление новых XML-диалектов.....	9
Особенности внедрения беспроводных услуг.....	10
Экономические факторы.....	10
Технические трудности создания беспроводных сервисов.....	14
Работа мобильного телефона.....	16
Глава 2. Справочник по WML.....	22
Тег <i><a></i>	22
Тег <i><access></i>	23
Тег <i><anchor></i>	23
Тег <i></i>	24
Тег <i><big></i>	24
Тег <i>
</i>	25
Тег <i><card></i>	25
Тег <i><catch></i>	27
Тег <i><do></i>	28
Тег <i></i>	31
Тег <i><exit></i>	31
Тег <i><fieldset></i>	31
Тег <i><go></i>	32
Тег <i><head></i>	34
Тег <i><i></i>	34
Тег <i></i>	34
Тег <i><input></i>	37
Тег <i><link></i>	39
Тег <i><meta></i>	40
Тег <i><noop></i>	41

Тег <i><onevent></i>	42
Тег <i><optgroup></i>	43
Тег <i><option></i>	43
Тег <i><p></i>	44
Тег <i><postfield></i>	44
Тег <i><prev></i>	45
Тег <i><receive></i>	45
Тег <i><refresh></i>	46
Тег <i><reset></i>	46
Тег <i><select></i>	46
Тег <i><send></i>	48
Тег <i><setvar></i>	48
Тег <i><small></i>	48
Тег <i><spawn></i>	49
Тег <i></i>	50
Тег <i><table></i>	50
Тег <i><td></i>	51
Тег <i><tr></i>	51
Тег <i><template></i>	51
Тег <i><throw></i>	52
Тег <i><timer></i>	53
Тег <i><u></i>	54
Тег <i><wml></i>	54
Краткое описание требований к GSM-приложениям в m-services	54
Правила преобразования текстов WML версий 1.3 и 2.0	56
Глава 3. Справочник по WMLScript	58
Операторы присвоения	58
Арифметические операторы	59
Логические операторы	60
Операции действия со строками	61
Операторы сравнения	61
Оператор <i>typeof</i>	62
Языковая библиотека	64
Библиотека <i>Lang</i>	64
Библиотека <i>Float</i>	76
Библиотека <i>String</i>	80
Библиотека <i>URL</i>	90
Библиотека <i>WMLBrowser</i>	97
Библиотека <i>Dialogs</i>	101
Библиотеки <i>WTA</i>	102
Библиотека <i>WTAPublic</i>	102
Библиотека <i>WTAVoiceCall</i>	103

Библиотека <i>WTANetText</i>	104
Библиотека <i>WTAPhoneBook</i>	105
Библиотека <i>WTACallLog</i>	107
Библиотека <i>WTAMisc</i>	107
Вспомогательные библиотеки компаний-производителей	108
Библиотеки <i>Console</i>	108
Библиотека <i>Debug</i>	110
Коды наборов символов	110
Глава 4. Справочник по HDML	112
Тег <i><HDML></i>	112
Тег <i><A></i>	113
Тег <i><ACTION></i>	115
Тег <i><ENTRY></i>	118
Тег <i><CHOICE></i>	120
Тег <i><CE></i>	121
Тег <i><DISPLAY></i>	124
Тег <i><NODISPLAY></i>	125
Тег <i></i>	125
Глава 5. Голосовые диалекты XML	126
Телефонная сеть как платформа для разработки приложений	126
Зачем нужна платформа разработки?	126
Основные возможности	128
Пример приложения	128
Стандартизация VoiceXML и CallXML	129
Роль мета-языка XML	131
Основы Web-телефонии	131
Провайдеры приложений	134
Заключительные замечания	135
Глава 6. Обзор языка VoiceXML	136
Обзор языка	136
Цели языка	136
Основные понятия языка XML	136
Пример XML-документа	137
XML-комментарии	139
Проверка корректности XML-документа	139
Зарезервированные символы и разделы CDATA	140
Сравнение XML и HTML	140
Где найти дополнительную информацию?	141
Основные VoiceXML	141
Структура голосового приложения, созданного с использованием языка VoiceXML	141

Описание элемента <code><audio></code>	143
Управление процессом обработки вызова.....	143
Верификация и отладка приложений.....	144
Верификация VoiceXML-документов.....	145
Ошибки времени исполнения.....	145
Управление выводом отладочной информации.....	146
Прослушивание голосового сообщения от входящего телефонного вызова.....	147
Необходимые параметры для создания интерактивности.....	147
Создание резерва памяти для ответа звонящего.....	148
Подсказка звонящему перед вводом информации.....	148
Ограничение ввода.....	148
Синтаксис грамматики.....	149
Введение в события VoiceXML.....	153
Грамматики.....	160
Описание грамматик.....	160
Повторное использование грамматик Tellme.....	161
Переменные.....	161
Объявление переменных.....	162
Объявление переменных с использованием элемента <code><field></code>	163
Присваивание значений переменным.....	163
Описание области действия.....	163
Системные переменные сеанса.....	164
Обеспечение нового поведения области действия.....	164
Перекрывающиеся области действия переменных и деактивация переменных.....	165
Обращение к неактивным переменным.....	166
Извлечение данных из переменных.....	166
Обращение к аудиоданным посредством переменных.....	168
Интеграция JavaScript.....	169
Использование выражений JavaScript.....	169
Определение блоков JavaScript.....	170
Доступ к переменным VoiceXML из JavaScript.....	173
Область действия блока <code><script></code>	173
Обработка событий.....	175
Использование стандартных событий обработки ошибок.....	175
Использование событий, определенных приложением.....	176
Построение VoiceXML-приложений.....	181
Задание корневого документа приложения.....	183
Ссылка на корневой документ приложения.....	184
Использование вложенных диалогов.....	185
Передача параметров к вложенному диалогу.....	186
Использование сценариев на стороне сервера для передачи данных к поддиалогу.....	187

Возврат данных звонящему	188
Исполнение поддиалога на стороне сервера	189
Данные приложения и JavaScript	191
Функция <code>xmllog</code>	191
Объект <code>xmldata</code>	191
Глава 7. Описание тегов VoiceXML	195
Теги	195
Теги документа	195
Теги формы	195
Теги поля	196
Теги меню	196
Исполняемые теги	197
Тег <code><assign></code>	197
Тег <code><audio></code>	198
Тег <code><block></code>	200
Тег <code><catch></code>	201
Тег <code><choice></code>	203
Тег <code><clear></code>	205
Тег <code><confirm></code>	206
Тег <code><debug></code>	208
Тег <code><default></code>	209
Тег <code><disconnect></code>	210
Тег <code><dtmf></code>	211
Тег <code><error></code>	213
Тег <code><exit></code>	214
Тег <code><field></code>	216
Тег <code><filled></code>	221
Тег <code><foreach></code>	222
Тег <code><form></code>	225
Тег <code><gosub></code>	228
Тег <code><goto></code>	229
Тег <code><grammar></code>	231
Тег <code><help></code>	236
Тег <code><if><elseif><else></code>	238
Тег <code><link></code>	239
Тег <code><listen></code>	241
Тег <code><log></code>	242
Тег <code><menu></code>	243
Тег <code><meta></code>	245
Тег <code><noinput></code>	246
Тег <code><nomatch></code>	248
Тег <code><param></code>	250
Тег <code><pause></code>	251

Тег <i><prompt></i>	252
Тег <i><property></i>	254
Тег <i><record></i>	258
Тег <i><reprompt></i>	260
Тег <i><result></i>	261
Тег <i><return></i>	263
Тег <i><script></i>	264
Тег <i><subdialog></i>	265
Тег <i><submit></i>	268
Тег <i><throw></i>	269
Тег <i><transfer></i>	271
Тег <i><value></i>	274
Тег <i><var></i>	275
Тег <i><vxml></i>	276
Глава 8. Справочник по CallXML.....	278
Обзор	278
Сравнение CallXML с HTML.....	278
Сравнение CallXML и VoiceXML	279
Пример CallXML-документа	280
Если вы знаете HTML.....	280
Элементы манипулирования переменными	281
Тег <i><assign></i>	281
Тег <i><clear></i>	283
Тег <i><clearDigits /></i>	283
Тег <i><goto></i>	283
Тег <i><run></i>	285
Тег <i><sendEvent></i>	286
CallXML — выполнение действий с вызовом	287
Тег <i><answer/></i>	287
Тег <i><hangup/></i>	288
Тег <i><call></i>	288
Тег <i><conference></i>	289
Тег <i><waitForConferenceEnd /></i>	289
Тег <i><wait></i>	290
CallXML — действия уровня среды распространения.....	290
Тег <i><getDigits></i>	290
Тег <i><play... /></i>	292
Тег <i><recordAudio></i>	293
Тег <i><text></i>	295
Тег <i><addChannel></i>	296
Тег <i><removeChannel></i>	296
Элементы уровня блока	296

Тег <code><block></code>	296
Тег <code><menu></code>	297
Тег <code><inputDigits></code>	299
Тег <code><inputAudio></code>	300
Элементы задания событий CallXML	302
Тег <code><onAnswer></code>	302
Тег <code><onCallFailure></code>	303
Тег <code><onError></code>	303
Тег <code><onHangup/></code>	304
Тег <code><onMaxTime /></code>	304
Тег <code><onMaxPages /></code>	305
Тег <code><onMaxSilence /></code>	305
Тег <code><onTermDigit /></code>	305
Тег <code><onExternalEvent></code>	306
Тестирование и отладка	306
Тег <code><simline></code>	306
Пример приложения	307
Приложение 1. Особенности платформы Tellme	309
Отличия от стандарта VoiceXML 1.0	309
Интерпретация тегов	309
Приложение 2. Особенности платформы Nuance	317
Приложение 3. Преобразование из WML 1.3 в WML 2.0	320
Таблица преобразования XSLT	320
Использованные ресурсы и литература	334

Предисловие

Беспроводные технологии быстро входят в нашу жизнь. Появление новых видов услуг оказывает все большее влияние на наши представления о возможностях традиционных средств связи. Одной из таких услуг является беспроводной доступ к данным, которому посвящена эта книга.

Точнее она посвящена описанию языков разметки, используемых при создании услуг доступа к данным. Книга состоит из двух частей, с каждой из которых можно ознакомиться независимо.

В первой части, охватывающей главы с первой по четвертую, представлен материал, который может быть полезен широкой аудитории технических специалистов, работающих в области создания услуг на основе WAP-технологии.

Первая глава представляет краткое введение в предметную область. В следующей главе приводится описание тегов языка WML версии 1.3. В третьей главе приведены основные сведения о языке WMLScript с примерами тестовых приложений и внешним видом результатов работы этих тестов. Четвертая глава содержит детальные сведения о языке разметки HDML, который был предшественником существующего стандарта, и все еще используется во многих телефонных аппаратах.

Глава 5 открывает вторую часть справочника, которая посвящена разработке голосовых приложений. В русскоязычной литературе этот материал практически недоступен, и книга частично заполняет пробел. В отличие от первой части справочника, который предполагает некоторое знакомство с особенностями архитектуры WAP-приложений, вторая часть не требует начальной подготовки.

Пятая глава вводит читателя в область создания Web-ориентированных голосовых услуг, раскрывая возможности языков CallXML и VoiceXML.

Шестая глава является кратким руководством по созданию голосовых приложений на VoiceXML. Материал этой главы следует диалекту языка, принятому в компании TellMe.

Полный список тегов языка Voice XML представлен в седьмой главе.

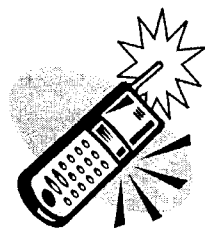
Восьмая глава дает представление о CallXML. В этой главе приведены основные теги языка, а в конце представлен текст простого приложения.

В заключительном разделе справочника приведены характеристики диалектов языка VoiceXML у компаний Voxeo и TellMe и их сравнение со стандартом.

Последнее приложение включает таблицу стилей, которую можно использовать в процессе автоматического преобразования текстов на языке WML из стандарта 1.1 к стандарту 2.0.

Глава 1

Конвергентные сети и услуги



Введение

Последние 10 лет стали годами технологических инноваций в области предоставления услуг телефонии, беспроводного доступа к данным и Интернета. Использование этих технологий происходит все более быстрыми темпами. Если для того, чтобы радио смогло набрать первые 300 миллионов радиослушателей потребовалось 38 лет, телевидение получило первые 300 миллионов зрителей за 15 лет, 300 миллионов пользователей персональных компьютеров и Интернета появились за 5 лет, то существующие прогнозы по беспроводному доступу к данным утверждают, что первые 300 миллионов абонентов будут набраны всего за 3 года. Это впечатляет не только обывателей.

Внешне процессы развития сетей выглядят как появление принципиально новых идей. В то же время эти идеи часто являются отражением единого процесса, который можно назвать *конвергенцией* и *взаимопроникновением технологий*.

Замечание

Для того чтобы избежать неясностей, последующие два раздела будут следовать терминологии и подходу, изложенному в [15].

В самом деле, в области традиционной телефонии все большее число производителей предлагает решение Intelligent Network (Интеллектуальная сеть) для организации современных услуг в телефонных сетях. Одним из основных принципов данного решения можно было бы назвать построение такого метода управления коммутационным оборудованием, который не зависит от типа сети и от набора предоставляемых в ней услуг. Обычно этот способ управления основан на IP-сети в качестве опорной сети управляющего контура (рис. 1.1).

Очень похожий процесс наблюдается в сети Интернет, где услуги IP-телефонии и другие услуги, основанные на QoS (Quality of Service, услуга с гарантированным качеством), предъявляют новые требования к используемой опорной сети. Обычно эти требования выражаются в обеспечении достаточной пропускной способности для заданного набора потоков данных

заранее фиксированных направлений в каждом узле сети. При большом количестве управляемых узлов в такой сети необходимо наличие специальных алгоритмов управления ими. Попытка построить систему управления вышеописанного типа заставляет задуматься над несколькими возможностями, одной из которых может быть способ управления, аналогичный технологии использования Java-апплета во всемирной паутине. Рассмотрение подобных возможностей привело к созданию так называемых *активных сетей*. В активных сетях программные управляющие компоненты загружаются из узлов той же самой сети, которая является опорной для базового набора услуг. Основными принципами снова становятся независимость от типа сети и независимость от набора предоставляемых услуг. Как видно, одни и те же основные принципы используются для построения сетей и для создания технологии управления сетями, предоставляющими услуги.

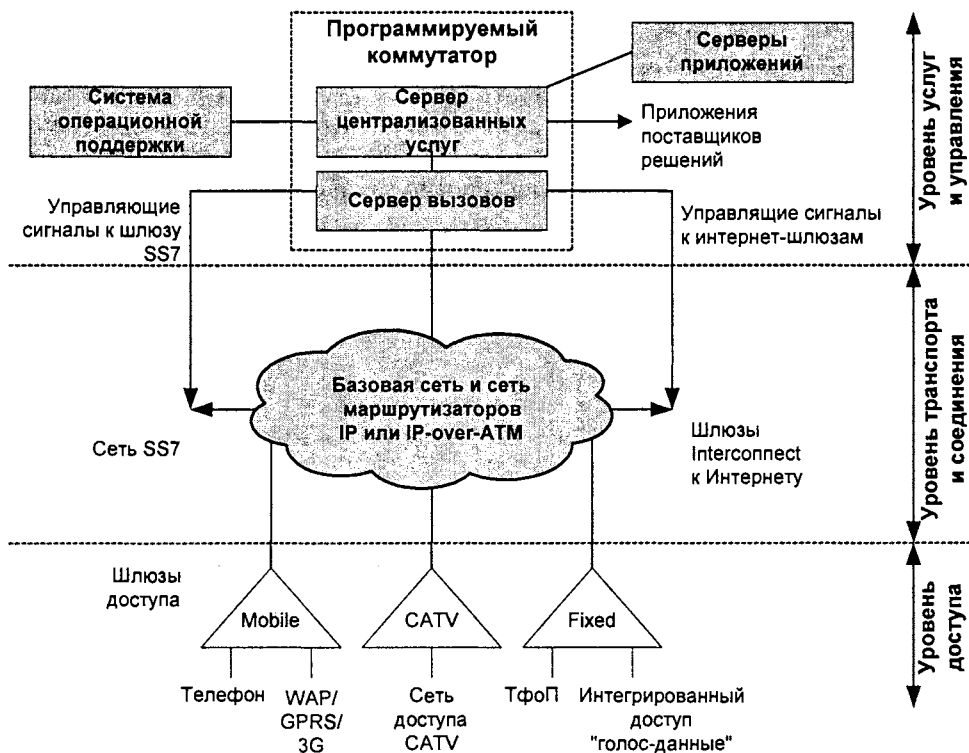


Рис. 1.1. Использование сервера приложений для управления услугами в интеллектуальной сети

В [15] было показано, что взаимопроникновение технологий сетей доступа различных стандартов приводит к появлению похожих методов управления

сеть и к появлению новых видов услуг. Услуги, основанные на конвергенции PSTN (Public Switched Telephone Network, коммутируемая телефонная сеть общего пользования) и Интернета, сегодня разрабатываются двумя группами: PSTN Internet Interworking (PINT) и PSTN/IN Requesting Internet Services (SPIRITS) из IETF (Internet Engineering Task Force, группа разработки стандартов в лаборатории Bell). Группа PSTN Internet Interworking (PINT) предлагает следующие специфические телефонные услуги.

- ❑ **Запрос на вызов** — запрос, который посылается с некоторого IP-адреса с целью установления соединения от телефона *A* к телефону *B*.
- ❑ **Запрос на факс** — запрос от хоста IP на отправку факса. Сам запрос может содержать или не содержать данные факса.
- ❑ **Запрос на прослушивание контента** — запрос, который отправляется с некоторого IP-адреса с целью проиграть абоненту телефонной сети предварительно подготовленное сообщение.

Данные услуги входят в документ RFC 2848 и используют расширения SDP и SIP.

Группа PSTN/IN Requesting Internet Services (SPIRITS) предлагает такие услуги телефонной сети, которые требуют взаимодействия с IP-сетью:

- ❑ **Internet Call waiting** — ожидание вызова при работе с Интернетом;
- ❑ **Internet caller ID delivery to the called party** — доставка идентификатора клиента вызываемой стороне;
- ❑ **Internet Call forwarding** — переадресация интернет-звонков. Служит для переадресации телефонного вызова на другой номер, если основной номер занят доступом в сеть;
- ❑ **Internet Call center** — услуги Информационного интернет-центра.

Оба предлагаемых стандарта предполагают использование серверов в сети для обеспечения необходимой функциональности и используют SIP (Session Initiation Protocol, протокол установления сеанса) для взаимодействия с IP-клиентом. Однако протокол взаимодействия с коммутационным оборудованием не специфицирован.

Как видим, взаимопроникновение технологий PSTN и интернет-сетей приводит не только к появлению совершенно новых услуг, но и дает отправную точку технологиям, улучшающим гибкость уже существующих услуг. Одной из таких групп услуг являются автоматические корпоративные службы голосового сопровождения пользователей. Работа большинства из существующих в настоящее время сервисных сетей за рубежом основана на двух принципах:

1. Возможность вынесения управляющих процедур от коммутационной сети.
2. Использование языка XML (Extensible Markup Language, язык разметки с расширенными возможностями) в качестве точки организации сервиса

содержания, работающего по протоколу HTTP (HyperText Transfer Protocol, протокол передачи гипертекста).

Второй подход имеет несколько реализаций (Voxeo, Tellme, BeVocal, VoiceGenie), которые позволяют создавать услуги нового типа. Используемые в этих сетях XML-диалекты будут кратко описаны в *главах 5—8*.

Для того чтобы завершить общую картину, необходимо указать влияние услуг беспроводного доступа к данным. Место стыка телефонных сетей с Интернетом и беспроводными сетями в настоящее время требует использования шлюза для организации сервисной сети. В роли этого шлюза может выступать сервер i-mode¹ или шлюз WAP-протокола² (Wireless Application Protocol, протокол беспроводных приложений). В любом случае, опорной сетью снова является IP-сеть, которая представляет собой транспорт для доставки содержания к серверу преобразования.

Таким образом, процесс конвергенции услуг заключается в появлении общего поля взаимодействия нескольких существующих сетей. Это общее поле является источником и основой их взаимодействия и появления новых видов услуг. Общая картина приведена на рис. 1.2.

Частью вышеописанного процесса конвергенции является создание сервисных сетей, использующих в качестве платформы интерпретирующие решения на базе различных диалектов XML. В тех областях, которые обозначены, в роли этих диалектов выступают:

- ❑ CallXML (Voxeo network) — язык разметки, предложенный компанией Voxeo и реализованный в сети этого оператора;
- ❑ VoiceXML (Tellme, Voxeo, NUANCE, Speechworks, BeVocal, IBM, Motorola, VoiceGenie) — язык разметки, используемый для создания голосовых приложений в сетях, поддерживающих решения компаний;
- ❑ WAP WML-WTA (WAP FORUM) — подмножество языка WML, обеспечивающее функциональность WTA (Wireless Telephony Application, приложения беспроводной телефонии) в стандарте WAP (Wireless Application Protocol, протокол беспроводных приложений);
- ❑ WDDX (Allaire) — протокол, который поддерживается в решениях компании Allaire при создании Web-приложений;
- ❑ XML (Microsoft) — язык разметки с расширенными возможностями.

¹ i-mode — служба оператора DoCoMo в Японии, которая позволяет выполнять доставку Internet-содержания, представленного в формате compact HTML, на экраны мобильных телефонов.

² Протокол, предназначенный для создания приложений, которые можно просматривать с экранов мобильных терминалов.

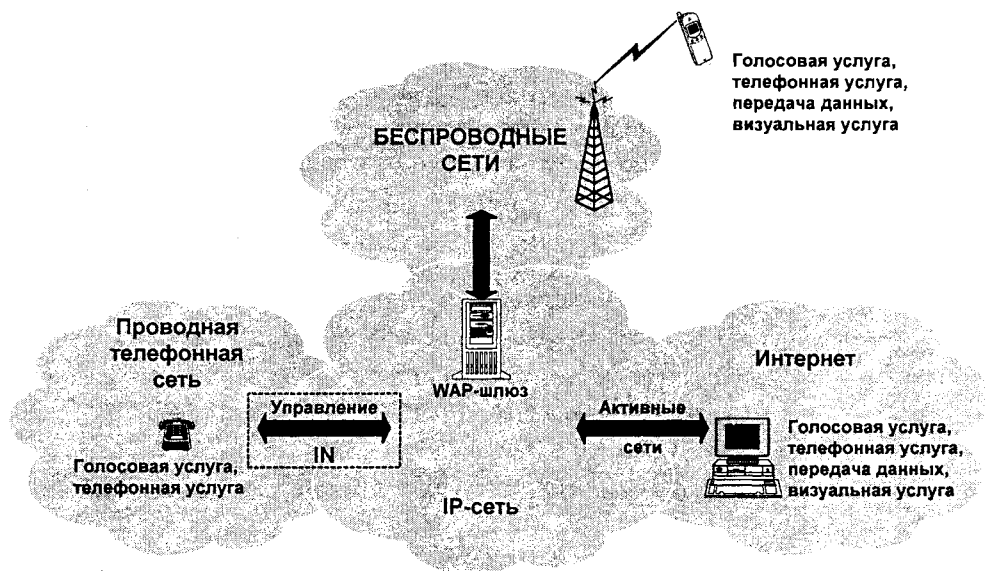


Рис. 1.2. Схема конвергенции

В основе всех этих диалектов лежит технология отделения процессов оказания услуги (которая часто называется интерпретацией или превращением содержания), процесса доставки и процесса управления ее созданием. Рассмотрим кратко каждый из них.

На рис. 1.3 показаны возможные варианты взаимодействия пользователя с сервисными серверами, размещенными в сети Интернет.

Процесс интерпретации

Можно видеть, что процесс интерпретации обычно выполняется в точке доступа к услуге и может быть выполнен несколькими способами. Одним из наиболее показательных, возможно, будет пример автоматического голосового интерфейса.

В самом общем виде этот интерфейс может быть организован по двум различным каналам доступа:

- ☐ сеть Интернет (при управлении голосом);
- ☐ телефонная линия.

Если доступ выполняется через Интернет, то необходим интерпретатор, который будет озвучивать на ПК пришедший пакет голосового сопровождения диалога. Этот интерпретатор, очевидно, будет работать на том же ПК, с которого осуществляется запрос на получение услуги.

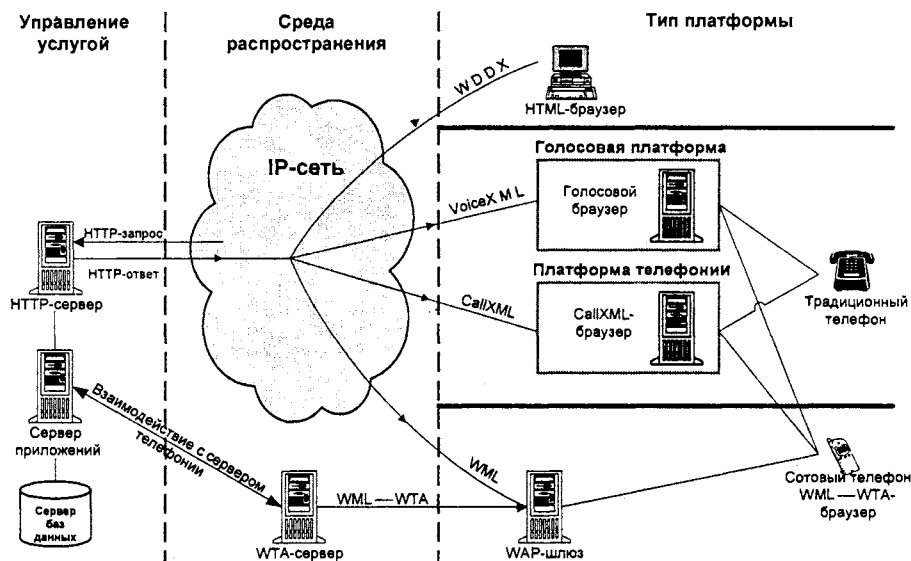


Рис. 1.3. Варианты взаимодействия пользователя с серверами предоставления услуг

В случае доступа к услуге с обычного телефона необходима сервисная платформа, на которой будет выполняться преобразование в голосовое сообщение. В роли такой платформы может выступать оборудование компаний CISCO, IBM, COMPAQ, оснащенное голосовой шиной и программным обеспечением голосового сервера. Последний и должен выполнять функцию интерпретации и генерации голосового сообщения — ответа на запрос.

Таким образом, процесс интерпретации голосового интерфейса может выполняться как на стороне клиента (ПК), так и в промежуточной точке (сервисная платформа).

Процесс доставки содержания в точку интерпретации осуществляется по HTTP-протоколу и в настоящее время широко используется практически во всех реализациях рассматриваемого класса. В большинстве случаев в качестве транспорта используется протокол TCP/IP (Transmission Control Protocol/Internet Protocol, протокол управления передачей/протокол Интернета, стек протоколов Интернета).

Процесс управления созданием содержания, как правило, реализован на основе HTTP-сервера со встроенным сервером приложений.

Сервер приложений по запросу производит генерацию XML-пакетов нужного диалекта. Созданный пакет отправляется на платформу, с которой пришел запрос. Далее, в зависимости от типа платформы пакет превращается в сервис на самой платформе (VoiceXML или CallXML) или отправляется для превращения к браузеру, расположенному в самом терминале (WML-WTA или WDDX).

Во всех случаях точкой создания услуги является сервер приложений, обеспечивающий работу всех новых приложений, управляемых данными.

Поскольку генерация XML-содержания во всех диалектах может выполняться на однотипных платформах, оснащенных возможностями межсерверного взаимодействия, не существует никаких препятствий к созданию комбинированных услуг, в которых сильные стороны одного диалекта могут сочетаться с сильными сторонами других XML-подобных языков или компенсировать их слабости.

В последующих главах будут приведены краткие характеристики языков WML, CallXML, VoiceXML и языка описания сценариев беспроводного доступа к данным WMLScript.

Новые возможности

Появление новых XML-диалектов

Конвергенция услуг телефонии и услуг Интернета породила волну новых XML-диалектов, каждый из которых занимает свое особое место в мире телекоммуникаций.

Например, беспроводные приложения используют языки XHTML, HTML, WML или WMLScript, а телефонные голосовые услуги, размещенные на серверах в Интернете, обычно разрабатываются на основе VoiceXML, CallXML, XTML.

Появление в августе 2001 года черновой версии стандарта WAP 2.0 поставило перед разработчиками требование обратной совместимости с приложениями, написанными для стандарта WAP версии 1.1. К моменту написания этих строк спецификация WAP 2.0 еще не была принята в качестве стандарта. Поэтому полное описание версии 2.0 языков WML и WMLScript не вошло в настоящее издание. Тем не менее, требование обратной совместимости стандартов красной нитью проходит по всем новым инициативам лидеров телекоммуникационного рынка, включая m-services (Ассоциации GSM), MET (Mobile Electronic Trading — Мобильная электронная торговля) и инициативы организации WAP FORUM (<http://www.wapforum.org>). Поэтому автор посчитал уместным указать в *главах 1—2* правила преобразования документов WML 1.3 и WML 2.0, а в *приложении 3* привести таблицу стилей, автоматически выполняющую такое преобразование.

MET (<http://www.mobiletransaction.org>) — описывает функциональность услуг мобильной коммерции, которая в соответствии с документами инициативной группы может выполняться на основе смешанной технологии WAP и Bluetooth.

M-services (<http://www.openwave.com/m-services/index.html>) — определяет требования и рекомендации по созданию услуг беспроводного доступа к данным в сетях GSM. В частности, рекомендуется при создании сервисов учитывать особенности сотовых терминалов, работающих в сети оператора сотовой связи.

Особенности внедрения беспроводных услуг

Одной из самых горячих тем, обсуждаемых в российской и западной прессе в 2000 году, была тема беспроводного доступа к данным на основе WAP-технологии. Широко разрекламированная на Западе, WAP-технология, в соответствии с заявлениями журналистов, должна была в самые короткие сроки покорить абонентов сотовых сетей и обеспечить операторам сотовой связи новый источник прибылей.

В качестве привлекательного для оператора сотовой связи свойства нового сервиса приводилась перспектива обеспечения не только текущих, но и будущих прибылей, основанных на плате за "содержание". В недалеком будущем плата за "содержание", скорее всего, значительно превысит плату за доступ в сеть. "Содержание" должно обладать для абонентов высоким уровнем привлекательности, который можно достичь уникальными свойствами сервисов — здесь и сейчас. WAP-услуги на основе самых последних стандартов WTA и PUSH-технологий дают такую возможность.

Среди основных достоинств технологии указываются:

- ☐ независимость от типа сети;
- ☐ независимость от типа сотового терминала;
- ☐ возможность реализации проектов для новой аудитории без значительных дополнительных инвестиций.

Многие из этих обещаний оказались исполненными уже сегодня. Абоненты сотовых сетей практически всех развитых стран, в том числе и России, могут получать доступ к данным Интернета и интранет, пользуясь обычными сотовыми телефонами. Однако до победного марша новой технологии еще далеко. Ниже предпринята попытка анализа основных трудностей внедрения WAP-услуг на российском рынке сотовой связи.

Экономические факторы

Первой и наиболее важной причиной является сама природа носителей данных в беспроводных сетях. Известно, что в качестве носителей WAP-протокола в беспроводных сетях разных стандартов могут выступать различ-

ные радиointерфейсы. Список наиболее часто используемых носителей WAP-протокола приведен в табл. 1.1.

Таблица 1.1. Носители WAP-протокола, используемые в беспроводных сетях с различными радиointерфейсами

Стандарт беспроводной сети	Носитель данных, используемый для WAP-протокола
GSM	<ul style="list-style-type: none"> • SMS (Short message system — система коротких сообщений); • USSD (передача данных в режиме передачи команд); • C-SData (Circuit Switch Data — передача данных по голосовым каналам связи); • GPRS (General Packet Radio System — услуги универсальной пакетной передачи)
IS-136	<ul style="list-style-type: none"> • C-SData (Circuit Switch Data — передача данных по голосовым каналам связи); • Packet (передача данных в пакетном режиме)
CDMA	<ul style="list-style-type: none"> • SMS (Short message system — система коротких сообщений); • C-SData (Circuit Switch Data — передача данных по голосовым каналам связи)
PDC	<ul style="list-style-type: none"> • C-SData (Circuit Switch Data — передача данных по голосовым каналам связи); • Packet (передача данных в пакетном режиме)
PHC	<ul style="list-style-type: none"> • C-SData (Circuit Switch Data — передача данных по голосовым каналам связи)

Можно видеть, что наиболее часто используемым вариантом беспроводного интерфейса является C-SData (Circuit Switch Data). Это значит, что данные передаются по тем же самым каналам связи, что и голос, и в большинстве случаев конкурируют с ним за каналные ресурсы.

Вышеописанная конкуренция обостряется в периоды быстрого роста абонентской базы у сотовых операторов, когда возникают трудности обеспечения нужного качества связи. В эти периоды оператор больше заботится о наличии достаточной пропускной способности каналов и обеспечении желаемого уровня покрытия, возможности которых он успешно продает своим абонентам по высоким ценам. Конечно, предоставление услуг по заниженным ценам в такие периоды нежелательно. Планирование новых затрат на создание и обеспечение дополнительных услуг практически всегда оценивается как нецелесообразное.

Поэтому первые внедрения WAP-услуг у российских операторов сотовой связи приводили к низкому качеству и породили первые сомнения у пользователей, которые обнаружили, что:

- ☐ в реализованных версиях услуг нет доступа ко всем возможностям Интернет-ресурсов;
- ☐ время установления соединения занимает в сетях некоторых сотовых операторов более 35 секунд, а полный цикл получения данных о прогнозе погоды в регионе составляет 2,5 минуты. В условиях, когда за пользование услугой нужно платить, как за обычный телефонный разговор, оплата длительных периодов ожидания не выглядит привлекательной;
- ☐ стоимость телефонов, оснащенных WAP-браузером, необоснованно высока, хотя в основном используется только телефония.

Сторонники новой технологии обычно возражают, что время установления соединения зависит от технической реализации точки входа. Например, в компании КиевСтар это время составляет 7 секунд, а в Норвегии у оператора Telenor — 3,5 секунды. Дальнейшая скорость работы зависит от используемых каналов доступа в Интернет из локальных сетей сотовых операторов. При использовании хороших каналов время выбора следующей страницы редко превышает 10 секунд.

Известно также, что стоимость передачи данных в некоторых регионах (сеть Северо-Западный GSM) составляет 5 центов в минуту, что сравнимо по стоимости с доступом по обычным проводным линиям и в 4 раза ниже стоимости звонка.

Последнее обстоятельство позволяет создавать выездные службы, использующие беспроводной доступ к корпоративным данным. Применение такого подхода может давать значительный экономический эффект, который может быть оценен, например, на основе методик, приведенных в Интернете по адресу <http://www.phone.com>. Более полный пример подобного анализа можно найти в книге [15].

Если отвлечься от налета эмоциональности, свойственного подобному обмену мнениями, то можно объективно утверждать, что WAP является достаточно сложной услугой, часто требующей интеграции с несколькими службами, работающими в ЛВС оператора сотовой связи и за ее пределами. Организация службы беспроводного доступа операторского класса является гораздо более дорогим проектом, нежели создание корпоративных приложений.

Как всякая сложная услуга, она неэластична к ценам и обладает следующими дополнительными свойствами:

- ☐ трудное продвижение;
- ☐ сложное модифицирование;
- ☐ дорогое сопровождение;

- ☐ непредсказуемое влияние на имидж компании;
- ☐ в том случае, когда не используются элементы мобильной коммерции, каждый узкий сегмент рынка со специфическими потребностями пользователей должен иметь собственный специализированный сервис;
- ☐ сложное позиционирование в общей корзине услуг оператора для слабоизученной, хотя и перспективной аудитории (характерно для российского рынка).

Указанные особенности отрицательно влияют на рынок массовых услуг доступа к данным. В то же время корпоративный сегмент рынка в основном свободен от этих недостатков и имеет другие характеристики. Кратко их можно представить следующим списком:

- ☐ высокий уровень индивидуальности;
- ☐ наличие внутренних критериев эффективности проектов;
- ☐ высокая мотивация к сложному, но целевому использованию;
- ☐ сотрудники знают свое географическое положение и задачи;
- ☐ нет проблем тарификации содержания;
- ☐ вся информация извлекается из корпоративной базы данных;
- ☐ технологические процессы обеспечения информацией сотрудников компании обычно имеют уникальные свойства, которые необходимо учитывать в процессе внедрения новых технологий.

Для того чтобы внедрение корпоративных приложений было экономически целесообразным, необходимо наличие трех составляющих:

1. Техническая возможность выхода в интернет-пространство с использованием беспроводного терминала.
2. Умеренная стоимость проектов создания службы беспроводного доступа к корпоративным базам данных.
3. Выгодные тарифы операторов сотовой связи.

Сегодня практически все лидеры сотового рынка обеспечивают первую из указанных возможностей. Выполнение проектов беспроводного доступа к корпоративным данным по умеренным ценам также становится реальностью.

Таким образом, в настоящее время наиболее привлекательным сегментом рынка беспроводного доступа к данным в России является рынок корпоративных систем. Однако даже в этом сегменте существуют условия, не позволяющие рассчитывать на появление новых возможностей. И маловероятно, что в самое ближайшее время ситуация изменится.

Слово "маловероятно" в этом контексте имеет оттенок двойственности. С одной стороны, все вышесказанное подводит к выводу о том, что технология беспроводного доступа к данным не будет развиваться ожидаемыми

темпами. С другой стороны, создатели WAP-протокола относили следующий ряд ситуаций на технологическом рынке к маловероятным:

- ☐ маловероятно, что используемый частотный спектр со временем станет более дешевым, чем в настоящее время;
- ☐ маловероятно, что закончится война стандартов;
- ☐ совершенно невероятно, что уровень конкуренции в среде операторов сотовой связи снизится;
- ☐ маловероятно, что пользователи сотовых телефонов кардинально изменят свое отношение к беспроводным устройствам. Современные пользователи имеют специфические требования к сотовым телефонам и предоставляемым услугам. Даже в высокоскоростных сетях более 40% всех востребованных услуг будут основаны на низкоскоростном доступе к данным;
- ☐ маловероятно, что характеристики рынка сотовых телефонов и рынка компьютеров будут в одинаковой степени чувствительны к ценам;
- ☐ маловероятно, что в ближайшие три года емкость беспроводных каналов связи будет расширена настолько, что внедрение услуг высокоскоростного доступа к данным станет массовым явлением для абонентов, использующих возможности беспроводных терминалов.

В оставшейся части главы приведено детальное описание технических трудностей создания беспроводных услуг на основе WAP-стандарта.

Технические трудности создания беспроводных сервисов

- ☐ **Трудно вводить текст.** Это один из наиболее важных факторов при проектировании мобильных устройств. По умолчанию в телефонах используется метод ввода, при котором на каждой клавише имеется три связанных буквы, это трудно для всех, за исключением возможно небольшого количества пользователей экспертов. Более совершенные процедуры обеспечения ввода, такие как Tegic's T9, часто совершенно бесполезны, так как нацелены на ввод естественных фраз языка и не подходят для ввода данных при заполнении форм или e-mail-адресов. Поэтому разработчику приходится постоянно заботиться о предоставлении пользователю альтернативы вводу текста. Кроме того, необходимы специальные усилия, направленные на минимизацию объема вводимого текста.
- ☐ **Эфирное время стоит денег.** В Европе WAP работает через CSD. Каждый сеанс тарифицируется на минутной основе, также как ISP несколько лет назад. Кто-то платит поминутно сразу, кто-то имеет несколько бесплатных секунд. При этом оплата передачи данных часто выше, чем оплата голосового трафика. Пакетная коммутация, в конце концов, изменит эту ситуацию, однако CSD является нашим стандартом для ближайшего будущего.

- ❑ **Использование является критическим.** Мобильные телефоны оптимизированы для использования в качестве голосовых устройств, но они совершенно не подготовлены для эффективной работы в качестве устройств обработки данных. В них используется относительно малый размер экрана и слабая обратная навигация. Трудности ввода текстовой информации становятся критическим фактором для успеха любой услуги. Разработчики вынуждены принять во внимание, что пользователи уже встретились со всеми этими проблемами, и не должны заставлять испытывать дополнительные трудности при работе с новыми услугами.
- ❑ **Содержание имеет первостепенное значение.** Следует иметь в виду, что абоненты получают преимущество от частей содержания, расположенных на стороне сервера. Необходимо помнить, что представление этого содержания, кроме других факторов, ограничивается еще и возможностями самого языка WML. Нельзя полагаться на высокий уровень графики приложений и других приспособлений, занимающих весь интерес абонента.
- ❑ **Прежде всего, это телефоны.** Большинство пользователей хотят иметь маленькие телефоны и маленькие экраны. И хотят их именно для обеспечения голосовых услуг. Пользователи вряд ли станут покупать телефоны большего размера просто потому, что эти телефоны предоставляют более широкие возможности при работе с данными. Если приложение не работает на телефоне в соответствии с ожиданиями пользователя, тот вообще не будет пользоваться этим приложением.
- ❑ **Первичным вариантом использования будет не хождение по сети.** По многим причинам, абоненты вообще не бродят по сети. Обычно они используют браузер для доступа к небольшим разделам информации, расположенным на HTTP-сервере для того, чтобы воспользоваться продуктом или сервисом. Исследование целевых групп клиентов до запуска системы в эксплуатацию является обязательным условием успеха.
- ❑ **Беспроводные данные приходят не быстро.** Передача данных по каналам беспроводного Интернета выполняется со скоростью 14 400 бит в секунду. И, несмотря на то, что сайт WML относительно меньше сайта HTML, каждый байт нужно учитывать, так как абонент платит поминутно. Необходимо постоянно удалять бесполезное содержание с сервера и следить за эффективностью передачи данных.
- ❑ **Каждое нажатие клавиши прибавляет цену использования.** Процесс прокручивания может оказаться трудоемким. Ввод текста и навигация могут оказаться трудоемкими. Каждое из этих действий является результатом нажатия клавиш. Необходимо уменьшать количество нажимаемых клавиш в процессе работы приложения.
- ❑ **В беспроводных приложениях краткость обычно перевешивает точность и дружелюбие содержания.** Тот факт, что у вас имеется море информации, которое вы готовы предложить пользователям, еще не означает, что они

хотят получать все это на свои телефоны. То, что ваша компания имеет известный или дружелюбный имидж еще не означает, что этот стиль подходит к маленькому экрану. А если точность данных превышает шесть знаков после запятой, то совсем не обязательно, что такой метод округления подходит для представления на сотовом телефоне.

- **Маленьких телефонов будет больше чем больших.** Многие или большинство пользователей будут продолжать хотеть использовать маленькие телефоны, даже если большие будут предлагать лучшие браузеры. Ваше приложение должно работать и на маленьких телефонах.

Работа мобильного телефона

Прежде чем начать проектировать беспроводное приложение, необходимо понять условия его функционирования на мобильном терминале. Очевидно, эти условия не совпадают с аналогичными условиями для пользователя браузера на настольном компьютере. Что это означает для проектировщика дизайна? Здесь будут описаны наиболее важные с нашей точки зрения вопросы, начиная с пользователя, который будет использовать создаваемое приложение.

Пользователи

Абоненты сотовой связи в основном средние пользователи и явно не новаторы. Несмотря на то, что вы создаете приложение для специальной аудитории, лучше всего предположить, что пользователи не захотят отказываться от своих привычек и использовать плохо созданный сайт только потому, что он связан с высокими технологиями или уникальным сервисом. Сети сотовой связи используются потому, что телефоны доступны. Аналогично WML-сайт посещается пользователями, если он доступен.

Пользователи сотовой связи часто очень беспокоятся, когда они платят за эфирное время поминутно. Так как беспроводной Интернет является CSD, они будут также платить за минуту использования. Не делайте процессы обработки более длинными, чем они должны быть. Пользователи, которые почувствуют торможение работы, будут отказываться от использования такого сервиса просто потому, что это дороже, чем они ожидают.

Еще один взгляд на пользователей потребует разделить их на тех, кто знаком и тех, кто не знаком с вашим HTML-сайтом. Некоторые из них могут иметь представление о том, что это такое, в то время как другие — нет. WML-сайты, которые позволяют новым пользователям подписаться на что-то, с чем не предполагается проводить предварительного знакомства (набор свойств, структура меню, жаргон и т. д.), и что может быть непривлекательным. Даже если пользователи должны быть предварительно зарегистрированы для пользования WML-услугами, добавление раздела для работы с не-

опытными и незарегистрированными абонентами может дать несколько новых клиентов вашему сервису.

Палитра использования

Вопрос, который следует задать самому себе, как разработчику, звучит так: "Что будут делать мои пользователи, когда получат доступ к моему сайту?" Этот вопрос не звучит в такой постановке применительно к HTML-сайтам, потому что пользователи этих сайтов обычно уделяют 100% своего внимания сайтам. Человек пользуется услугами сотовой связи в различной окружающей обстановке. Он может вести автомобиль, прогуливаться, разговаривать с другим человеком, обедать и т. п. Он может положить свой телефон, потому что официант только что принес еду. Подумайте, каким образом ваше приложение могло бы стать более эффективным для занятого или невнимательного клиента. Также подумайте, где этот абонент может в данный момент находиться. Он может быть дома, в офисе, на фабрике или в вагоне поезда. Он может быть знаком с местом расположения или не очень. Он может иметь доступ к другим ресурсам или нет. Так что делайте ваш дизайн гибким и не создавайте слишком многого из того, что ограничивает пользование услугой, которую вы предлагаете.

Телефоны

Для большинства разработчиков ясно, что условия использования мобильного приложения отличаются от условий настольного приложения. Однако многие разработчики ошибочно предполагают, что мобильные телефоны имеют много общего с ручными или карманными компьютерами и делают свои приложения аналогичными.

Так как разработчики иногда вовлечены в проектирование сайтов для всего диапазона мобильных устройств, они часто применяют одинаковые стратегии к созданию телефонных приложений и к созданию приложений для карманных компьютеров. В действительности с точки зрения дизайнера ручные и карманные компьютеры гораздо ближе по среде использования к настольным компьютерам, чем мобильные телефоны. Несмотря на то, что некоторые мобильные телефоны имеют экраны и устройства ввода, такие же сложные, как и у карманных компьютеров, необходимо помнить, что WML-приложение должно работать для всех телефонов.

Замечание

Так как не рекомендуется доставлять WML-приложения для HDML-браузеров, описание этого раздела не включает HDML-браузеры. Если вы собираетесь использовать свои приложения WML для HDML-браузеров, изучайте руководства, направленные на такую разработку.

Следующие предположения кажутся осмысленными при работе с браузерами Phone.com версии 4.0 или выше.

□ **Клавиатура.** Числовая клавиатура предназначена для ввода текста и применения функциональных клавиш. Клавиши <2>—<9> используются соответственно обозначениям, в то время как остальные — в другой последовательности. Не нужно использовать приложения, в которых по умолчанию предполагается, что клавишам <1>, <0>, <*> и <#> назначены некоторые функции.

- Все телефоны поддерживают стандартную 12-кнопочную клавиатуру (<1>—<9>, <*>, <0> и <#>). Все телефоны назначают символы букв клавишам <2>—<9>.
- Символ пробела назначается на клавиши <1>, <0> или <#>.
- Все телефоны имеют символы, назначенные на клавишу <1> (например, нажмите <1> повторно для получения символов @ или \$). Некоторые телефоны имеют спецсимволы, назначенные на клавишу <0>. Обратите внимание, что набор символов, назначенных на эти клавиши, и порядок их ввода отличаются.
- Некоторые телефоны используют клавиши <*> и <#> для навигации вверх/вниз.
- Некоторые телефоны используют клавиши <*> и <#> для движения курсора влево и вправо.
- Клавиша <*> иногда используется, как клавиша <Shift> (переключение регистра).
- Клавиша <#> часто используется для продвижения курсора в правую позицию.
- Телефоны, которые поддерживают Tegic T9 или Motorola Lexicus, используют клавиши <*>, <0> и <#> для управления упреждающим вводом.

□ **Экран.** В этом разделе приведены минимальные характеристики экранов браузеров.

- Три или более строк для показа содержания карты (плюс строка программируемых клавиш).
- Большинство телефонов поддерживают графику.
- Черно-белый экран. Некоторые телефоны поддерживают цвет и градации серого, но браузеры не разрешают управлять этими возможностями.
- 14 или более символов в текстовой строке. Для элементов списка число доступных символов уменьшается до 12.

- **Программируемые клавиши.** Существуют минимальные требования к программируемым функциональным клавишам.
 - Должно быть минимум две клавиши.
 - Главная клавиша (<ассерт>) используется для выбора. Вторичная клавиша (<options>) всегда доступна и может быть более трудна для использования.
 - Шесть или более символов для каждой клавиши указаны в качестве меток.

Обычно телефоны используют следующие методы доступа и активации программируемых клавиш.

- **Специально выделенные клавиши.** Две метки показаны на экране, каждая из которых имеет выделенную клавишу для активации.

- **Переназначенные клавиши.** Две метки клавиш, показанные на экране, могут обслуживать другие функции в период неактивности браузера. Все телефоны используют различные методы активации программируемых клавиш.

- **<BACK>.** Данная функция всегда отображается на жесткую клавишу или зарезервированную на телефоне функциональную клавишу. Наиболее общее решение заключается в назначении для этих целей клавиши <CLR>, которая также используется для удаления текста в поле ввода. Для того чтобы избежать недоразумений, следует использовать непересекающиеся клавиши.

- Не используйте <BACK> в качестве программируемой клавиши <HOME>. Эта клавиша будет любой программируемой или аппаратной реализацией, однако ее трудно найти на некоторых видах телефонов. Аппаратно реализованная кнопка для <HOME> недоступна в большинстве существующих браузеров и маловероятно, что будет использоваться в ближайшем будущем. Обратите внимание, что клавиша <HOME> будет возвращать абонента в главное меню системы, а не на главную страницу приложения.
- Не ссылайтесь на первичную карту приложения как на домашнюю страничку. Рекомендуются использовать слово "Main" для ссылки на главную страничку приложения. Если имя сайта достаточно коротко или может быть сделано таковым, то лучше отсылать пользователя на главную страничку.

- **<MENU>.** Эта клавиша существует на большинстве телефонов, но в различных формах и назначается для выполнения различных функций. Для нее нет типичного исполнения. Поэтому дизайн не должен использовать назначение на клавишу <MENU>, т. к. она является разрешенной программируемой клавишей и подходит для обработки случаев, когда нужны больше, чем две клавиши.

Замечание

Некоторые телефоны имеют метку "MENU" на клавише, связанной со вторичной клавишей, поэтому <MENU> должно работать как вторичная клавиша, а не как первичная. Не нужно решать эту проблему путем изменения метки, заменяя <MENU> на аналогичное слово. Более подходящим вариантом является вызов меню и включение в него нового элемента.

- ❑ **<Up>/<Down>.** Все телефоны имеют клавиши для навигации вверх и вниз. Некоторые телефоны будут иметь прокручивание странички, которое обычно отображается на кнопке управления мощностью звука.
- ❑ **Закладки.** Все телефоны поддерживают закладки.
 - Закладки хранятся на сервере, а не на устройствах. Таким образом, память для закладок виртуально неограничена.
 - Большинство пользователей не будет знать, как использовать закладки, однако эксперты будут ими широко пользоваться. Сайт с полной возможностью закладок будет привлекать к себе регулярных абонентов.
 - Доступ к закладке легче, чем ее создание. Для того чтобы создать закладку, абонент должен получить доступ к меню браузера в телефоне. Не нужно полагаться на способность пользователя самостоятельно создавать закладки.
- ❑ **Текстовый экран.**
 - НЕ предполагайте фиксированную ширину шрифта. Некоторые телефоны имеют пропорциональный шрифт. Это важно для создания таблиц.
 - Все телефоны поддерживают ссылки, но иногда их трудно понять. Большинство телефонов помещают скобки вокруг ссылок (вместо более традиционно используемого подчеркивания). Если ссылка занимает несколько строк, ближайшие скобки будет трудно найти.
 - Все телефоны поддерживают как тег <wrap>, так и <nowrap>, но могут показывать их по-разному.
- ❑ **Текстовый ввод.**
 - Все телефоны имеют способность вводить маленькие символы (alpha), большие символы (ALPHA), числа (NUM) и спецсимволы (SYM).
 - Все телефоны поддерживают ввод текста, при котором нажатие на одну клавишу разное количество раз приводит к вводу разных символов (например, нажатие <5> дважды вводит "K"). Этот механизм ввода символов может слегка отличаться у разных типов телефонов.
 - Некоторые телефоны поддерживают современные методы ввода текста, например, Tegic T9 или Motorola Lexicus. Несмотря на то, что все население вашей целевой аудитории будет иметь телефоны, поддерживаю-

щие такой режим ввода, разработчик тем не менее не должен предполагать наличия такой возможности у всех пользователей сервиса.

- Некоторые телефоны могут иметь возможности ввода рукописного текста, виртуальную или реальную клавиатуру.
- Метод управления капитализацией символов отличается на разных беспроводных устройствах. Некоторые телефоны имеют клавишу модификации, которая работает, как программируемая клавиша, другие требуют повторного нажатия на клавишу для получения доступа к первому символу в нижнем или верхнем регистре. В будущем могут появиться другие методы управления капитализацией. К сожалению, некоторые из этих методов скрыты и интуитивно непонятны пользователю, поэтому они не будут знать, как ввести символ в верхнем регистре. Если ваш сайт требует чувствительного к регистру ввода (например, в пароле), то на этот аспект требуется обратить особое внимание.

❑ Подсказки.

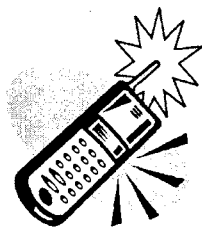
- Все телефоны поддерживают подсказки (Alerts), но они не должны приводить к недоразумениям при совместном использовании услуги SMS.
- Низкоприоритетные подсказки не дают уведомления пользователю.
- Подсказки среднего приоритета показывают только визуальный индикатор в виде иконки конверта.
- Высоко приоритетная подсказка показывает визуальный индикатор и звуковой сигнал.
- Срочная подсказка показывает визуальный индикатор, звуковой сигнал и сообщение поверх экрана.

❑ **Cookies** (данные о посещении, которые хранятся в браузере). Шлюзы операторского класса поддерживают cookies, которые хранятся на сервере-шлюзе. Телефоны не хранят cookies локально.

❑ **Несколько типов звуков для звонков.** Не все телефоны поддерживают несколько вариантов звуковых установок.

Глава 2

Справочник по WML



Материал данной главы состоит из справочника по языку WML версии 1.3 и описывает главные элементы, их атрибуты и расширения, реализованные в продуктах основных производителей.

Стандарт быстро развивается, поэтому везде, где была такая возможность, автор указал особенности самой последней версии стандарта WML 1.3 и процедуры перехода к следующей версии языка.

Замечание

Тексты примеров в приложении построены таким образом, чтобы их можно было просматривать даже на устройствах, не поддерживающих национальные кодировки. Это также избавляет от необходимости включать модули перекодировки содержания на демонстрационном сервере. Большинство текстов примеров доступно на сервере компании Peter Service Ltd. по адресу: <http://wap.billing.ru/>.

Тег `<a>`

Обязателен в стандарте WML 1.3.

Это короткий синтаксис от тега `<anchor>`, который может быть использован только для определения `<go>`-задач, требующих явного указания URL (Uniform Resource Locator, унифицированный указатель информационного ресурса).

Синтаксис

```
<a href="url"
    title="label"
    accesskey="key">
content
</a>
```

где `content` — любая корректная информация в виде набора элементов `<text>`, `` и `
`.

Атрибуты

□ `href` — URL гиперссылки.

- ❑ `title` — имя гиперссылки, которое может быть представлено WML-агентом различными способами в зависимости от реализации.
- ❑ `accesskey` — число от 0 до 9, которое появляется с левой стороны экрана перед ссылкой. Если пользователь нажимает соответствующую клавишу на терминале, последний будет выполнять задачу, назначенную ссылкой.

Тег `<access>`

Обязателен в стандарте WML 1.3.

Элемент управления доступом к информации WML-колоды. Вы должны описать этот элемент внутри заголовка колоды вместе с метаинформацией (см. элементы `<head>` и `<menu>`). Каждая колода может иметь только один элемент `<access>`. Все колоды являются общедоступными по умолчанию.

Синтаксис

```
<head>
  <access domain="domain" path="path">
  ...
</head>
```

Атрибуты

- ❑ `domain` — URL домена других колод, которые могут иметь доступ к картам данной колоды. По умолчанию это домен текущей колоды.
- ❑ `path` — корневой URL других колод, которые могут иметь доступ к данной колоде. Значение по умолчанию `"/` (корневой элемент данной колоды), которое позволяет любой колоде внутри данного домена иметь доступ к данной колоде.

Особенности

Если `<access domain="billing.ru"/>` описывает элемент `access`, то ссылка <http://www.billing.ru/> будет корректной, а <http://www.ling.ru/> или <http://www.billing.net/> будет неверной.

Если `<access path="/russian"/>` описывает элемент `access`, `/russian/index.wml` будет работать корректно, а путь `/russian-wml` — нет.

Тег `<anchor>`

Обязателен в стандарте WML 1.3.

Элемент связывает задачу (*task*) со строкой форматированного текста, часто называемого ссылкой (*link*). Вы можете описать ссылку внутри любого форматированного текста или элемента изображения.

Синтаксис

```
<anchor title="label"  
        accesskey="key">  
task text  
</anchor>
```

где *task* представляет действие для выполнения, когда пользователь активирует ссылку, а *text* является текстом устройства, которое будет показывать связь.

В качестве *task* вы можете указать одно из следующих действий:

- ☐ `<go>`
- ☐ `<prev>`
- ☐ `<refresh>`

Значение элемента *text* обычно равно OFF.

Атрибуты

- ☐ *title* — метка, которая идентифицирует ссылку. Если вы не описали этот атрибут, устройство использует слово `link` в качестве метки по умолчанию.
- ☐ *accesskey* — число от 0 до 9, которое появляется с левой стороны экрана перед ссылкой. Если пользователь нажимает соответствующую клавишу на терминале, он выполняет задачу, назначенную ссылкой.

Тег ****

Необязателен в стандарте WML 1.3.

Этот необязательный элемент стандарта позволяет для некоторых типов браузеров определять жирный шрифт текста.

Синтаксис

```
<b> text </b>
```

где *text* — это текст, который будет показан жирным шрифтом.

Тег **<big>**

Необязателен в стандарте WML 1.3.

Этот необязательный элемент позволяет разработчику определить увеличенный шрифт текста в некоторых типах браузеров.

Синтаксис

```
<big> text </big>
```

где *text* — это текст, который будет показан увеличенным шрифтом.

Тег **
**

Обязателен в стандарте WML 1.3.

Этот элемент определяет разрыв строки. Например, его применение приведет к тому, что устройство будет показывать последующий текст или графический элемент на следующей строке.

Синтаксис

```
text <br/>
```

где *text* — это текст, после которого будет произведен разрыв строки.

В отличие от HTML, все элементы языка WML требуют завершения символом конца элемента (/). Таким образом, для того чтобы вставить разрыв строки в WML, нужно писать `
`, а не `
`.

Замечание по работе UP.Browser

При задании элементов `<input>` или `<select>` браузер показывает текст, который был задан в качестве подсказки для пользователя. Если на одной карте определено несколько таких элементов, вставка тега `
` внутри подсказки приводит к некорректному выводу, поскольку браузер будет использовать его для разбиения информации ваших полей на несколько экранов. В результате только последняя порция подсказок станет появляться на том же самом экране, что и поле для ввода или список выбора.

Тег **<card>**

Обязателен в стандарте WML 1.3.

WML-колода состоит из одного или нескольких `<card>`-элементов, каждый из которых осуществляет отдельное взаимодействие между пользователем и беспроводным устройством (терминалом). Терминал показывает лишь одну карту одновременно. Однако в некоторых случаях одна карта может появляться как серия экранов (см. также описания элементов `<template>` и `<wml>`).

Синтаксис

```
<wml>
```

```
  <card id="name"
        title="label"
        newcontext="boolean"
```

```
        ordered="true"
        onenterforward="url"
        onenterbackward="url"
        ontimer="url">
content
    </card>
</wml>
```

где *content* представляет WML-карту и состоит из одного или нескольких следующих элементов:

- ☐ `<onevent>`
- ☐ `<timer>`
- ☐ `<do>`
- ☐ `<a>`
- ☐ `<fieldset>`
- ☐ ``
- ☐ `<input>`
- ☐ `<select>`
- ☐ `<p>`

За исключением вышеназванных элементов, устройство показывает элементы в том порядке, в котором вы их написали.

Замечание

Элементы внутри описания карты должны располагаться в следующем порядке: `<onevent>`, `<timer>`, `<do>`.

Атрибуты

- ☐ `id` — описывает имя карты. Имя действует как локальная ссылка для навигации по набору карт. Например, для того чтобы достичь карты с указанным именем, вы можете описать `<go href="#cardname"/>`, чтобы достичь карты с указанным именем.
- ☐ `title` — описывает краткую метку карты. UP.Browser использует метку как имя закладки, по умолчанию присваиваемую карте, когда пользователь помечает карту закладкой *bookmark*. Другие микробраузеры могут использовать ее для иных целей, например, для выпадающего меню подсказок. UP.Browser не показывает заголовок как часть содержимого карты. Производитель телефонов может дополнительно отображать заголовок, если он считает, что в устройстве достаточно для этого места.

- ❑ `newcontext` — принимает значение `true` или `false`. Описывает, должно ли устройство инициализировать содержание, когда пользователь выполняет навигацию по задачам, оформленным в виде `<go>`-элементов. Если вы описали `newcontext="true"`, то удаляются все контекстно-зависимые переменные, очищается стек историй и сбрасывается состояние устройства к заранее определенному значению.
- ❑ `ordered` — принимает значение `true` или `false`. Описывает организацию содержимого карты. Значение по умолчанию приводит к тому, что устройство отображает содержание в фиксированной последовательности. В противном случае пользователь может выбирать порядок визуализации и прохождения элементов в карте.
- ❑ `onenterforward` — описывает URL, который открывается при навигации пользователем серии элементов `<go>`.
- ❑ `onenterbackward` — выбор `<prev>` приводит к активизации внутреннего события.
- ❑ `ontimer` — если установлен, то по истечении указанного времени активируется событие.

Особенности

Атрибут `ordered` позволяет описать способ, которым устройство будет показывать несколько элементов `content` внутри одной карты. Значение `true` показывает эти элементы в линейной последовательности (по умолчанию). Используйте данную установку для коротких форм, содержащих требуемые поля. Если устройство не может показывать все элементы на одном экране, оно разбивает их на несколько экранов, основываясь на:

- ❑ группировке полей, описанных в одном или более элементах `<fieldset>`;
- ❑ индивидуальных полях (с предшествующим пояснительным текстом, используемым в качестве подсказки).

Тег `<catch>`

Расширение стандарта от компании Openwave.

Этот элемент описывает обработчик нестандартных ситуаций, которые могут возникнуть в процессе выполнения задачи.

Внутреннее событие возникает в момент фиксации нестандартной ситуации и может быть обработано посредством атрибута `onthrow` или вложением `<onevent>` внутрь элемента `<catch>`.

Элемент не может содержать более одного элемента `<catch>` для одного имени.

Синтаксис

```
<catch name="#error#1"
      onthrow="/displayError">
  <receive name="msg"/>
</catch>
```

Атрибуты

- ☐ `name` — описывает имя исключения. Если атрибут имени пропущен, элемент `<catch>` будет обрабатывать любое исключение.
- ☐ `onthrow` — событие возникает только в том случае, когда исключение совпадает со специфицированным элементом.

Тег `<do>`

Обязателен в стандарте WML 1.3.

Этот элемент связывает задачи с элементом внутри пользовательского интерфейса.

Синтаксис

```
<do type="type"
    label="label"
    name="name"
    optional="boolean">
task
</do>
```

где *task* представляет действие, которое должно выполняться, когда пользователь активизирует предписанный интерфейсом механизм.

В качестве *task* должна быть одна из следующих конструкций:

- ☐ `<go>`
- ☐ `<prev>`
- ☐ `<noop>`
- ☐ `<refresh>`

Если телефон поддерживает графические объекты, атрибуты `localsrc`, `src` и `alt` используются именно в этом порядке. Если телефон не поддерживает графических объектов, атрибут `alt` описывает метку. Если ``-элемент описан, атрибут `label` элемента `<do>` игнорируется.

Если ``-элемент не описан, атрибут `label` элемента `<do>` используется.

Замечание по работе UP.Browser

Браузер UP.Browser использует в своей работе расширение стандарта, которое позволяет применять спецификации графических объектов, как метки внутренних элементов `<do>`. Эта возможность осуществляется вложением элемента `` внутрь предложения `<do>`. Например:

```
<do type="accept">
  <go href="/foo"/>
  
</do>
```

Атрибуты

- ❑ `type` — идентифицирует механизм общего пользовательского интерфейса, который переключает описанный в `<do>` элемент `<task>` (см. описание ниже).
- ❑ `label` — метка, которая идентифицирует задачу с механизмом пользовательского интерфейса. Например, если вам нужна задача, связанная ключом `accept`, устройство показывает это значение в качестве метки функциональной клавиши. Если вы не описали атрибут `label`, устройство использует слово "ОК" в качестве метки `accept`-ключа по умолчанию. Для того чтобы обеспечить совместимость с широким набором устройств, `label` должна быть максимум из пяти символов. Устройства игнорируют атрибут `label`, если они не поддерживают динамической разметки. Программное обеспечение браузера UP.Browser текущей версии не поддерживает этот атрибут для следующих значений `type`:
 - `type="delete"`;
 - `type="help"`;
 - `type="prev"`.
- ❑ `name` — описывает имя элемента `<do>`. Если имена элементов `<do>` уровня карты и уровня колоды совпадают, то используется следующее правило разрешения конфликта. Если содержание элемента `<do>` уровня карты имеет то же самое имя, что и содержание элемента `<do>` уровня колоды (например, определен внутри элемента `<template>`), определение, заданное для карты, превалирует над определением уровня колоды.
- ❑ `optional` — имеет значение `true` или `false` (по умолчанию). Позволяет устройству игнорировать этот элемент.

Вы можете описать значения атрибута `type`, указанные в табл. 2.1 (все типы являются зарезервированными словами за исключением отдельных случаев).

Таблица 2.1. Значения атрибута *type*

Значение <i>type</i>	Выполняет задачу, если пользователь:
accept	вызывает механизм accept (функциональный ключ, клавиша и т. д.)
delete	вызывает механизм delete (функциональный ключ, клавиша и т. д.)
help	вызывает механизм help (может быть контекстно-чувствительным)
options	вызывает механизм options (функциональный ключ, клавиша и т. д.)
prev	вызывает карту посредством активизации механизма prev из другой карты
reset	вызывает механизм reset (очищает и сбрасывает текущий статус устройства). UP.Browser в текущей версии не поддерживает это значение атрибута
unknown	вызывает механизм Unknown (эквивалентно <code>type=""</code>). UP.Browser в текущей версии не поддерживает это значение атрибута
x-*, x-*	UP.Browser в текущей версии не поддерживает это значение атрибута. Зарезервировано для будущих применений
vnd.co-type	вызывает механизм, специфицированный поставщиком устройства, где <i>co</i> описывает поставщика, а <i>type</i> — действие (не специфицировано). UP.Browser в текущей версии не поддерживает это значение атрибута

Ни одно из описанных значений атрибута *type* не дает конкретного механизма пользовательского интерфейса. Некоторые устройства отображают каждый тип на физическую клавишу, в то время как другие связывают тип с действием, зависящим от контекста (например, нажатие или нажатие и удержание). Таким образом, при проектировании вашего пользовательского интерфейса помните, что вы не можете описать (или даже предположить), какой конкретно механизм будет использован в этом устройстве.

Замечание

Если вы определяете несколько элементов `<do>` одинакового типа в одной карте, вам следует описать атрибут `name` для каждого элемента `<do>`, чтобы уникальным образом определить сущность каждого элемента одного и того же типа. При определении нескольких элементов `<do>` ключ `options` помечается, как `menu`, чтобы обеспечить меню, которое включает все элементы `<do>` с действиями `accept` или `options`.

Тег ****

Необязателен в стандарте WML 1.3.

Этот элемент описывает выделенный текст.

Синтаксис

```
<em> text </em>
```

где `text` обозначает выделенный текст.

UP.Browser в текущей версии не поддерживает этот элемент.

Тег **<exit>**

Расширение стандарта от компании Openwave.

Этот элемент декларирует завершение задачи, указывая, что текущее содержание должно быть уничтожено. Значения могут посылааться в вызывающую страницу содержания с вложением элемента `<send>`. Элемент `<exit>` приводит к тому, что текущее содержание разрушается, включая все переменные и историю состояния, которая хранится в `context`.

Синтаксис

```
<exit>  
  <send value="393"/>  
  <send value="$X"/>  
</exit>
```

Этот элемент отсутствует в следующей версии стандарта WML и приведен в справочнике, так как он используется в продуктах компании Phone.com и поддерживается самым распространенным в настоящее время браузером UP.Browser.

Тег **<fieldset>**

Необязателен в стандарте WML 1.3.

Этот элемент позволяет группировать несколько текстовых строк или элементов ввода внутри одной карты. Указывая один или несколько элементов `<fieldset>`, вы можете контролировать, как устройство представляет содержание карты для того, чтобы упростить навигацию по нему.

Синтаксис

```
<fieldset title="label">  
content  
</fieldset>
```

где *content* представляет элементы, которые следует сгруппировать вместе для показа на экране устройства.

Это может быть один или несколько следующих элементов:

- ☐ `<fieldset>` (вложенный `<fieldset>`)
- ☐ `<input>`
- ☐ `<select>`
- ☐ `text`

Вы можете также описать текст в группе `<fieldset>`. Устройство использует этот текст для различных целей в зависимости от описанных элементов (например, для того, чтобы подсказать пользователю варианты ввода).

Устройства показывают эти элементы в той последовательности, в которой вы их описываете.

Атрибуты

Атрибут `title` описывает краткую метку группы `<fieldset>`. Некоторые устройства используют эту метку как заголовок при показе `<fieldset>`-содержания. Другие могут применять ее как метку для механизма пользовательского интерфейса, который разрешит пользователю выполнять навигацию по содержанию этой группы. Если устройство не может показать все содержимое карты на одном экране и `ordered=true` (см. атрибут `ordered`), то UP.Browser использует атрибут `title` для указания группы элементов в меню общего уровня.

Тег `<go>`

Обязателен в стандарте WML 1.3.

Тег является элементом `task` инструкции `<do>`. Этот элемент заставляет устройство открыть указанный URL. Если данный URL описывает конкретную карту, устройство показывает эту карту. Если URL указывает на конкретную колоду, устройство будет показывать первую карту из колоды.

Синтаксис

```
<go href="url"
    sendreferer="boolean"
    method="method"
    accept-charset="charset">
  <postfield name="data" value="value"/>
  content
</go>
```

где *content* представляет переменные, которые должны быть установлены при открытии URL. С помощью *content* вы можете описать одну или более переменных в элементе `<go>`, используя конструкцию `<setvar>`.

Замечание

В отличие от других конструкций языка, имеющих содержание, описание содержания в элементе `<go>` является дополнительной возможностью. Если вы не описываете содержание в этом элементе, следует использовать синтаксис `<go attributes/>`, а не `<go attributes> content </go>`.

Атрибуты

- ❑ `href` — URL, который будет открываться.
- ❑ `sendreferer` — имеет значение `true` или `false` (по умолчанию). Описывает, следует ли устройству включать URL-колоды в URL-запрос. Указание `true` приведет к тому, что устройство установит заголовок `HTTP_REFERER` на относительный URL запрашиваемой колоды. Если вы хотите ограничить доступ к защищенным сервисам, колоды которых запрашивают конкретный URL, то должны устанавливать эту опцию в значение `true`.
- ❑ `method` — имеет значение `GET` (по умолчанию) или `POST`. Описывает метод подстановки для HTTP-протокола. Указание `"post"` приведет к тому, что WAP-шлюз будет декодировать данные, представляющие значения переменных в набор символов, описанных HTTP-заголовком. Следует предусмотреть это преобразование, если устройство передает не ASCII-набор символов (например, UTF-8). Информация об используемых кодировках и HTTP-заголовках может быть найдена, например, в руководстве по разработке компании Phone.com. Если вы не описываете этот атрибут, а элемент `<do>` указывает вложенный элемент `<postfield>`, устройство автоматически использует метод `"post"`.
- ❑ `accept-charset` — указывает процедуру кодирования, которую ваше приложение будет принимать во внимание при выполнении. Устройство использует этот атрибут, чтобы преобразовывать данные, указанные в элементе `<postfield>`.
 - Обычно WAP-шлюз предполагает кодировку UTF-8 как кодировку, установленную по умолчанию (в которой US ASCII является подмножеством), таким образом, WML-услуги в США, Канаде или Австралии не требуют использования этого атрибута.
 - Вы можете также опустить этот атрибут, если указали вашу кодировку в заголовке HTTP-запроса. Заметим, что атрибут `accept-charset` элемента `<go>` имеет более высокий приоритет, чем аналогичные указания, представленные в заголовке HTTP.

- Этот атрибут допускает указание нескольких таблиц кодировки, перечисленных через запятую или пробел. Все таблицы кодировки должны быть из списка IANA. Например:
`accept-charset="utf-8,US-ascii,ISO-8859-1"`
- Полный список поддерживаемых WAP-шлюзами имен Phone.com вы можете найти в руководстве по разработке этой компании. Полный список IANA-таблиц кодировки доступен по адресу <http://www.iana.org/>.

Тег **<head>**

Обязателен в стандарте WML 1.3.

Этот элемент описывает информацию о колоде как едином целом, включая метаданные и информацию управления доступом.

Синтаксис

```
<head> content </head>
```

где *content* представляет информацию заголовка уровня колоды.

Тег **<i>**

Необязателен в стандарте WML 1.3.

Этот необязательный элемент позволяет отобразить курсивом текст в некоторых типах браузеров.

Синтаксис

```
<i> text </i>
```

где *text* будет показан курсивом. Например:

```
<wml>  
  <card>  
    <p>  
      <i>Telemix</i>, <i>LEDA</i>, and <i>  
      ULTRA STAR</i> are DEALERS.  
    </p>  
  </card>  
</wml>
```

Тег ****

Обязателен в стандарте WML 1.3.

Элемент `` указывает беспроводному терминалу на необходимость показать некоторый графический объект вместе с форматированным текстом.

В настоящее время не все типы браузеров позволяют показывать изображения, однако эта возможность является обычным свойством любого браузера последующих стандартов. Некоторые типы браузеров расширяют функциональность элементов `<do>` и `<option>`, позволяя вставлять графику.

Синтаксис

```

```

Атрибуты

- ❑ `alt` — определяет поведение устройства в случае, если оно не поддерживает графику или не может найти предписанный для показа графический объект. Не обязателен.
- ❑ `src` — задает URL графического объекта, который должен быть показан. Если задан корректный значок в атрибуте `localsrc`, то устройство должно игнорировать этот атрибут.
- ❑ `localsrc` — имя известного значка. Если устройство не в состоянии найти значок в ROM-памяти (Read Only Memory, память только для чтения), оно должно попытаться извлечь ее с сервера WML-содержания. Если значок описан корректно (ниже представлен полный список имен значков), устройство будет игнорировать атрибуты `src` и `alt`, даже если они являются обязательными.

Не поддерживается в Nokia 7110.

- ❑ `align` — имеет значения `top`, `middle`, `bottom`. Описывает относительное выравнивание в текущей строке текста. Не обязателен в стандарте 1.3.
- ❑ `height` — UP.Browser не поддерживает этот атрибут. Не обязателен в стандарте 1.3.
- ❑ `width` — UP.Browser не поддерживает этот атрибут. Не обязателен в стандарте 1.3.
- ❑ `vspace` — UP.Browser не поддерживает этот атрибут. Не обязателен в стандарте 1.3.
- ❑ `hspace` — описывает пространство слева и справа от графического объекта. По умолчанию равно 0. Не обязателен в стандарте 1.3.

Замечание

Некоторые значки имеют другую форму при показе из-за использования разных шрифтов. Поскольку нет возможности управления размером шрифта, на разных устройствах значки могут немного отличаться по внешнему виду.

Список разрешенных имен встроенных значков:

1 exclamation1	34 downtri2	67 calendar	100 book2
2 exclamation2	35 uptri2	68 smileyface	101 book3
3 question1	36 bigdiamond1	69 star2	102 book4
4 question2	37 bigdiamond2	70 rightarrow2	103 document2
5 lefttri1	38 biggestsquare1	71 leftarrow2	104 scissors
6 righttri1	39 biggestsquare2	72 gem	105 day
7 lefttri2	40 bigcircle1	73 checkmark1	106 ticket
8 righttri2	41 bigcircle2	74 dog	107 cloud
9 littlesquare1	42 uparrow2	75 star3	108 envelope1
10 littlesquare2	43 downarrow2	76 sparkle	109 check
11 i symbol	44 sun	77 lightbulb	110 videocam
12 wineglass	45 baseball	78 bird	111 camcorder
13 speaker	46 clock	79 folder1	112 house
14 dollarsign	47 moon2	80 head1	113 flower
15 moon1	48 bell	81 copyright	114 knife
16 bolt	49 pushpin	82 registered	115 videotape
17 medsquare1	50 smallface	83 briefcase	116 glasses
18 medsquare2	51 heart	84 folder2	117 roundarrow1
19 littlediamond1	52 martini	85 phone1	118 roundarrow2
20 littlediamond2	53 bud	86 voiceballoon	119 magnifyglass
21 bigsquare1	54 trademark	87 creditcard	120 key
22 bigsquare2	55 multiply	88 uptri3	121 notel
23 littlecircle1	56 document1	89 downtri3	122 note2
24 littlecircle2	57 hourglass1	90 usa	123 boltnut
25 wristwatch	58 hourglass2	91 note3	124 shoe
26 plus	59 floppy1	92 clipboard	125 car
27 minus	60 snowflake	93 cup	126 floppy2
28 star1	61 cross1	94 camera1	127 chart
29 uparrow1	62 cross2	95 rain	128 graph1
30 downarrow1	63 rightarrow1	96 football	129 mailbox
31 circleslash	64 leftarrow1	97 book1	130 flashlight
32 downtri1	65 mug	98 stopsign	131 rolocard
33 uptri1	66 divide	99 trafficlight	132 check2

133 leaf	144 present	155 phone2	166 fax
134 hound	145 tag	156 factory	167 partcloudy
135 battery	146 meal1	157 ruler1	168 plane
136 scroll	147 books	158 ruler2	169 boat
137 thumbtack	148 truck	159 graph2	170 dice
138 lockkey	149 pencil	160 meal2	171 newspaper
139 dollar	150 uplogo	161 phone3	172 train
140 lefthand	151 envelope2	162 plug	173 blankfull
141 righthand	152 wrench	163 family	174 blankhalf
142 tablet	153 outbox	164 link	175 blankquarter
143 paperclip	154 inbox	165 package	

Тег `<input>`

Обязателен в стандарте WML 1.3.

Использование этого элемента разрешает пользователю вводить текст, который в дальнейшем устройство присваивает в качестве значения внутренней переменной приложения.

Синтаксис

```
<input name="variable"
      title="label"
      type="type"
      value="value"
      format="specifier"
      emptyok="boolean"
      size="n"
      maxlength="n"
      tabindex="n"/>
```

где `value` представляет текст или графический объект, которые устройство будет использовать в качестве подсказки для пользователя.

Атрибуты

- ❑ `name` — указывает имя переменной, в которой устройство сохраняет введенный пользователем текст. Если переменная не имеет значения, устройство должно установить его по умолчанию. Если значение по умолчанию также не установлено, устройство должно инициализировать переменную с пустой строкой ("").
- ❑ `title` — указывает краткую метку для вводимого элемента. Некоторые устройства используют эту метку в качестве подсказки при показе поля

ввода. Другие могут применять его как метку механизма пользовательского интерфейса, которая разрешает пользователю выполнять навигацию по элементам. Например, если устройство не в состоянии показать все содержание карты на одном экране, а атрибут `ordered` установлен в значение `true`, UP.Browser использует атрибут `title` для определения этого вводимого элемента в меню уровня `summary`.

- ❑ `type` — описывает, как устройство должно показывать текст, введенный пользователем. Указывая значение `"text"`, можно сделать текст видимым. Указание значения `"password"` приводит к маскировке текста (замена на символ `*`). Заметим, что этот режим не приводит к шифрованию паролем, поэтому вам не следует полагаться на него как на защищенный режим передачи секретных данных.
- ❑ `value` — этот атрибут указывает значение, которое будет присваиваться при вводе по умолчанию.
- ❑ `accesskey` — число от 0 до 9, которое появляется с левой стороны экрана перед ссылкой. При нажатии соответствующей клавиши на клавиатуре терминала, устройство выполняет задачу, определенную в ссылке. Рекомендуется нумеровать ссылки в том порядке, в котором они появляются.
- ❑ `format` — описывает формат данных, которым вводимые пользователем данные должны соответствовать. Если атрибут опущен, устройство предполагает `"*м"` (по умолчанию первый символ с большой буквы и следом строка символов максимально разрешенной длины, состоящая из чисел и символов произвольного регистра).
- ❑ `emptyok` — описывает, может ли пользователь оставить поле пустым при вводе. Значение `true` указывает на то, что поле необязательно для заполнения, однако в случае ввода в этом поле данных, к ним предъявляются все требования формата, о которых говорилось при описании атрибута формата.
- ❑ `maxlength` — описывает максимально разрешенную длину строки вводимых символов. Если он не описан, UP.Browser предполагает длину в 256 символов.
- ❑ `tabindex` — UP.Browser не поддерживает этот атрибут.
- ❑ `size` — UP.Browser не поддерживает этот атрибут.

Особенности атрибута *format*

Атрибут `format` может иметь следующие значения.

- ❑ Для того чтобы ограничить количество вводимых пользователем символов, следует описать одиночную цифру перед тегом символа. Например, `format="3X"` разрешает пользователю вводить максимум три символьных, цифровых или буквенных знака в верхнем регистре знака.

- ❑ Для ввода строк неограниченной длины нужно использовать "*", например, `format="*a"`.

В значении атрибута `format` могут использоваться следующие метасимволы:

- ❑ `A` — любой алфавитно-цифровой символ в верхнем регистре `A—Z` или знак пунктуации;
- ❑ `a` — любой алфавитно-цифровой символ в нижнем регистре `a—z` или знак пунктуации;
- ❑ `N` — любая цифра;
- ❑ `x` — любой алфавитно-цифровой символ в верхнем регистре `A—Z`, цифры `0—9` или знак пунктуации;
- ❑ `x` — любой алфавитно-цифровой символ в верхнем регистре `a—z`, цифры `0—9` или знак пунктуации;
- ❑ `m` — любой символ;
- ❑ `m` — любой символ, по умолчанию первый символ в нижнем регистре.

Тег `<link>`

Расширение стандарта от компании Openwave.

Этот элемент языка задает отношение между текущей колодой и другим документом. Элемент не имеет содержания и должен существовать внутри элемента `<head>`.

Синтаксис

```
<wml>
  <head>
    <link href="/next"
          rel="next"/>
    <link href="/bar"
          rel="NEXT"/>
  </head>
  . . .
</wml>
```

Атрибуты

- ❑ `href` — описывает позицию документа, на который выполняется ссылка.
- ❑ `rel` — описывает отношение между текущим и документом, на который выполняется ссылка.
- ❑ `sendreferer` — имеет значение `true` или `false`. Указывает устройству, нужно ли включать URL колоды в URL запроса. Конструкция `sendreferer="true"`

заставляет устройство устанавливать HTTP_REFERER-заголовок на относительный URL запрашиваемой колоды. Если существует потребность ограничить доступ к защищенным услугам, устройство должно поддерживать этот параметр.

Тег <meta>

Необязателен в стандарте WML 1.3.

Необязательный элемент <meta> обеспечивает метаинформацию для колоды. Элемент описывается внутри заголовка колоды вместе с информацией об ограничении доступа. В настоящее время не все устройства поддерживают любые типы метаинформации.

Синтаксис

```
<head>
  <meta http-equiv="Cache-Control"
        content="max-age=time"
        forua=true/>
```

...

```
</head>
```

Атрибуты

- ❑ **property** — должен быть описан один из следующих атрибутов:
 - **name="name"**. Если атрибут **name** описан, WAP-шлюз игнорирует метаданные;
 - **http-equiv="name"**. Если атрибут **http-equiv** описан, WAP-шлюз преобразует метаданные в HTTP-заголовок.
- ❑ **content** — описывает значение метаданных, связанных с атрибутом **property**.
- ❑ **scheme** — UP.Browser не поддерживает этот атрибут.
- ❑ **forua** — имеет значение **true** или **false**. Указывает, что использует атрибут **property** для управления устройством пользователя.

Если **forua="false"**, промежуточный агент должен удалить <meta>-элемент прежде, чем отправить документ клиенту. При указании **forua="true"** метаданные должны передаваться клиенту. Метод доставки может изменяться. Например, метаданные **http-equiv** могут быть доставлены посредством HTTP- или WAP-заголовков.

Управление кэшем

Как и обычный Web-браузер, WAP-шлюз имеет кэш памяти. Он кэширует каждую колоду, которую посетил пользователь для того, чтобы быстро показать ее снова в случае необходимости без обращения к WAP-серверу.

Время, в течение которого устройство хранит колоду, называется *временем жизни* (TTL, time to live). По умолчанию WAP-шлюз имеет время жизни, равное 30 дням (или пока не закончатся ресурсы памяти). Если колода содержит информацию, чувствительную к времени, можно описать более короткий промежуток TTL, чтобы устройство чаще обращалось к серверу содержания. Следующий пример показывает, как задать более короткий интервал TTL:

```
<meta http-equiv="Cache-Control"
      content="max-age=3600" forua= true/>
```

Параметр max-age задает время в секундах. В примере устройство будет заново обращаться к серверу содержания по истечении часа. Чтобы определить правильное значение TTL для колоды, необходимо сбалансировать чувствительность информации к изменению времени с увеличением времени перезагрузки информации с сервера. Присвоение параметру max-age нулевого значения будет приводить к перезагрузке колоды при каждом обращении к ней. Однако можно ограничить количество таких дополнительных обращений, чтобы заставить браузер показывать колоду из кэша, если пользователь, например, нажимает клавишу <BACK>.

Закладки

Закладки беспроводного клиента совершенно аналогичны закладкам обычного Web-браузера. Если пользователь запоминает карту, браузер создает закладку, которая состоит из двух частей:

- ☐ метки, которая идентифицирует закладку (обычно включается из атрибута title элемента <card>);
- ☐ URL, который открывается, когда впоследствии пользователь выберет закладку для исполнения.

Так как все колоды запоминаются по умолчанию, элемент <meta> используется только для того, чтобы выключить режим запоминания имени колоды в закладке. Используемый в этом случае синтаксис приведен ниже:

```
<meta name="vnd.up.markable" forua="true" content="false"/>
```

При попытке создать закладку для карты, браузер автоматически устанавливает URL закладки на URL колоды. Если необходимо установить другой URL, следует использовать другой <meta>-элемент в заголовке колоды:

```
<meta name="vnd.up.bookmark" forua="true" content="url"/>
```

где url является тем URL, который вы хотите использовать в закладке.

Тег <noop>

Обязателен в стандарте WML 1.3.

Этот элемент указывает устройству на необходимость отсутствия действий. Полезен для блокировки действия элементов `<do>` уровня колоды, называемой *пассивизацией* (*shadowing*).

Синтаксис

```
<noop/>
```

Тег `<onevent>`

Обязателен в стандарте WML 1.3.

Этот элемент связан с внутренним состоянием перехода или внутренним событием задачи. При возникновении внутреннего события устройство выполняет связанную с этим событием задачу.

Синтаксис

```
<onevent type=" type"> task </onevent>
```

где *task* представляет действие для выполнения. При возникновении внутреннего события *task* описывает одно из следующих действий для события:

- ☐ `<go>`
- ☐ `<prev>`
- ☐ `<noop>`
- ☐ `<refresh>`

Атрибуты

Атрибут *type* определяет внутреннее событие, которое переключает задачу. Если `<onevent>` уровня карты конфликтует с таким же элементом уровня колоды, то выполняется элемент уровня карты.

Вы можете описать следующие значения атрибутов, указанные в табл. 2.2.

Таблица 2.2. Значения атрибута *type*

Значение <i>type</i>	Выполняемая задача
Onpick	Пользователь выбирает или отказывается от выбора части элемента <code><option></code>
Onenterforward	Пользователь идет по картам посредством задачи элемента <code><go></code>
Onenterbackward	Пользователь идет по картам посредством задачи элемента <code><prev></code> или вызывает механизм <code>prev</code> (например, нажимает клавишу <code><BACK></code>)
Ontimer	Описанный элемент <code><timer></code> указывает истечение срока давности

Тег **<optgroup>**

Необязателен в стандарте WML 1.3.

Этот элемент позволяет группировать несколько элементов `<option>` или вложенных элементов `<optgroup>` внутри одной карты. Создание таких групп позволяет описывать управляющую информацию о том, как устройство должно показывать содержание карты.

Синтаксис

```
<optgroup title="label">
```

```
content
```

```
</optgroup>
```

где *content* представляет один или более элементов `<optgroup>` или `<option>`.

Устройства показывают эти элементы в порядке, который им предписан.

Атрибуты

Атрибут `title` описывает короткую метку для группы `<optgroup>`. Некоторые устройства используют эту метку в качестве заголовка при показе содержания `<optgroup>`.

Тег **<option>**

Обязателен в стандарте WML 1.3.

Этот элемент описывает конкретный выбор пользователя внутри элемента `<select>`.

Синтаксис

```
<option title="label"
```

```
value="value"
```

```
onpick="url">
```

```
content
```

```
</option>
```

где *content* представляет текст, который устройства будут показывать, чтобы представить конкретный вариант выбора и действия, связанные с ним. Можно дополнительно указать за текстом возникающее событие и способ его обработки с помощью дополнительного внутреннего элемента `<onevent>`.

Атрибуты

❑ `title` — метка, связанная с действием. Используется некоторыми браузерами (UP.Browser) как ключ "Асепт". Для обеспечения совместимости

с широким диапазоном устройств, метка не должна превышать пяти символов.

- ❑ `value` — указывает значение, назначаемое переменной в элементе `<select>`.
- ❑ `onpick` — определяет URL, который открывается, если пользователь выбрал заданный элемент из списка. Этот элемент является вариацией элемента `<onevent>`.

Тег `<p>`

Обязателен в стандарте WML 1.3.

Этот элемент описывает новый параграф и имеет атрибуты выравнивания и обтекания строки.

Синтаксис

```
<p align="alignment"
  mode="wrapmode">
content
</p>
```

Атрибуты

- ❑ `align` — имеет значение `left`, `right` или `center`. Задаёт выравнивание относительно области показа. Указание `<p>` без атрибута выравнивания приводит к выравниванию по левому краю.
- ❑ `mode` — имеет значение `wrap` или `nowrap`. Задаёт режим обтекания. При `mode="nowrap"` устройство использует другой механизм, такой как горизонтальный скроллинг, чтобы показать длинные строки пользователю. Устройство продолжает применять этот режим, пока не встретит другой элемент `<p>` с иным режимом. Другими словами:
 - если описан этот атрибут, устройство применяет этот режим;
 - если атрибут не описан, устройство применяет тот режим, который был описан в последнем элементе `<p>`;
 - при отсутствии предыдущего `<p>` используется режим умолчания (`wrap`).

Тег `<postfield>`

Обязателен в стандарте WML 1.3.

Этот элемент `<postfield>` определяет пары "имя/значение", которые передаются HTTP-серверу, получающему запрос `<go>`.

Синтаксис

```
<postfield name= "name"  
            value=" value"/>
```

Атрибуты

- ❑ `name` — метка, которая идентифицирует поле.
- ❑ `value` — строка, задающая значение этого поля по умолчанию.

Тег *<prev>*

Обязателен в стандарте WML 1.3.

Этот элемент относится к категории `<task>`, которая заставляет устройство удалять текущий URL из стека и открывать предыдущий URL. Если такого нет, действие элемента не имеет эффекта.

Синтаксис

```
<prev> content </prev>
```

где `content` представляет переменные, которые устанавливаются перед переходом на предыдущий URL.

В отличие от других элементов WML, имеющих содержание, описание содержания для `<prev>` является опцией. Если оно не описано, следует использовать `<prev/>`, а не `<prev> content </prev>`.

В элементе `<prev>` можно дополнительно указать один или несколько элементов `<setvar>`.

Тег *<receive>*

Расширение стандарта от компании Openwave.

Этот элемент описывает имя переменной.

Синтаксис

```
<receive name="X"/>
```

Элемент `<receive>` без атрибута `name` приведет к тому, что значение блока параметров будет проигнорировано браузером.

При получении блока параметров WML-агент должен назначить каждой объявленной переменной значение из блока полученных параметров.

Если в блоке параметров недостаточно значений, каждое дополнительное получение блока должно быть обработано, как если бы в строке параметров были пустые строки в этих позициях.

Атрибуты

- `name` — описывает имя переменной. Приводит к ошибке, если значение имени не является разрешенным именем переменной.

Тег **<refresh>**

Обязателен в стандарте WML 1.3.

Элемент из конструкции `task`, который предписывает устройству перезагрузить указанные переменные карты. Устройство также заново обновляет экран, если эти переменные показаны на нем.

Синтаксис

```
<refresh> content </refresh>
```

где `content` представляет переменные, которые нужно обновить, при этом должна присутствовать, по меньшей мере, одна переменная.

Тег **<reset>**

Этот элемент заставляет очищать все переменные в текущем содержании. Возможен лишь один такой элемент на карте. Если элемент, описывающий задачу (`<go>`, `<prev>` или `<refresh>`), содержит тег `<reset>`, то действие по сбросу значений переменных текущего содержания выполняется при активизации этой задачи.

Синтаксис

```
<go href="/bar">  
  <reset/>  
</go>
```

Тег **<select>**

Обязателен в стандарте WML 1.3.

Этот элемент описывает список выбора для пользователя. Возможно описание одного выбора из многих и многих из многих.

Синтаксис

```
<select title="label"  
  multiple="boolean"  
  name="variable"
```

```
value="value"  
iname="index-variable"  
ivalue="default-value"  
tabindex="n">
```

```
content  
</select>
```

где *content* представляет текст или графический объект, который устройство будет показывать как подсказку перед списком выбора, и может включать любой набор любого из элементов `<optgroup>` или `<option>`.

Атрибуты

- ❑ `title` — описывает краткую метку для списка. Некоторые устройства показывают ее как заголовок при отображении списка выбора.
- ❑ `multiple` — имеет значение `true` или `false`. Указывает, может ли пользователь выбирать сразу несколько элементов списка.
- ❑ `name` — имя переменной, в которой устройство хранит значения, связанные с выбором (выборами) пользователя. Это значение, связанное с каждым элементом выбора, берется из `value`-атрибута элемента `<option>`. Значение описанной переменной определяет выбор по умолчанию, который устанавливается, когда устройство показывает элемент `<select>`. Если переменная не имеет значения, устройство присваивает ей значение атрибута `value`.
 - Если не описывать значение по умолчанию, устройство должно использовать для инициализации переменной пустую строку (`""`).
 - В случае множественного выбора, значения хранятся как список с разделителями в виде точки с запятой.
- ❑ `value` — строка, описывающая значения по умолчанию для переменных, указанных в атрибуте `name`. Если атрибут `name` уже имеет значение в процессе навигации элемента `<select>`, устройство игнорирует значение атрибута `value`.
- ❑ `iname` — идентично атрибуту `name` за исключением того, что указанная переменная хранит индекс значений, связанных с выбором пользователя. Значения индекса соответствуют позиции элемента выбора в списке, начиная с 1. При отсутствии выбора значение индекса равно 0.
- ❑ `ivalue` — имя переменной, которая содержит значения выбранных элементов списка. При множественном выборе значения отделяются точкой с запятой, пустые значения считаются `invalid`.
- ❑ `tabindex` — зависит от типа используемого браузера. UP.Browser не поддерживает этот атрибут.

Тег **<send>**

Расширение стандарта от компании Openwave.

Этот элемент описывает одно значение, включенное в блок параметров. Каждое вхождение идентифицируется его позицией в блоке параметров, а позиция, в свою очередь, определяется порядком указания элементов `<send>`.

Синтаксис

```
<send value="$X99"/>
```

Атрибуты

- ☐ `value` — указывает данные, которые должны быть отправлены в данной позиции блока параметров. По умолчанию — пустая строка.

Тег **<setvar>**

Обязателен в стандарте WML 1.3.

Этот элемент устанавливает переменную на заданное значение при исполнении устройством задачи `<go>`, `<prev>` или `<refresh>`.

Синтаксис

```
<setvar name="name"  
        value="value"/>
```

Атрибуты

- ☐ `name` — имя переменной. Устройство игнорирует элемент `<setvar>`, если имя не определено.
- ☐ `value` — значение, присваиваемое переменной.

Тег **<small>**

Необязателен в стандарте WML 1.3.

Это необязательный элемент форматирования текста.

Синтаксис

```
<small> text </small>
```

где `text` отображается уменьшенным шрифтом по отношению к основному тексту.

Тег `<spawn>`

Расширение стандарта от компании Openwave.

Этот элемент объявляет задачу `spawn`, указывая на создание порожденного содержания и вызов URL в этом порожденном контексте.

Если URL именует WML-карту или колоду, соответствующая карта показывается, а URL становится основой нового стека истории в порожденном контексте.

Когда это содержание покидается через использование задачи `<exit>`, возникает внутреннее событие `onexit`, которое может быть обработано с помощью атрибута `onexit` тега `<onevent>` внутри элемента `<spawn>`. Задача `<spawn>` может инициировать создание передаваемых переменных с помощью элемента `<setvar>`. Возвращаемые параметры указываются в элементе `<receive>`, а исключения обрабатываются посредством конструкции `<catch>`.

Элемент `<spawn>` может также содержать один или более элементов `<postfield>`.

Синтаксис

```
<spawn href="/child"
      onexit="/continue">
  <setvar name="Name" value="Joe"/>
</spawn>
```

Элемент `<spawn>` создает новое содержание браузера и вызывает порожденный URL в это содержание.

Атрибуты

- ☐ `href` — указывает URL карту, которая должна быть показана.
- ☐ `onexit` — событие `onexit` возникает при выходе из карты порожденного содержания с использованием задачи `exit`.
- ☐ `sendreferer` — имеет значение `true` или `false`. Описывает, следует ли устройству включать URL в HTTP-запрос. Указание `sendreferer="true"` будет заставлять устройство устанавливать заголовок `HTTP_REFERER` на относительный адрес URL запрашиваемой колоды. Для ограничения доступа к защищенным сервисам колоды, запрашивающие этот URL, должны включать эту опцию в режим `true`.
- ☐ `method` — имеет значение `get` или `post`. Указывает используемый метод протокола HTTP. Задание `method="post"` заставляет шлюз перекодировать данные передаваемых переменных в кодировку, указанную в заголовке HTTP вашего приложения. Необходимо выполнить перекодирование, если передается не ASCII-символ (особенно UTF-8).

- `accept-charset` — указывает тип кодировки, которую должно воспринимать приложение на сервере. Устройство использует этот атрибут для перекодировки данных, указанных в элементе `<postfieldt>`. Обычно по умолчанию предполагается UTF-8, для которой US-ASCII является подмножеством. Таким образом, WML-услуги в США, Канаде или Австралии могут не использовать этот атрибут. Атрибут можно опустить в случае задания его в заголовке HTTP-запроса. Заметим, что атрибут `accept-charset` превалирует над аналогичными атрибутами, указанными в заголовке HTTP-запроса.

Синтаксис для этого атрибута предписывает разделять запятыми или пробелами кодировки из IANA-кодировок. Например:

```
accept-charset="UTF-8, US-ASCII, ISO-8859-1"
```

Полный регистр IANA-кодировок приведен по адресу <http://www.iana.org/>.

Тег ******

Необязателен в стандарте WML 1.3.

Это необязательный элемент, предписывающий браузеру выделить текст.

Синтаксис

```
<strong> text </strong>
```

где *text* будет показан полужирным шрифтом.

Тег ***<table>***

Обязателен в стандарте WML 1.3.

Этот элемент позволяет описать формат табулированных данных. Возможно управление выравниванием табулированных данных.

WML-таблицы аналогичны HTML-таблицам с несколькими исключениями. При определении таблицы необходимо указать количество столбцов с последующим содержанием. Содержание может включать пустые строки и столбцы.

Синтаксис

```
<table title="name"
      align="left|right|center"
      columns="number of columns">
...data
</table>
```

Атрибуты

- ☐ `align` — принимает значение `left`, `right` или `center`. Задаёт выравнивание текста относительно столбца.
- ☐ `title` — задаёт метку таблицы.
- ☐ `columns` — задаёт количество столбцов для набора строк. Указание нуля не разрешено.

Тег `<td>`

Обязателен в стандарте WML 1.3.

Этот элемент используется в качестве контейнера для хранения одной ячейки таблицы внутри строки. Ячейка может быть пустой.

Синтаксис

```
<td> content </td>
```

В браузере UP.Browser *content* может иметь в качестве содержания следующие элементы:

- ☐ ``
- ☐ `<anchor>`

Тег `<tr>`

Обязателен в стандарте WML 1.3.

Этот элемент используется в качестве контейнера для хранения строки таблицы. Таблица может быть пустой.

Синтаксис

```
<tr>  
  <td> content </td>  
</tr>
```

где *content* представляет текст внутри ячейки или графический объект, или `<anchor>`.

Тег `<template>`

Обязателен в стандарте WML 1.3.

Колода WML может содержать элемент `<template>`, который определяет характеристики всей колоды карт.

Эти характеристики могут позднее изменяться указанием аналогичных характеристик конкретной карты.

Синтаксис

```
<wml>
  <template onenterforward="url"
            onenterbackward="url"
            ontimer="url">
    content
  </template>
</wml>
```

где *content* представляет общие действия при возникновении событий в колоде.

Параметры колоды могут быть заданы с использованием следующих элементов определения темплетов:

- ☐ `<do>`
- ☐ `<onevent>`

Атрибуты

- ☐ `onenterforward` — описывает URL, который будет открываться, если пользователь переходит к карте с помощью задачи `<go>`. Этот атрибут представляет сжатую форму элемента `<onevent>`.
- ☐ `onenterbackward` — описывает URL, который будет открываться в случае, если пользователь переходит к следующей карте с помощью задачи `<prev>`. Представляет сжатую форму элемента `<onevent>`.
- ☐ `ontimer` — описывает URL, который открывается, если время, указанное в элементе `<timer>`, истекает. Представляет сжатую форму элемента `<onevent>`.

Тег *<throw>*

Расширение стандарта от компании Openwave.

Этот элемент объявляет задачу, которая выполняется при возникновении исключительных событий.

Значения могут быть посланы обработчику исключений посредством элементов `<send>`, включенных в конструкцию `<throw>`. Начало обработки исключений завершает текущее содержание и приводит к разрушению содержания на WML-клиенте, включая все виды переменных и историю состояний клиента.

Если родительское содержание не имеет обработчика исключений (элемента `<catch>`), который требуется в этом исключении, такое содержание также разрушается.

Эта операция повторяется до тех пор, пока правильный обработчик исключений не будет найден или все типы содержания не завершатся. В последнем случае UP.Browser выполняет возврат к первичному состоянию.

Например, элемент `<throw>` обрабатывает исключение с именем "user input error". Кроме того, блок параметров включается для передачи большей информации об ошибке.

Синтаксис

```
<throw name="user input error">  
  <send value="Bad numeric value"/>  
</throw>
```

Атрибуты

- `name` — описывает имя исключения. Это имя используется для поиска корректного обработчика исключения. Атрибут должен быть написан в нижнем регистре.

Тег *<timer>*

Обязателен в стандарте WML 1.3.

Этот элемент `<timer>` задает метод автоматического вызова задачи по достижении некоторого периода пользовательской неактивности. Любая задача или действие пользователя начинают отсчет времени неактивности, а следующая задача или действие пользователя прекращает отсчет и начинает его заново. Имеется возможность связать с моментом завершения заранее заданного периода неактивности выполнение некоторой задачи.

Синтаксис

```
<timer name="variable"  
  value="value"/>
```

Атрибуты

- `name` — имя переменной, в которой устройство хранит значение таймера. Если переменная не имеет значения, когда таймер инициализирован, устройство должно установить значение по умолчанию. Это либо текущее значение таймера, при котором пользователь покидает карту, либо 0, если время таймера истекло.

Если атрибут `name` уже имеет значение при инициализации таймера, устройство игнорирует значение по умолчанию. Если атрибут `name` не существует, устройство устанавливает его на значение, указанное в атрибуте `value`.

- ❑ `value` — строка, описывающая значение переменной по ключевому атрибуту. Его нужно указать в десятых долях секунды, например значение 100 соответствует 10 секундам. Задание значения 0 выключает таймер.

Тег `<u>`

Необязателен в стандарте WML 1.3.

Это необязательный элемент выделения текста подчеркиванием.

Синтаксис

```
<u> text </u>
```

Текст будет подчеркнут.

Тег `<wml>`

Обязателен в стандарте WML 1.3.

Элемент `<wml>` задает WML-колоду.

Синтаксис

```
<wml xml:lang="lang">
  <card>
    content
  </card>
</wml>
```

где `content` представляет внутренние элементы колоды.

Атрибуты

- ❑ `xml:lang` — описывает, используя естественный или формальный язык для WML-документа. Задание атрибута `xml:lang` имеет более высокий приоритет, чем любая спецификация языка в документе.

См. формальную спецификацию XML "Extensible Markup Language (XML), W3C Proposed Recommendation" по адресу <http://www.w3.org>.

Краткое описание требований к GSM-приложениям в m-services

Инициатива m-services, принятая членами ассоциации GSM, включает ряд рекомендаций и список требований к создаваемым в сетях GSM услугам на

основе стандарта WAP. В данном приложении кратко описаны требования к беспроводным приложениям, работающим в сетях GSM.

- ❑ Требование обратной совместимости. WML-приложения, направленные на доставку общедоступной информации, не должны приводить к ошибкам интерпретации в HDML-браузерах Phone.com версий 3.0, 3.1 и 3.2.
- ❑ Не нужно использовать <BACK> в качестве программируемой ключевой метки.
- ❑ Не нужно ссылаться на первую карту приложения как на "HOME PAGE".
- ❑ Размер колоды не должен превышать 1492 байт.
- ❑ Сервис должен завершаться выходом из него.
- ❑ Элемент обратной навигации должен быть определен для каждой карты.
- ❑ Не разрешено задавать задачу prev с действием <noop/>.
- ❑ Нельзя разрешать пользователю возвращаться к карте авторизации и просматривать пароль.
- ❑ Если предоставлена возможность входа в сервис после авторизации, то нужно предоставить также и возможность выхода из режима авторизации.
- ❑ Возможность выхода (logout) предоставляется пользователю только в том случае, если он еще не вышел из режима.
- ❑ Автоматическая доставка напоминаний (alerts) должна быть инициирована только в том случае, если она была запрошена пользователем.
- ❑ URL в подсказке должен быть активным и доступным минимум в течение 24 часов после отправки.
- ❑ Заголовок напоминания не должен превышать 24 символов.
- ❑ Клавишу <TALK> не нужно использовать в беспроводном приложении для других целей, кроме как для инициирования телефонного вызова.
- ❑ Клавишу <BACK> (<CLR>) нужно использовать только для выполнения действия prev.
- ❑ Не нужно добавлять скобки для выделения ссылок.
- ❑ Используйте форматирование шрифта только в крайних случаях.
- ❑ Метки программируемых действий не должны превышать 7 символов.
- ❑ Назначайте метки каждому событию, заданному в элементе <do>, за исключением <prev>.
- ❑ Приложения, спроектированные для работы с любым телефоном, должны использовать только две "горячих" клавиши на каждой карте.
- ❑ Следующий список показывает перечень запрещенных к использованию меток: "HOME", "help", "back", "clear", "clr", "end", "power", "pwr".

- ☐ Необходимо всегда задавать первую "горячую" клавишу.
- ☐ Не использовать одинаковые метки для разных "горячих" клавиш.
- ☐ Не определять более одной конструкции `<do type=accept>` на карте.
- ☐ Не определять более одной конструкции `<do type=options>` на карте.
- ☐ Не определять `<do type=delete>` или `<do type=help>` в качестве единственного определения необходимой функциональности.
- ☐ Не дублировать заданные на "горячих" клавишах действия в ссылки, размещенные на карте.
- ☐ Таблица должна быть не шире 14 символов.
- ☐ Список выбора в режиме "wgap" не должен превышать три строки текста.
- ☐ В списке множественного выбора все элементы должны иметь обработку событий `onpick`.
- ☐ Либо все элементы списка должны иметь событие `onpick`, либо ни один.
- ☐ Все элементы `input` должны иметь заголовок.
- ☐ Формат ввода пароля можно задавать только тогда, когда он полностью числовой.
- ☐ На карте `input` все используемые клавиши должны быть стандартными (используемыми по умолчанию в данном браузере).
- ☐ Используйте только черно-белую графику.
- ☐ Графический объект должен быть представлен в WBMP-формате.
- ☐ Не создавайте размеченных активных графических объектов.
- ☐ Графический объект не должен превышать 96 пикселей в ширину.

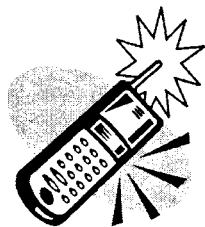
Правила преобразования текстов WML версий 1.3 и 2.0

В этом разделе представлен краткий обзор изменений, который внесен в предварительную версию 2.0 языка WML по сравнению с представленной в главе версией 1.3. Правила преобразования приведены в *приложении 3*.

- ☐ Вместо того чтобы определять собственный тип документа, WML 2.0 является расширением следующих модулей языка XHTML. Это означает, что некоторые элементы теперь имеют больше атрибутов, и имеется также много новых элементов языка:
 - каждый модуль XHTML поддерживается XHTML Basic;
 - модуль представления уровня блока;
 - модуль вложенной таблицы стилей;
 - модуль таблицы стилей.

- ❑ Каждый элемент объявляет собственное пространство имен XML. По умолчанию XHTML. Пространство имен WML ограничено префиксом "wml".
- ❑ Корневым элементом теперь является "html" вместо "wml".
- ❑ Элемент `head` является обязательным. Если этот элемент в документе отсутствует, он должен быть вставлен на этапе преобразования.
- ❑ Элемент `title` языка XHTML в заголовке документа теперь является обязательным. В процессе преобразования из WML 1.3 названием документа становится название первой карты документа.
- ❑ Элемент `template` удален из спецификации. Вместо этого элементы уровня темплет могут быть помещены сразу после элемента `head`. При преобразовании элементы из блока `template` должны быть перемещены в новую позицию.
- ❑ Теневые элементы `do` исключены из спецификации. Процесс преобразования распространяет элементы `do`-уровня темплета внутрь карты документа в соответствии с заданным алгоритмом (`shadowing algorithm`).
- ❑ Атрибут `ordered` элемента `card` удален из спецификации. Игнорируется в процессе трансформации.
- ❑ Вложенный элемент `optgroup` теперь является некорректной конструкцией. Преобразуется только первый уровень `optgroup`. Все внутренние уровни сворачиваются в плоский список элементов `optgroup`.
- ❑ Новый атрибут `label` для `optgroup` является обязательным. Преобразование вставляет пустую метку `label`.
- ❑ Атрибут WML `type` элемента `do` теперь именуется `role`, и также имеет некоторые новые значения.
- ❑ Атрибут `optional` для элемента `do` был удален из спецификации. Игнорируется в процессе трансформации.
- ❑ Новый элемент `widget` может быть использован внутри элемента `do` для описания предпочтительных типов представления.
- ❑ Элемент `element` может быть использован в большем количестве мест, чем в предыдущей версии языка WML. Например, внутри элемента `td`.
- ❑ WML 2.0 расширяет элементы XHTML с атрибутами WML и расширяет модель документа XHTML путем использования элементов WML. В некоторых случаях WML 2.0 также расширяет модель документа XHTML следующими элементами XHTML.
 - Таблицы. В WML 2.0 элемент `table` может находиться внутри элемента `p`. Это недопустимо в XHTML и разрешено в WML 2.0.
 - Элементы управления формой (`input` и `select`) могут размещаться внутри элемента `pre`. Это недопустимо в XHTML и разрешено в WML 2.0.

Глава 3



Справочник по WMLScript

Материал этой главы предназначен для программистов, активно использующих язык WMLScript в своей работе. Приведенная информация может быть также полезна начинающим разработчикам беспроводных приложений. Представленные сведения включают перечень всех стандартных библиотек, которые должны быть реализованы на WAP-совместимом шлюзе версии 1.1.

Все примеры могут быть проверены с использованием тестовой колоды следующего вида:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <head>
    <meta http-equiv="Cache-Control" content="max-age=0" forua="true"/>
  </head>
  <card id="Test">
    <p align="center"> Lang.isInt<br/> </p>
    <p>
      <do type="accept">
        <go href="isint.wmls#isIntTest()"/>
      </do>
      Press OK to test the Lang.isInt function
    </p>
  </card>
</wml>
```

Операторы присвоения

Операторы присвоения присваивают значение переменной. WMLScript поддерживает операторы присвоения, указанные в табл. 3.1.

Таблица 3.1. Операторы присвоения

Оператор	Операция
=	Присвоить
+=	Добавить (числа)/ конкатенировать (строки) и результат присвоить
-=	Вычесть и присвоить
*=	Умножить и присвоить
/=	Разделить и присвоить
Div=	Разделить (целочисленное деление) и присвоить
%=	Взять остаток и назначить (знак результата равен знаку делимого)
<=	Присвоение результата побитового сдвига влево
>>=	Присвоение результата побитового сдвига вправо с сохранением знака
>>>=	Присвоение результата побитового сдвига вправо с заполнением ведущих битов нулями
&=	Присвоение результата побитового И
^=	Присвоение результата побитового исключающего ИЛИ
=	Присвоение результата побитового ИЛИ

Арифметические операторы

WMLScript поддерживает все основные арифметические действия и операции, которые указаны в табл. 3.2 — 3.4.

Таблица 3.2. Арифметические операторы (бинарные арифметические действия)

Оператор	Операция
+	Сложение (чисел)/конкатенация (строк)
-	Вычитание
*	Умножение
/	Деление
Div	Целочисленное деление

Таблица 3.3. Арифметические операторы (двоичные операции)

Оператор	Операция
%	Остаток от деления, знак результата равен знаку делимого
<<	Побитовый сдвиг влево

Таблица 3.3 (окончание)

Оператор	Операция
>>	Побитовый сдвиг вправо с сохранением знака
>>>	Побитовый сдвиг вправо с ведущими нулями
&	Побитовое И
	Побитовое ИЛИ
^	Побитовое исключающее ИЛИ

Таблица 3.4. Арифметические операторы (унарные операции)

Оператор	Операция
+	Плюс
-	Минус
--	Пре- или пост-уменьшение
++	Пре- или пост-увеличение
~	Побитовое НЕ

Пример:

```
var bill = 1/7;
var rate = bill*3(++b);
```

Логические операторы

WMLScript поддерживает логические действия, указанные в табл. 3.5.

Таблица 3.5. Логические операторы

Оператор	Операция
&&	Логическое И
	Логическое ИЛИ
!	Логическое НЕ

Оператор логического И оценивает первый операнд и проверяет результат. Если результат имеет значение `false`, то результат операции становится `false`, а второй операнд не рассматривается. Если значение первого операн-

да оценивается как `true`, то результатом операции становится оценка значения второго операнда. Если первый операнд `invalid`, то второй операнд не оценивается и результат операции `invalid`. Аналогично, для логического ИЛИ оценивается первый операнд и тестируется результат. Если результат имеет значение `true`, то результат операции `true`, а второй операнд не оценивается. В противном случае, результатом операции становится оценка значения второго операнда. Если первый операнд имеет значение `invalid`, то второй операнд не оценивается и результатом операции становится `invalid`.

Пример:

```
weAgree = (iAmRight && youAreRight) ||  
(!iAmRight && !youAreRight);
```

WMLScript требует значения логического типа для логических операций. При этом поддерживается автоматическое преобразование типов логических операций.

Замечание

Если значение первого операнда в логическом И или ИЛИ имеет значение `invalid`, второй операнд не оценивается, а результат операции становится `invalid`.

```
var a = (1/0) || foo(); // result: invalid, no call to foo()  
var b = true || (1/0); // true  
var c = false || (1/0); // invalid
```

Операции действия со строками

WMLScript поддерживает конкатенацию строк как встроенную операцию. Операторы `+` и `=` используются для выполнения конкатенации строк. Другие строковые операции выполняются с использованием стандартной библиотеки работы со строками.

Пример:

```
var str = "Start" + "End";  
var chr = String.charAt(str,6); // chr = "E"
```

Операторы сравнения

WMLScript поддерживает все основные операции сравнения, что и показано в табл. 3.6.

Таблица 3.6. Операторы сравнения

Оператор	Операция
<	Меньше чем
<=	Меньше или равно
==	Равно
>=	Больше или равно
>	Больше чем
!=	Не равно

Оператор *typeof*

Несмотря на то, что WMLScript относится к слабо типизированному языку, поддерживаются следующие основные типы данных:

- ☐ boolean
- ☐ integer
- ☐ floating-point
- ☐ string
- ☐ invalid

Оператор *typeof* возвращает целое значение, которое описывает тип данного выражения. Возможные значения результата указаны в табл. 3.7.

Таблица 3.7. Возможные результаты при использовании оператора *typeof*

Тип	Код
Целое (integer)	0
Плавающая точка (floating-point)	1
Строка (string)	2
Булево (boolean)	3
Invalid	4

Оператор *typeof* не пытается конвертировать результат из одного типа в другой, а возвращает его тип как есть после оценки выражения.

Язык WMLScript имеет средства управления процессом выполнения сценариев, которые реализованы в виде команд управления. Команды WMLScript состоят из выражений и ключевых слов. При этом одна команда может занимать несколько строк, и одна строка может включать несколько команд.

Ниже приведено краткое описание основных команд языка WMLScript:

- ☐ empty
- ☐ expression
- ☐ block
- ☐ break
- ☐ continue
- ☐ for
- ☐ if...else
- ☐ return
- ☐ var
- ☐ while

Основные команды языка WMLScript указаны в табл. 3.8.

Таблица 3.8. Основные библиотеки языка WMLScript

Имя	Описание
Lang	Библиотека содержит набор функций, которые реализуют основные возможности языка WMLScript
Float	Библиотека содержит набор типичных арифметических функций операций с плавающей точкой, которые часто используются приложениями. Поддержка исполнения функций этой библиотеки является дополнительной возможностью и выполняется только устройствами, которые могут поддерживать такую функциональность (в основном карманные компьютеры и коммуникаторы). Если операторы с плавающей точкой не поддерживаются, то все функции этой библиотеки должны возвращать в браузере значение <code>invalid</code>
String	<p>Библиотека содержит набор строковых функций. Строки являются массивами символов. Каждый символ имеет индекс. Первый символ в строке имеет индекс ноль (0). Длина строки задает количество символов в массиве. Пользователь этой библиотеки может описать специальный разделитель для строк. Эти элементы могут адресоваться с помощью специального разделителя и индекса элемента. Первый элемент в строке имеет индекс ноль (0). Каждое вхождение разделителя в строку отделяет два соседних элемента.</p> <p>"Пустой символ" может быть одним из следующих символов:</p> <ul style="list-style-type: none"> <input type="checkbox"/> TAB: горизонтальная табуляция; <input type="checkbox"/> VT: вертикальная табуляция; <input type="checkbox"/> FF: перевод формы; <input type="checkbox"/> SP: пробел; <input type="checkbox"/> LF: перевод строки; <input type="checkbox"/> CR: возврат каретки

Таблица 3.8 (окончание)

Имя	Описание
URL	<p>Эта библиотека содержит набор функций для обработки абсолютных и относительных URL.</p> <p>В настоящее время эта библиотека поддерживает доступ только к подмножеству элементов URL, определенных в [RFC2396]</p>
WMLBrowser	<p>Эта библиотека содержит функции, посредством которых WMLScript может получать доступ к WML-содержанию. Эти функции не должны иметь никакого визуального эффекта.</p> <p>Если система не имеет WML-браузера, или если интерпретатор WMLScript не был вызван WML-браузером, эти функции всегда возвращают <code>invalid</code></p>
Dialogs	Библиотека содержит набор типичных функций пользовательского интерфейса
Debug	Библиотека компании Nokia, используемая для отладки программных модулей
Console	Библиотека компании Phone.Com, используемая для отладки программных модулей

Языковая библиотека

Языковая библиотека содержит информацию о функциях поддерживаемых WMLScript.

Библиотека *Lang*

Функция *abs*

Функция возвращает абсолютное значение числа. Тип результата совпадает с типом аргумента.

Пример

```
Extern function AbsTest()
{
    var Argument1 = -5;
    var Result1 = Lang.abs(Argument1);
    var Argument2 = -7.;
    var Result2 = Lang.abs(Argument2);
    var ResultString = "Lang.abs()" + "\r\r" +
        "abs(-5) = " + Result1 + "\r" +
```

```

        "abs(-7.) = " + Result2;
    Dialogs.alert (ResultString);
}

```

Результат выполнения кода

```

Lang.abs()
abs(-5)=5
abs(-7.)=7.000000e+000

```

Функция *min*

Функция возвращает минимальное значение из пары чисел. Тип результата совпадает с типом выбранного числа. Выбор производится следующим образом:

1. Применяется правило преобразования типов данных оператора WMLScript для выполнения процедуры сравнения.
2. Сравниваются числа, и выбирается наименьшее. В случае равных значений выбирается первое.

Пример

```

Extern function MinTest()
{
    var Argument1 = 5.0;
    var Argument2 = 500;
    var Result1 = Lang.min(Argument1, Argument2);
    var Argument3 = -7;
    var Argument4 = -10.0;
    var Result2 = Lang.min(Argument3, Argument4);
    var Argument5 = "String";
    var Argument6 = 0.0;
    var Result3 = Lang.min(Argument5, Argument6);
    var ResultString = "Lang.min()" + "\r\r" +
        "min(5.0, 500) = " + String.toString(Result1) + "\r" +
        "min(-7, -10.0) = " + String.toString(Result2) + "\r" +
        "min(String, 0.0) = " + String.toString(Result3);
    Dialogs.alert (ResultString);
}

```

Результат выполнения кода

```

Lang.min()
min(5.0,500)=1.000000e+000
min(-7,-.0.0)=-1.000000e+00
min(String,0.0)=Invalid

```

Функция *max*

Функция возвращает максимальное значение из пары чисел. Тип результата совпадает с типом выбранного числа. Выбор производится следующим образом:

1. Применяется правило преобразования типов данных оператора WMLScript для выполнения процедуры сравнения.
2. Сравниваются числа, и выбирается наибольшее. В случае равных значений выбирается первое.

Пример

```
Extern function MaxTest()  
{  
    var Argument1 = 5.0;  
    var Argument2 = 500;  
    var Result1 = Lang.max(Argument1, Argument2);  
    var Argument3 = -7;  
    var Argument4 = -10.0;  
    var Result2 = Lang.max(Argument3, Argument4);  
    var Argument5 = "String";  
    var Argument6 = 0.0;  
    var Result3 = Lang.max(Argument5, Argument6);  
    var ResultString = "Lang.max()" + "\r\r" +  
        "max(5.0, 500) = " + String.toString(Result1)  
        + "\r" +  
        "max(-7, -10.0) = " + String.toString(Result2)  
        + "\r" +  
        "max(String, 0.0) = " + String.toString(Result3);  
    Dialogs.alert(ResultString);  
}
```

Результат выполнения кода

```
Lang.max()  
max(5.0,500)=500  
max(-7,-10.0)=7  
max(String,0.0)=Invalid
```

Функция *parseInt*

Функция возвращает целое значение, полученное путем преобразования строки, содержащей только цифры. При этом разбор заканчивается при первом появлении в строке символа, отличного от следующих: лидирующий "+", или "-", или цифра.

Пример

```
Extern function ParseIntTest()
{
    var Argument1 = "5";
    var Result1 = Lang.parseInt(Argument1);
    var Argument2 = "15 sq. in.";
    var Result2 = Lang.parseInt(Argument2);
    var Argument3 = "A = 5";
    var Result3 = Lang.parseInt(Argument3);
    var ResultString = "Lang.parseInt()" + "\r\r" +
        "parseInt(5) = " + String.toString(Result1) + "\r" +
        "parseInt(15 sq. in.) = " + String.toString(Result2) + "\r" +
        "parseInt(A = 5) = " + String.toString(Result3);
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
Lang.parseInt()
parseInt(15 sq.in)=15
parseInt(A=5)=Invalid
```

Функция *ParseFloatvalue*

Функция возвращает десятичное значение, полученное путем преобразования строки, содержащей цифры. При этом разбор заканчивается при первом появлении в строке символа, который не может быть частью представления десятичного числа с плавающей точкой.

Пример

```
Extern function ParseFloatTest()
{
    var Argument1 = "5.0";
    var Result1 = Lang.parseFloat(Argument1);
    var Argument2 = "15 sq. in.";
    var Result2 = Lang.parseFloat(Argument2);
    var Argument3 = "A = 5";
    var Result3 = Lang.parseFloat(Argument3);
    var ResultString = "Lang.parseFloat()" + "\r\r" +
        "parseFloat(5.0) = " + String.toString(Result1) + "\r" +
        "parseFloat(15 sq. in.) = " + String.toString(Result2) + "\r" +
        "parseFloat(A = 5) = " + String.toString(Result3);
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
Lang.parseFloat()  
parseFloat(5.0)=5.000000e+000  
parseFloat(15 sq.in)=1.500000e+001  
parseFloat(A=5)=invalid
```

Функция *isInt*

Функция возвращает булево значение true, если заданное число может быть преобразовано в целое с использованием функции ParseInt(value). В противном случае возвращается значение false.

Пример

```
Extern function isIntTest()  
{  
    var Argument1 = "5.0";  
    var Result1 = Lang.isInt(Argument1);  
    var Argument2 = "7";  
    var Result2 = Lang.isInt(Argument2);  
    var Argument3 = "String";  
    var Result3 = Lang.isInt(Argument3);  
    var ResultString = "Lang.isInt()" + "\r\r" +  
        "isInt(5.0) = " + Result1 + "\r" +  
        "isInt(7) = " + Result2 + "\r" +  
        "isInt(String) = " + Result3;  
    Dialogs.alert(ResultString);  
}
```

Результат выполнения кода

```
Lang.isInt()  
IsInt(5.0)=false  
IsInt(7)=true  
IsInt(String)=false
```

Функция *isFloat*

Функция возвращает булево значение, равное true, если value может быть преобразовано с помощью функции ParseFloat(value) в число с плавающей точкой. В противном случае возвращается значение false.

Пример

```
<?xml version="1.0"?>  
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
```

```

"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <head>
    <meta http-equiv="Cache-Control" content="max-age=0" forua="true"/>
  </head>
  <card id="Test">
    <p align="center">
      Lang.isFloat<br/>
    </p>
    <p>
      <do type="accept">
        <go href="isfloat.wmls#isFloatTest()" />
      </do>
      Press OK to test the Lang.isFloat function
    </p>
  </card>
</wml>

```

Результат выполнения кода

```

Lang.is.Float()
IsFloat(5.0)=true
IsFloat(7)=true
IsFloat(String)=false

```

Пример

```

Extern function isFloatTest()
{
  var Argument1 = "5.0";
  var Result1 = Lang.isFloat(Argument1);
  var Argument2 = "7";
  var Result2 = Lang.isFloat(Argument2);
  var Argument3 = "String";
  var Result3 = Lang.isFloat(Argument3);
  var ResultString = "Lang.isFloat()" + "\r\r" +
    "isFloat(5.0) = " + Result1 + "\r" +
    "isFloat(7) = " + Result2 + "\r" +
    "isFloat(String) = " + Result3;
  Dialogs.alert(ResultString);
}

```

Результат выполнения кода

```
Lang.isFloat()
IsFloat(5.0)=true
IsFloat(7)=true
IsFloat(String)=false
```

Функция *maxInt*

Функция возвращает максимальное целое значение.

Пример

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
           "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <head>
    <meta http-equiv="Cache-Control" content="max-age=0"
          forua="true"/>
  </head>
  <card id="Test">
    <p align="center"> Lang.maxInt<br/>  </p>
    <p>
      <do type="accept">
        <go href="maxInt.wmls#maxIntTest()" />
      </do>
      Press OK to test the Lang.maxInt function
    </p>
  </card>
</wml>
extern function maxIntTest()
{
  var Result = Lang.maxInt();
  var ResultString = "Lang.maxInt()" + "\r\r" + Result;
  Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
Lang.maxInt()
2.47483647
```

Функция *minInt*

Функция возвращает минимальное целое значение.

Пример

```
Extern function minIntTest()
{
    var Result = Lang.minInt();
    var ResultString = "Lang.minInt()" + "\r\r" + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
Lang.minInt()
-2147483648
```

Функция *float*

Функция возвращает `true`, если операции с числами с плавающей точкой поддерживаются на устройстве, и `false` — в противном случае.

Пример

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
    "http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>
    <head>
        <meta http-equiv="Cache-Control" content="max-age=0" fo-
        rua="true"/>
    </head>
    <card id="Test">
        <p align="center">
            Lang.float<br/>
        </p>
        <p>
            <do type="accept">
                <go href="float.wmls#FloatTest()" />
            </do>
            Press OK to test the Lang.float function
        </p>
    </card>
</wml>

extern function FloatTest()
{
    var Result = Lang.float();
```

```
var ResultString = "Lang.float()" + "\r\r" + Result;
Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
Lang.Float()
true
```

Функция *exit*

Функция заканчивает интерпретацию байт-кода WMLScript и возвращает управление вызывающей карте с определенным кодом возврата. Функция может быть использована для осуществления нормального завершения в случаях, когда выполнение байт-кода должно быть сегментировано.

Пример

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
           "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <head>
    <meta http-equiv="Cache-Control" content="max-age=0" fo-
rua="true"/>
  </head>
  <card id="Test">
    <p align="center">
      Lang.exit<br/>
    </p>

    <p>
      <do type="accept">
        <go href="exit.wmls#ExitTest()" />
      </do>
      Press OK to test the Lang.exit function
    </p>
  </card>
</wml>

extern function ExitTest()
{
  //.
  //.
```

```
var Argument = 0;
BadFun(Argument);
Dialogs.alert("This will never display");
}

function BadFun(Argument)
{
    Lang.exit("Error:");
    Dialogs.alert("Neither will this");
}
```

Результат выполнения кода

Lang.exit
Press OK to test the Lang.exit function

Функция *abort*

Функция прерывает интерпретацию байт-кода WMLScript и возвращает управление вызывающему компоненту с описанием возникшей ошибки. Обычно используется для информирования о ненормальном завершении интерпретации кода в тех случаях, когда WMLScript должен быть сегментирован для обнаружения серьезных ошибок процесса выполнения программ. Если аргумент `errordescription` имеет тип `Invalid`, то в результирующей строке появится `"Invalid"`.

Пример

```
Extern function AbortTest()
{
    //.
    //.
    var Argument = 0;
    BadFun(Argument);
    Dialogs.alert("This will never display");
}

function BadFun(Argument)
{
    Lang.abort("Error:");
    Dialogs.alert("Neither will this");
}
```

Результат выполнения кода

Lang.abort
Press OK to test the Lang.abort function

Функция *random*

Функция возвращает положительное целое значение, большее или равное 0, но меньшее или равное данному значению. Возвращаемое значение выбирается случайным образом или на основе псевдослучайной генерации, используя зависящий от процесса выполнения алгоритм или стратегию. Если значение `value` является числом с плавающей точкой, сначала используется функция `Float.int()` для вычисления целого значения.

Пример

```
Extern function RandomTest()
{
    var Argument1 = 10;
    var Result1 = Lang.random(Argument1);
    var Argument2 = -10;
    var Result2 = Lang.random(Argument2);
    var ResultString = "Lang.random()" + "\r\r" +
        "random(10) = " + String.toString(Result1)
        + "\r" + "random(-10) = "
        + String.toString(Result2);
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
Lang.random()
random(10)=8
random(-10)=Invalid
```

Функция *seed*

Функция иницирует псевдослучайную последовательность и возвращает пустую строку. Если `value` больше или равно 0, то данное значение берется для инициализации, в противном случае используется значение, зависящее от операционной системы. Значение функции, большее или равное 0, может быть использовано в повторяющейся последовательности псевдослучайных чисел. Значение, меньшее 0, приводит к созданию неповторяющейся последовательности. Если аргумент имеет тип числа с плавающей точкой, то для приведения его к целому применяется функция `Float.int()`. Если аргумент не является числовым, возвращается значение `Invalid`, а текущее значение `Seed` остается неизменным.

Пример

```
Extern function SeedTest()
{
    var Argument1 = 10;
```

```

var Result1 = Lang.seed(Argument1);
var Argument2 = 5;
var Result2 = Lang.random(Argument2);
var ResultString = "Lang.seed()" + "\r\r" +
    "seed(10) = " + String.toString(Result1);
Dialogs.alert(ResultString);
}

```

Результат выполнения кода

```

Lang.seed()
Seed(10)=

```

Функция *characterSet*

Функция возвращает набор символов, поддерживаемый WMLScript-интерпретатором. Возвращаемое значение является целым MIBEnum-значением, назначаемым IANA для всех кодовых наборов символов.

Пример

```

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
    "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
    <head>
        <meta http-equiv="Cache-Control"
            content="max-age=0" forua="true"/>
    </head>
    <card id="Test">
        <p align="center">
            Lang.characterSet<br/>
        </p>
        <p>
            <do type="accept">
                <go href="charset.wmls#CharSetTest()"/>
            </do>
            Press OK to test the Lang.characterSet function
        </p>
    </card>
</wml>
extern function CharSetTest()
{
    var Result = Lang.characterSet();

```

```
var ResultString = "Lang.characterSet()" + "\r\r" + Result;  
Dialogs.alert(ResultString);  
}
```

Результат выполнения кода

```
Lang.characterSet()  
2251
```

Библиотека *Float*

Функция *int*

Функция возвращает целую часть аргумента. Если аргумент уже является целым, результат будет значением самого аргумента.

Пример

```
extern function IntTest()  
{  
    var Argument1 = 10.5;  
    var Result1 = Float.int(Argument1);  
    var Argument2 = -5.5;  
    var Result2 = Float.int(Argument2);  
    var ResultString = "Float.int()" + "\r\r" +  
        "int(10.5) = " + Result1 + "\r" +  
        "int(-5.5) = " + Result2;  
    Dialogs.alert(ResultString);  
}
```

Результат выполнения кода

```
Float.int()  
Int(10.5)=10  
Int(-5.5)=-5
```

Функция *floor*

Функция возвращает минимальное целое, не большее значения аргумента. Если аргумент уже является целым, результат будет значением самого аргумента.

Пример

```
extern function FloorTest()  
{  
    var Argument1 = 10.5;
```

```
var Result1 = Float.floor(Argument1);
var Argument2 = -5.5;
var Result2 = Float.floor(Argument2);
var Argument3 = 300;
var Result3 = Float.floor(Argument3);
var ResultString = "Float.floor()" + "\r\r" +
    "floor(10.5) = " + Result1 + "\r" +
    "floor(-5.5) = " + Result2 + "\r" +
    "floor(300) = " + Result3;
Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
Float.floor()
Floor(10.5)=10
Floor(-5.5)=-6
Floor(300)=300
```

Функция *ceil*

Функция возвращает максимальное целое, не меньшее значения аргумента. Если аргумент уже является целым, результат будет значением самого аргумента.

Пример

```
extern function CeilTest()
{
    var Argument1 = 10.5;
    var Result1 = Float.ceil(Argument1);
    var Argument2 = -5.5;
    var Result2 = Float.ceil(Argument2);
    var ResultString = "Float.ceil()" + "\r\r" +
        "ceil(10.5) = " + Result1 + "\r" +
        "ceil(-5.5) = " + Result2;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
Float.ceil()
Ceil(10.5)=11
Ceil(-5.5)=-5
```

Функция *pow*

Функция возвращает машинно-зависимую аппроксимацию возведения *value1* в степень *value2*. Если *value1* является отрицательным числом, то *value2* должно быть целым.

Пример

```
extern function PowTest()
{
    var Argument1 = 3;
    var Argument2 = 2;
    var Result1 = Float.pow(Argument1, Argument2);
    var Argument3 = 16;
    var Argument4 = .5;
    var Result2 = Float.pow(Argument3, Argument4);
    var ResultString = "Float.pow()" + "\r\r" +
        "pow(3, 2) = " + Result1 + "\r" +
        "pow(16, .5) = " + Result2;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
Float.pow()
Pow(3,2)=9.000000e+000
Pow(16,5)=4.000000e+000
```

Функция *round*

Функция возвращает числовое значение, ближайшее к значению аргумента, округленное по правилам математики. Если два целых значения одинаково близки, то результатом будет большее значение. Если аргумент уже является целым числом, то результатом будет значение аргумента.

Пример

```
extern function RoundTest()
{
    var Argument1 = 5.4;
    var Result1 = Float.round(Argument1);
    var Argument2 = 5.5;
    var Result2 = Float.round(Argument2);
    var Argument3 = 5.6;
    var Result3 = Float.round(Argument3);
    var ResultString = "Float.round()" + "\r\r" +
```

```
        "round(5.4) = " + Result1 + "\r" +  
        "round(5.5) = " + Result2 + "\r" +  
        "round(5.6) = " + Result3;  
    Dialogs.alert (ResultString);  
}
```

Результат выполнения кода

```
Float.round()  
round(5.4)=5  
round(5.5)=6  
round(5.5)=6
```

Функция *sqrt*

Функция возвращает машинно-зависимую аппроксимацию квадратного корня аргумента.

Пример

```
extern function SqrtTest()  
{  
    var Argument1 = 2;  
    var Result1 = Float.sqrt(Argument1);  
    var Argument2 = -9;  
    var Result2 = Float.sqrt(Argument2);  
    var ResultString = "Float.sqrt() + "\r\r" +  
        "sqrt(2) = " + String.toString(Result1) + "\r" +  
        "sqrt(-9) = " + String.toString(Result2);  
    Dialogs.alert (ResultString);  
}
```

Результат выполнения кода

```
Float.sqrt()  
sqrt(z)=1.414214e+000  
sqrt(-9)=Invalid
```

Функция *minFloat*

Функция возвращает наименьшее ненулевое значение числа в представлении с плавающей точкой, определенное в документе IEEE754, для чисел с одинарной точностью.

Пример

```
extern function minFloatTest()  
{
```

```
var Result = Float.minFloat();
var ResultString = "Float.minFloat()" + "\r\r" + Result;
Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
Float.minFloat()
1.175494e-038
```

Библиотека *String*

Функция *maxFloat*

Функция возвращает наибольшее ненулевое значение числа в представлении с плавающей точкой, определенное в документе IEEE754, для чисел с одинарной точностью.

Пример

```
extern function maxFloatTest()
{
    var Result = Float.maxFloat();
    var ResultString = "Float.maxFloat()" + "\r\r" + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
Float.maxFloat()
3.402824e+038
```

Функция *length*

Функция возвращает длину строки символов (число символов в строке).

Пример

```
Extern function LengthTest()
{
    var Argument = "Champion Matt";
    var Result = String.length(Argument);
    var ResultString = "String.length()" + "\r\r" +
        "length(\"Champion Matt\") = " + Result;

    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
String.Length()  
Length("Champion Matt")=13
```

Функция *isEmpty*

Функция возвращает булево `true`, если длина строки равна 0, и `false` — в противном случае.

Пример

```
extern function IsEmptyTest()  
{  
    var Argument1 = "Matt";  
    var Result1 = String.isEmpty(Argument1);  
    var Argument2 = "";  
    var Result2 = String.isEmpty(Argument2);  
    var ResultString = "String.isEmpty()" + "\r\r" +  
        "isEmpty(\"Matt\") = " + Result1 + "\r" +  
        "isEmpty(\"\") = " + Result2;  
    Dialogs.alert(ResultString);  
}
```

Результат выполнения кода

```
String.isEmpty()  
IsEmpty("Matt")=false  
IsEmpty(" ")=true
```

Функция *charAt*

Функция возвращает новую строку единичной длины, содержащую символ, стоящий в позиции `index` заданной строки. Если `index` имеет тип числа с плавающей точкой, то сначала используется функция `Float.int()` для вычисления целочисленного индекса.

Пример

```
extern function charAtTest()  
{  
    var Argument1 = "Phil";  
    var Argument2 = 1;  
    var Result1 = String.charAt(Argument1, Argument2);  
    var Argument3 = 2;  
    var Result2 = String.charAt(Argument1, Argument3);
```

```
var ResultString = "String.charAt()" + "\r\r" +  
    "String.charAt(Phil, 2) = " + Result1 + "\r" +  
    "String.charAt(Phil, 3) = " + Result2;  
Dialogs.alert(ResultString);  
}
```

Результат выполнения кода

```
String.charAt()  
String.charAt(Phil,2)=h  
String.charAt(Phil,3)=i
```

Функция *subString*

Функция возвращает подстроку заданной строки. Подстрока начинается с позиции `startIndex` заданной строки, а ее длина имеет `length` символов. Если `startIndex` меньше 0, то в качестве этого аргумента используется 0. Если значение `length` больше оставшейся части строки, то `length` заменяется на число оставшихся символов в строке. В случае, если `startIndex` или `length` имеют тип числа с плавающей точкой, то для вычисления целочисленных значений используется функция `Float.int()`.

Пример

```
extern function SubStringTest()  
{  
    var List = "Matt Krissy Tommy";  
    var startChar = 5;  
    var Length = 6;  
    var Result = String.subString(List, startChar, Length);  
    var ResultString = "String.subString()" + "\r\r" + "The sub-  
string is: " + Result;  
    Dialogs.alert(ResultString);  
}
```

Результат выполнения кода

```
String.sub.String()  
The substring is: Krissy
```

Функция *find*

Функция возвращает индекс первого символа в строке `string`, с которого начинается совпадение с подстрокой `subString`. Если совпадения нет, то возвращается `-1`. Две строки совпадают, если они идентичны. Сравнение выполняется независимо от регистра.

Пример

```
extern function FindTest()
{
    var Argument = "abcABC123";
    var LookFor = "AB";
    var Result = String.find(Argument, LookFor);
    var ResultString = "String.find()" + "\r\r" +
        "The Index of AB is " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
String.find()
The Index of AB is 3
```

Функция *replace*

Функция возвращает новую строку, в которой замещены все вхождения подстроки `oldSubString` на подстроку `newSubString`. Символы с несколькими возможными представлениями совпадают, только если они имеют одинаковое представление в обеих строках. Сравнение выполняется независимо от регистра.

Пример

```
extern function ReplaceTest()
{
    var List = "Matt Krissy Tommy";
    var toReplace = "Tommy";
    var replaceWith = "Summer";
    var Result = String.replace(List, toReplace, replaceWith);
    var ResultString = "String.replace()" + "\r\r" +
        "The new string is: " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
String.replace()
The new string is: Matt Krissy
Summer
```

Функция *elements*

Функция возвращает число элементов в данной строке, разделенных указанным разделителем `separator`. Пустая строка ("") является корректным

аргументом (таким образом, функция никогда не возвращает значение, меньшее или равное 0).

Пример

```
extern function elementsTest()
{
    var Argument = "Matt Tommy Krissy Rosey";
    var Separator = " ";
    var Result = String.elements(Argument, Separator);
    var ResultString = "String.elements()" + "\r\r" +
        "The number of elements is " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
String.elements()
The number of elements is 4
```

Функция *elementAt*

Функция ищет строку для элемента под номером *index*, учитывая, что между элементами существует разделитель *separator*, и возвращает соответствующий элемент. Если *index* меньше 0, то возвращается первый элемент. Если *index* больше количества элементов в строке, то возвращается последний элемент. Если строка является пустой, то возвращается пустая строка. Если *index* имеет тип числа с плавающей точкой, то для получения целочисленного значения используется функция *Float.int()*.

Пример

```
extern function elementAtTest()
{
    var Argument = "Matt Tommy Krissy Rosey";
    var Index=2;
    var Separator = " ";
    var Result = String.elementAt(Argument, Index, Separator);
    var ResultString = "String.elementAt()" + "\r\r" +
        "Third element is " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
String.elementAt()
Third element is Krissy
```

Функция *removeAt*

Функция возвращает новую строку, в которой элемент с номером `index` и соответствующий разделитель (если таковой имеется) удалены из заданной строки `string`. Если `index` меньше 0, то удаляется первый элемент. Если `index` больше числа элементов в строке, то удаляется последний элемент. Если строка пустая, то возвращается новая пустая строка. Если `index` имеет тип числа с плавающей точкой, то для получения целочисленного значения используется функция `Float.int()`.

Пример

```
Extern function RemoveAtTest()
{
    var List = "Matt Krissy Tommy";
    var Index = 1;
    var Separator = " ";
    var Result = String.removeAt(List, Index, Separator);
    var ResultString = "String.removeAt()" + "\r\n" +
        "The new string is: " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
String.removeAt()
The new string is: Matt Tommy
```

Функция *replaceAt*

Функция возвращает строку, в которой текущий элемент в специфицированной аргументом `index` позиции заменен на значение аргумента `element`. Если `index` меньше 0, то заменяется первый элемент. Если `index` больше числа элементов в строке, то заменяется последний элемент. Если строка пустая, то замещается пустая строка. Если `index` имеет тип числа с плавающей точкой, то для получения целочисленного значения используется функция `Float.int()`.

Пример

```
extern function ReplaceAtTest()
{
    var List = "Matt Krissy Tommy";
    var toReplace = 2;
    var Separator = " ";
    var replaceWith = "Summer";
```

```
var Result = String.replaceAt(List, replaceWith, toReplace,
    Separator);
var ResultString = "String.replaceAt()" + "\r\r" +
    "The new string is: " + Result;
Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
String.ReplaceAt()
The new string is: Matt Krissy
Summer
```

Функция *insertAt*

Функция возвращает строку с элементом *element* и разделителем *separator*, вставленным перед специфицированной аргументом *index* позицией строки *string*. Если *index* меньше 0, то 0 используется как *index*. Если *index* больше числа элементов в строке, то элемент добавляется в конец строки *string*. Если строка пуста, функция возвращает новую строку с новым элементом *element*. Если *index* имеет тип числа с плавающей точкой, то для получения целочисленного значения используется функция `Float.int()`.

Пример

```
extern function InsertAtTest()
{
    var List = "Matt Tommy";
    var toInsert = "Krissy";
    var Index = 1;
    var Separator = " ";
    var Result = String.insertAt(List, toInsert, Index, Separator);
    var ResultString = "String.insertAt()" + "\r\r" +
        "The new string is: " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
String.insertAt()
The new string is: Matt Krissy Tommy
```

Функция *squeeze*

Функция возвращает строку, в которой все последующие серии пробелов замещаются на строку с одним пробелом-разделителем между словами.

Пример

```
extern function SqueezeTest()
{
    var List = "Matt      Tommy";
    var Result = String.squeeze(List);
    var ResultString = "String.squeeze()" + "\r\r" +
        "The new string is: " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
String.squeeze()
The new string is: Matt Tommy
```

Функция *trim*

Функция возвращает строку с удаленными ведущими и завершающими пробелами.

Пример

```
extern function TrimTest()
{
    var Argument = "      Matt      ";
    var Result = String.trim(Argument);
    var ResultString = "String.trim()" + "\r\r" +
        "The new string is: " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
String.trim()
The new string is: Matt
```

Функция *compare*

Возвращаемое функцией значение указывает на лексикографическое соотношение строки `string1` и строки `string2`. Это соотношение основано на сопоставлении кодов символов в национальной таблице символов. Возвращаемое значение равно `-1`, если `string1` меньше, чем `string2`; `0`, если `string1` идентична `string2`; `1`, когда `string1` больше `string2`.

Пример

```
extern function CompareTest()
{
```

```

var Argument1 = "a";
var Argument2 = "A";
var Result1 = String.compare(Argument1, Argument2);
var Result2 = String.compare(Argument2, Argument1);
var ResultString = "String.compare()" + "\r\r" +
    "String.compare(a, A) = " + Result1 + "\r" +
    "String.compare(A, a) = " + Result2;
    Dialogs.alert(ResultString);
}

```

Результат выполнения кода

```

String.compare()
String.compare(a,A)=-1
String.compare(A,a)=1

```

Функция *toString*

Функция возвращает строковое представление данного числа. Функция выполняет точно такое же преобразование, что и автоматическое преобразование типов языка WMLScript для типов булево, целое и с плавающей точкой в строковое представление, за исключением того, что значение Invalid возвращает строку "Invalid".

Пример

```

extern function ToStringTest()
{
    var Argument1 = 3.14;
    var Result1 = String.toString(Argument1);
    var Argument2 = 1 / 0;
    var Result2 = String.toString(Argument2);
    var Argument3 = true;
    var Result3 = String.toString(Argument3);
    var ResultString = "String.toString()" + "\r\r" + "toString(3.14) = "
+ Result1 + "\r" + "toString(1/0) = " + Result2 + "\r" +
    "toString(true) = " + Result3;
    Dialogs.alert(ResultString);
}

```

Результат выполнения кода

```

String.toString()
ToString(3.14)=3.140000e+000

```

```
ToString(1/0)=Invalid
```

```
ToString(true)=true
```

Функция *format*

Функция преобразует данное значение аргумента *value* в строку, используя форматирование, определенное аргументом-строкой *format*. Строка *format* должна содержать только один описатель формата, который может находиться в любом месте строки. В случае наличия в строке более одного описания формата, выполняется только первый слева описатель, а остальные замещаются пустой строкой. Описатель *format* имеет следующую форму:

```
%[width].[precision]type
```

Аргумент *width* является неотрицательным десятичным целым, управляющим минимальным печатаемым числом символов. Если число символов в выводимой строке меньше значения этого аргумента, в нее слева добавляются символы пробелов. Аргумент *width* никогда не приводит к усечению значения *value*. Если число символов в выводимой строке больше заданного значением *width* или, если значение *width* не задано, печатаются все символы значения параметра *value* (в соответствии с аргументом, указывающим точность *precision*). Аргумент *precision* задает неотрицательное десятичное целое, следующее за точкой (.), которое может быть использовано для установки точности выводимого значения *value*. Интерпретация этого значения зависит от типа:

- ❑ **d** (целое) описывает минимальное количество цифр, которые должны быть выведены. Если число цифр в значении *value* меньше, чем точность, заданная в *precision*, выводимое значение смещается влево с нулями. Значение не усекается, когда число цифр превышает указанную точность. Кратность точности по умолчанию равна 1. Если точность описана как 0, и значение, которое должно быть преобразовано, также равно 0, то результатом будет пустая строка;
- ❑ **f** (плавающая точка) описывает число цифр после десятичной точки. Если появляется десятичная точка, то существует, по меньшей мере, одна цифра до нее. Значение округляется до подходящего числа цифр. По умолчанию точность равна 6. Если точность равна 0 или если точка (.) появляется без последующего числа, то десятичная точка не печатается;
- ❑ **s** (строка) описывает максимальное число символов, которые должны появляться. По умолчанию все символы печатаются.

В отличие от аргумента *width* аргумент *precision* может приводить либо к усечению выводимого значения, либо к округлению значения числа с плавающей точкой.

Аргумент *type* — единственный аргумент, который обязателен в форме описания формата; он появляется после любого необязательного аргумента формата полей. Символ *type* указывает, будет ли данное значение *value* интерпретироваться как целое, число с плавающей точкой или строка. Если

значение аргумента `value` имеет тип, отличный от описанного в форме формата, он преобразуется в соответствии с правилами автоматического преобразования типов стандарта WMLScript с тем дополнением, что если `value` имеет тип числа с плавающей точкой, а в качестве `type` указано `d`, то для преобразования значения `value` вызывается функция `Float.int()`.

Поддерживаемые типы аргумента `type`:

- ❑ `d` (целое). Выводимое значение имеет форму `[-]dddd`, где `dddd` — одна или более десятичных цифр;
- ❑ `f` (плавающая точка). Выводимое значение имеет форму `[-]dddd.dddd`, где `dddd` — одна или более десятичных цифр. Число цифр до десятичной точки зависит от значения числа, а количество цифр после десятичной точки — от требуемой точности. Когда число цифр после точки в значении `value` меньше, чем точность, указанная в `precision`, должен появиться 0, который будет помещен в заполненные колонки. Например, результат `String.format("%2.5f", 1.2)` будет `"1.20000"`.
- ❑ `s` (строка). Символы печатаются до конца строки или до тех пор, пока не будет достигнута указанная в формате точность. В том случае, когда значение `width` больше точности, величина `width` игнорируется. Символ процента (%) может быть включен в формат с предшествующим дополнительным символом процента (%%).

Пример

```
extern function FormatTest()
{
    var Format = "%6.1f";
    var Argument = 10.25;
    var Result = String.format(Format, Argument);
    var ResultString = "String.format()" + "\r\r" +
        "The result is: " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
String.format()
The results is: 10.3
```

Библиотека *URL*

Функция *isValid*

Функция возвращает `true`, если данный URL имеет правильный формат, в противном случае возвращает `false`. Поддерживаются как абсолютная, так

и относительная адресация URL. Преобразование относительных URL в абсолютные не выполняется.

Пример

```
Extern function IsValidTest()
{
    var Argument1 = "waplib.com";
    var Result1 = URL.isValid(Argument1);
    var Argument2 = "";
    var Result2 = URL.isValid(Argument2);
    var ResultString = "URL.isValid()" + "\r\r" +
        "isValid(waplib.com) = " + Result1 + "\r" +
        "isValid() = " + Result2;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
URL.isValid()
IsValid(waplib.com)=true
IsValid()=false
```

Функция *getScheme*

Функция возвращает схему, использованную в URL. Поддерживаются как абсолютная, так и относительная адресация URL. Относительные URL не преобразуются в абсолютные.

Пример

```
extern function GetSchemeTest()
{
    var Result = URL.getScheme("http://waplib.com/test;1;2;3");
    var ResultString = "URL.getScheme()" + "\r\r" +
        "The scheme is " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
URL.getScheme()
The scheme is http
```

Функция *getHost*

Функция возвращает имя хоста, заданного в URL. Поддерживаются как абсолютная, так и относительная адресация URL. Преобразование относи-

тельных URL в абсолютные не выполняется. Если часть хоста в URL не определена, функция возвращает пустую строку.

Пример

```
extern function GetHostTest()
{
    var Result = URL.getHost("http://waplib.com/test#abcd");
    var ResultString = "URL.getHost()" + "\r\r" +
        "The host is " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
URL.getHost()
The host is waplib.com
```

Функция *getPort*

Функция возвращает номер порта, заданного в URL. Если порт не описан, то возвращается пустая строка. Поддерживаются как абсолютная, так и относительная адресация. Преобразование относительных URL в абсолютные не выполняется.

Пример

```
extern function GetPortTest()
{
    var Result = URL.getPort("http://waplib.com:8080/test;1;2;3");
    var ResultString = "URL.getPort()" + "\r\r" +
        "The port is " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
URL.getPort()
The port is 8080
```

Функция *getPath*

Функция возвращает путь, заданный в URL. Параметры, описанные для указания каждой части сегмента пути, не возвращаются. Поддерживаются как абсолютная, так и относительная адресация URL. Преобразование относительных URL в абсолютные не выполняется.

Пример

```
extern function GetPathTest()
{
    var Result = URL.getPath("http://waplib.com/test;1;2;3");
    var ResultString = "URL.getPath()" + "\r\r" +
        "The path is " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
URL.getPath()
The path is /test
```

Функция *getParameter*

Функция возвращает параметры, используемые для описания последнего сегмента пути URL. Если параметр не указан, то возвращается пустая строка. Поддерживаются как абсолютная, так и относительная адресация URL. Преобразование относительных URL в абсолютные не выполняется.

Пример

```
Extern function GetParametersTest()
{
    var Result = URL.getParameters("http://waplib.com/test;1;2;3");
    var ResultString = "URL.getParameters()" + "\r\r" +
        "The parameters are " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
URL.getParameters()
The parameters are 1,2,3
```

Функция *getQuery*

Функция возвращает часть запроса, описанную в URL. Если не указано, то возвращается пустая строка. Поддерживаются как абсолютная, так и относительная адресация URL. Преобразование относительных URL в абсолютные не выполняется.

Пример

```
Extern function GetQueryTest()
{
    var Result = URL.getQuery ("http://waplib.com:8080/test;1;2;3?a=1&b=2");
}
```

```
var ResultString = "URL.getQuery()" + "\r\r" +  
    "The query is " + Result;  
Dialogs.alert(ResultString);  
}
```

Результат выполнения кода

```
URL.getQuery()  
The query is a=1 & b=2
```

Функция *getFragment*

Функция возвращает фрагмент, заданный в URL. Если фрагмент не задан, то возвращается пустая строка. Поддерживаются как абсолютная, так и относительная адресация URL. Преобразование относительных URL в абсолютные не выполняется.

Пример

```
extern function GetFragmentTest()  
{  
    var Result = URL.getFragment("http://waplib.com/test#abcd");  
    var ResultString = "URL.getFragment()" + "\r\r" +  
        "The fragment is " + Result;  
    Dialogs.alert(ResultString);  
}
```

Результат выполнения кода

```
URL.getFragment()  
The fragment is abcd
```

Функция *getBase*

Функция возвращает абсолютный URL (без фрагментов) текущей программной единицы WMLScript.

Пример

```
extern function GetBaseTest()  
{  
    var Result = URL.getBase();  
    var ResultString = "URL.getBase()" + "\r\r" +  
        "getBase() = " + Result;  
    Dialogs.alert(ResultString);  
}
```

Результат выполнения кода

```
URL.getBase()
```

```
GetBase()=HTTP://127.0.0.1/wapps/getbase.wmls
```

Функция *getReferer*

Функция возвращает наименьший относительный URL (по отношению к базовому URL текущей скомпилированной единицы) касательно к ресурсу, который называется текущей единицей компиляции. Если текущая единица компиляции не имеет ссылки, то возвращается пустая строка.

Пример

```
extern function GetRefererTest()
{
    var Result = URL.getReferer();
    var ResultString = "URL.getReferer()" + "\r\r" +
        "URL.getReferer = " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
URL.getReferer()
```

```
URL.getReferer=getreferer.wml
```

Функция *resolve*

Функция возвращает абсолютный URL из данного базового `baseUrl` и вложенного `embeddedUrl` в соответствии с правилами, заданными в документе RFC2396. Прежде чем применить правила RFC2396, проверяется `baseUrl`. Если компонент пути в `baseUrl` является пустой строкой, то вставляется одиночный символ (/) и предполагается, что он является частью пути. Если `embeddedUrl` уже является абсолютным URL, то функция возвращает его без модификации.

Пример

```
extern function ResolveTest()
{
    var Argument1 = "http://waplib.com";
    var Argument2 = "Demos.wml";
    var Result = URL.resolve(Argument1, Argument2);
    var ResultString = "URL.resolve()" + "\r\r" +
        "The absolute URL is " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
URL.resolve()
```

```
The absolute URL is APLIB.COM/Damos.wml
```

Функция *escapeString*

Функция вычисляет новую версию значения строки *string*, в которой специальные символы, описанные в документе RFC2396, заменяются шестнадцатеричной *escape*-последовательностью. Эти символы таковы:

- ☐ управляющие символы: символы в кодировке US ASCII 00-1F и 7F;
- ☐ пробел: US ASCII код 20;
- ☐ зарезервированные символы: ; / ? : @ & = + \$;
- ☐ спецсимволы: { } | \ ^ [] ` ;
- ☐ ограничители: < > # %;
- ☐ не US ASCII: символы с кодами 8F—FF.

Строка обрабатывается следующим образом: разбор URL не производится, не US ASCII символы должны быть преобразованы в символы с кодами национальных кодировок

Пример

```
extern function EscapeStringTest()
{
    var Argument = "@&#";
    var Result = URL.escapeString(Argument);
    var ResultString = "URL.escapeString()" + "\r\r" +
        "escapeString(@&#) = " + Result;
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
URL.escapeString()
```

```
EscapeString(@&#)=%40%26%23
```

Функция *unescapeString*

Функция вычисляет новую версию значения строки *string*, в которой каждая *escape*-последовательность заменяется символом, который она представляет (см. также *URL.escapeString()*).

Пример

```
extern function UnescapeStringTest()
{
```

```
var Argument = "%40%26%23";
var Result = URL.unescapeString(Argument);
var ResultString = "URL.unescapeString()" + "\r\r" +
    "unescapeString(%40%26%23) = " + Result;
Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
URL.unescapeString()
UnescapeString(%40%26%23)=@&#
```

Функция *loadString*

Функция возвращает содержание, заданное для данного абсолютного URL и типа `contentType`. Значение `contentType` является ошибочным, если оно не следует нижеуказанным правилам.

- ❑ Может быть задан только один тип содержания. Вся строка должна соответствовать только одному типу содержания, и никакие ведущие символы или последующие пробелы не допускаются.
- ❑ Тип должен быть строковым, но подтип может быть чем угодно. Таким образом, префикс должен выглядеть как `text/`.

Поведение функции следующее.

- ❑ Загружается содержание, соответствующее данному URL и типу содержания. Остальные атрибуты нужны для загрузки содержания, задаваемого установками по умолчанию для конкретной реализации WML-агента.
- ❑ Если загрузка выполнена успешно, а возвращаемое содержание совпадает с заданным, то содержание преобразуется в строку и возвращается.
- ❑ Если загрузка неудачна или возвращаемое содержание не соответствует заданному типу содержания, то возвращается код ошибки.

Библиотека *WMLBrowser*

Функция *getVar*

Функция возвращает значение переменной с данным именем `name` в текущем браузере. Результат является пустой строкой, если данная переменная не существует. Имя переменной должно удовлетворять стандартным правилам для WML.

Пример

```
extern function GetVarTest()
{
```

```

var Argument = "Name";
var Result = WMLBrowser.getVar(Argument);
var ResultString = "WMLBrowser.getVar()" + "\r\r" +
    "The name is " + Result;
Dialogs.alert(ResultString);
}

```

Результат выполнения кода

```

WMLBrowser.getVar()
The name is Matt

```

Функция *setVar*

Функция возвращает `true`, если содержание значения `value` успешно установлено для переменной `name` в текущей версии браузера, в противном случае — `false`. Имя переменной и ее значение должны удовлетворять Примеру WML.

Пример

```

extern function SetVarTest()
{
    var Argument1 = "Name";
    var Argument2 = "Krissy";
    var Result = WMLBrowser.setVar(Argument1, Argument2);
    var ResultString = "WMLBrowser.setVar()" + "\r\r" +
        "Setting Name to Krissy";
    Dialogs.alert(ResultString);
}

```

Результат выполнения кода

```

WMLBrowser.setVar()
Setting Name to Krissy

```

Функция *go*

Функция задает содержание с адресом, указанным в URL, которое должно быть загружено в браузер. Эта функция имеет тот же самый смысл, что и конструкция `go()` в языке WML (более подробное описание дано в *главе 2*). Это содержание загружается только после того, как браузер восстановит управление, возвратив его из WMLScript-интерпретатора, а вызов WMLScript завершится. Когда WML-браузер загружает содержание, ссылающимся URI (Uniform Resource Identifier — универсальный идентификатор ресурса) является URI текущей карты. Если используется относительный URI в качестве URL, то WML-браузер должен разрешить его, используя URI текущей карты.

Содержание не загружается, если URL представлен пустой строкой (""). Функции `go()` и `prev()` являются взаимоотменяемыми. Обе могут вызываться многократно, прежде чем вернуть управление в WML-браузер. Но только последняя установка будет активной в момент такого возврата. Например, если последний вызов `go()` устанавливает URL на пустую строку (""), то все предыдущие запросы `go()` и `prev()` завершаются. Вызов функции `Lang.Abort()` с кодом фатальной ошибки завершает любой из запросов `go()`. При этом функция возвращает пустую строку.

Пример

```
Extern function GoTest()
{
    var Argument = "getvar.wml";
    var Result = WMLBrowser.go(Argument);
    var ResultString = "WMLBrowser.go()" + "\r\r" +
        "Ready to start " + "\r" + "getvar.wml ";
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
WMLBrowser.go()
Ready to start
Getvar.wml
```

Функция *prev*

Функция указывает WML-браузеру перевод на предыдущую карту. Функция имеет ту же семантику, что и конструкция `prev` в языке WML. Предыдущая карта загружается только после того, как WML-браузер получит управление от интерпретатора WMLScript.

Пример

```
extern function PrevTest()
{
    var Result = WMLBrowser.prev();

    var ResultString = "WMLBrowser.prev()" + "\r\r" +
        "Ready to go back one deck ";
    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
WMLBrowser.prev()
Ready to go back one deck
```

Функция *newContext*

Вызов этой функции сбрасывает значения всех переменных из связанного WML-содержания и удаляет стек истории навигации за исключением текущей карты прежде, чем вернуть выполнение вызывающей сущности. Функция возвращает пустую строку.

Функция *GetCurrentCard*

Функция возвращает наименьший относительный URL (по отношению к базе текущей единицы компиляции), показывая карту, обрабатываемую в настоящее время в WML-браузере. Функция возвращает абсолютный URL в случае, когда WML-колода, содержащая карту, не имеет той же самой базы, что текущая единица компиляции.

Пример

```
extern function GetCurrentCardTest()
{
    var Result = WMLBrowser.getCurrentCard();

    var ResultString = "WMLBrowser.getCurrentCard()" + "\r\r" +
        "The current card is " + Result;

    Dialogs.alert(ResultString);
}
```

Результат выполнения кода

```
WMLBrowser.getCurrentCard()
The current card is getcard.wml
```

Функция *refresh*

Функция предписывает WML-браузеру обновить пользовательский интерфейс текущего содержания. Исполнение, которое поддерживает эту операцию, должно выполнять шаги, определенные в языке WML, за исключением рестарта неактивного таймера. Функция не должна восстанавливать таймер. Функция обязана выполнять блокировку до того момента, пока не завершатся все шаги операции. Действия обновления должны быть применимы к текущей карте WML. Если текущая карта не была превращена до этого вызова (например, был вызван скрипт по событию *onenterforward*, см. приложение 3), функция должна выполнить операцию превращения для карты.

Функция возвращает *Invalid*, если исполняющее устройство не поддерживает функцию немедленного обновления. В противном случае функция возвращает пустую строку, если обновление завершилось успешно, и непустую

строку, если, неудачно (например, не может обновить графику). Содержание строки зависит от исполнения. Функция должна вернуть краткое описание ошибки.

Замечание

Если текущее исполнение не поддерживает обновление, WML-агент все равно должен обновить карту при возврате управления агенту WML.

Библиотека *Dialogs*

Функция *prompt*

Функция показывает сообщение `message` и запрашивает ввод пользователя. Параметр `defaultInput` содержит начальное содержание пользовательского ввода. Возвращает результат ввода пользователя.

Пример

```
extern function PromptTest()
{
    var Message = "Dialogs.prompt()" + "\r\r" + "Phone Number:";
    var PhoneNo = "555-1212";
    Dialogs.prompt(Message, PhoneNo);
}
```

Результат выполнения кода

```
Dialogs.prompt()
Phone Number: 555-1212
```

Функция *confirm*

Функция показывает данное сообщение `message` и две альтернативы — `OK` и `Cancel`. Ожидает реакции от пользователя, возвращает `true` для `OK` или `false` для `Cancel`.

Пример

```
extern function ConfirmTest()
{
    var Message = "Dialogs.confirm()" + "\r\r" + "Are you sure?";
    var OK = "Yes";
    var Cancel = "No";
    Dialogs.confirm(Message, OK, Cancel);
}
```

Результат выполнения кода

```
Dialogs.confirm()
```

```
Are you sure?
```

Функция *alert*

Функция показывает пользователю сообщение `message` и ожидает подтверждения. Возвращает пустую строку.

Пример

```
extern function AlertTest()  
{  
    var ResultString = "Dialogs.alert()" + "\r\r" + "This is an alert!";  
    Dialogs.alert(ResultString);  
}
```

Результат выполнения кода

```
Dialogs.alert()
```

```
This is an alert!
```

Библиотеки *WTA*

Библиотека *WTAPublic*

Функция *makeCall*

Функция инициирует голосовой вызов мобильного терминала; вызов должен быть завершен с использованием стандартного MMI.

Параметр `number` описывает номер вызываемого абонента и должен быть правильным телефонным номером. Функция возвращает:

- ☐ пустую строку, если звонок произведен успешно;
- ☐ код ошибки при ошибке в процессе дозвона;
- ☐ `Invalid`, если функция выполняется неудачно.

Функция *sendDTMF*

Функция посылает DTMF-последовательность через голосовой вызов, только что созданный с использованием `WTAPublic.makeCall` или `wtai://wp/mc`. Это блокирующая функция. WTA-события не генерируются в результате прямого или косвенного вызова данной функции.

Параметр `dtmf` задает посылаемую DTMF-последовательность и должен быть корректной строкой вызова. Функция возвращает:

- ☐ пустую строку, если DTMF-последовательность послана успешно;
- ☐ код ошибки, который указывает на ошибку, при отправке DTMF-последовательности;
- ☐ `Invalid`, если функция завершается аварийно.

Функция ***addPBEntry***

Функция пишет новую строку в телефонную книжку.

Параметр `number` задает телефонный номер, связанный со строкой. Параметр `name` определяет имя, связанное со строкой, и может быть пустой строкой. Функция возвращает:

- ☐ пустую строку, если телефонная книжка заполнена успешно;
- ☐ код ошибки при ее возникновении;
- ☐ `Invalid`, когда функция завершается аварийно.

Библиотека ***WTAVoiceCall***

Функция ***setup***

Функция иницирует мобильный вызов. Последующие события WTA сигнализируют о процессе выполнения вызова. Параметр `number` задает получателя вызова и должен быть правильным телефонным номером. Параметр `mode` указывает, как вызов должен быть обработан, если WTA-агент завершает свою работу.

Функция возвращает управление вызовом при успешном завершении операции или `Invalid` в противном случае.

Функция ***accept***

Функция принимает входящий вызов, который ожидается в ответ. Параметр `callHandle` (логический идентификатор звонка) задает вызов, на который нужно реагировать. Параметр `mode` указывает режим, которым вызов должен быть обработан WTA-агентом, и как он должен быть завершен. Функция возвращает пустую строку при успехе или `Invalid` при неудаче.

Функция ***release***

Функция освобождает голосовой вызов. Параметр `callHandle` (логический идентификатор звонка) указывает, какой вызов должен быть освобожден. Функция возвращает пустую строку при успехе или `Invalid` при неудаче.

Функция ***sendDTMF***

Функция посылает DTMF-последовательность по голосовому вызову. Параметр `callHandle` (логический идентификатор звонка) указывает, какой вызов должен получить DTMF. Параметр `dtmf` описывает посылаемую DTMF-последовательность. Функция возвращает пустую строку при успехе или `Invalid` при неудаче.

Функция ***callStatus***

Функция извлекает информацию о конкретном голосовом вызове. Параметр `callHandle` задает логический идентификатор звонка. Сама функция возвращает значение, которое зависит от параметра `field`. Значением этого параметра может быть любая строка, однако если параметр `field` соответствует одному из имен полей, соответствующая информация должна быть возвращена. Если параметр `field` не опознан, не поддерживается или по каким-то причинам недоступен, возвращается пустая строка.

Функция ***list***

Функция возвращает `callHandle` — логический идентификатор звонка, который мог бы управляться посредством WTA-содержания, вызываемого этой функцией. Эта функция вызывает информацию обо всех таких вызовах.

Необходимо отметить, что функция может возвращать вызов, который более не активен, например, чтобы позволить услуге WTA определить длительность вызова после его завершения. Параметр `returnFirst` указывает на логический идентификатор вызова, который должен быть возвращен.

Библиотека ***WTANetText***

Функция ***send***

Функция посылает сетевое сообщение. Параметр `address` задает пункт назначения для сообщения и должен быть телефонным номером. Параметр `text` определяет посылаемый текст.

Функция ***list***

Функция возвращает логический идентификатор текущего сообщения. Функция вызывается повторно для извлечения информации обо всех существующих сообщениях. Эта функция не изменяет значение поля `read` в сообщении. Параметр `returnFirst` указывает логический идентификатор сообщения, который нужно вернуть.

Функция *remove*

Функция удаляет входящие и исходящие сетевые сообщения из устройства. Параметр `msgHandle` указывает, какое сообщение должно быть удалено. Функция возвращает:

- ☐ пустую строку, если результат работы функции успешен;
- ☐ код ошибки при определенных условиях;
- ☐ `Invalid`, если функция завершается аварийно.

Функция *getFieldValue*

Функция извлекает значение `field` из заданного сообщения. Параметр `msgHandle` указывает на сообщение, над которым нужно выполнить действие. Параметр `field` задает извлекаемое поле. Функция возвращает значение, которое зависит от параметра `field` (любая строка может быть задана в качестве этого параметра):

- ☐ пустую строку, если результат работы функции успешен;
- ☐ код ошибки при определенных условиях;
- ☐ `Invalid`, если функция завершается аварийно.

Функция *markAsRead*

Функция отмечает сообщение, как прочитанное. Параметр `msgHandle` определяет сообщение. Функция возвращает пустую строку при успешном завершении или `Invalid`, если она завершается аварийно.

Библиотека *WTAPhoneBook*

Функция *Write*

Функция пишет строку в телефонную книгу поверх существующей записи. Параметр `index` указывает позицию внутри телефонной книги. Если `index` равен 0, то строка записывается в позицию, которая в момент исполнения не заполнена; выбор позиции произволен. Параметр `number` задает телефонный номер, связанный с этой позицией, и должен быть допустимым телефонным номером. Параметр `name` задает имя и может быть пустой строкой.

Функция *Search*

Функция возвращает индекс строки в телефонной книжке, соответствующей заданному критерию поиска. Индекс незаполненной телефонной книжки не возвращается. Параметр `field` указывает, какое поле телефонной книжки должно проверяться. Параметр `value` задает желаемое значение поля, где пустая строка должна быть заменена.

Для того чтобы функция начала выполнение процедуры поиска, она должна быть вызвана с непустой строкой в качестве значения параметра `field`. В этом случае функция будет возвращать индекс всех строк в телефонной книжке, содержащих эту строку в любом месте значения поля `value` без учета регистра. Если параметр `value` является пустой строкой, результатом поиска будут индексы всех непустых строк в телефонной книжке.

Замечание: все реализации должны, как минимум, выполнять сравнение литерных строк между значениями полей параметра `value` и поля `value` из телефонной книжки. Однако, некоторые реализации могут использовать расширенный алгоритм сравнения строк, при котором схожие символы различных алфавитов могут восприниматься как одинаковые. Для продолжения поиска функция должна быть вызвана с пустой строкой для значений `field` и `value`.

Каждый вызов функции возвращает одно значение индекса или значение `Invalid`, если функция завершается аварийно.

Замечание

Эта функция предназначена для использования внутри цикла, который выполняется до тех пор, пока не будет возвращено значение `Invalid`.

Функция *remove*

Функция удаляет строку из телефонной книги, которая соответствует индексу, заданному параметром. Функция возвращает:

- ☐ пустую строку, если функция успешна;
- ☐ код ошибки или `Invalid`, когда функция завершается аварийно.

Функция *getFieldValue*

Функция извлекает значение поля из конкретной строки телефонной книжки устройства, положение которой определяется параметром `index`. Параметр `field` описывает имя извлекаемого поля. Если параметр `field` не может быть распознан, не поддерживается или недоступен, функция возвращает пустую строку.

Функция *change*

Функция записывает заданное значение в указанную строку телефонной книги. Функция возвращает пустую строку, если функция завершается успешно. В противном случае возвращает код ошибки или `Invalid`.

Библиотека *WTACallLog*

Функция *dialled*

Функция возвращает логический идентификатор звонка из регистра выполненных звонков. Функция вызывается повторно для извлечения всех сделанных звонков из регистра. Параметр `returnFirst` указывает, какой звонок должен быть извлечен. Логический идентификатор звонка извлекается из регистра звонков в порядке, обратном порядку их записи. Параметр `returnFirst` имеет значение `true`, если возвращаемый идентификатор относится к самому последнему звонку, или `Invalid`, если таких звонков нет.

Следует заметить, что функция предназначена для использования в цикле, который выполняется до тех пор, пока не будет возвращено значение `Invalid`, что указывает на конец списка звонков.

Функция *missed*

Функция возвращает идентификатор логического вызова из регистра пропущенных звонков. Имеет логику работы, аналогичную работе функции `dialled`.

Функция *received*

Функция возвращает идентификатор логического вызова из регистра принятых звонков. Имеет логику работы, аналогичную работе функции `dialled`.

Функция *getFieldValue*

Функция извлекает значение поля из заданного логического идентификатора звонка. Параметр `field` описывает имя извлекаемого поля. Функция возвращает значение, которое зависит от параметра `field`. Значением параметра `field` может быть любая строка. Тем не менее, если этот параметр соответствует одному из имен полей, описанных в документации, строка должна возвращаться в том виде, в котором она там находится.

Если параметр `field` не распознается, не поддерживается или по какой-либо причине недоступен, возвращается пустая строка.

Библиотека *WTAMisc*

Функция *setIndicator*

Функция модифицирует состояние логического индикатора. Параметр `type` указывает нужный индикатор. Параметр `newState` описывает желаемое состояние индикатора. Функция возвращает пустую строку при успешном завершении и `Invalid` — при аварийном.

Функция *endContext*

Функция завершает сеанс с текущим WTA-агентом и разрушает принадлежащее ему содержание. В качестве результата возвращает пустую строку.

Функция *getProtection*

Функция извлекает режим защиты текущего WTA-содержания. Возвращает булево значение, указывающее на текущий режим защиты.

Функция *setProtection*

Функция устанавливает режим защиты для текущего WTA-содержания. Параметр *mode* задает желаемый режим защиты. Функция возвращает пустую строку при нормальном завершении.

Вспомогательные библиотеки компаний-производителей

Библиотеки *Console*

Функция *Print*

Функция выводит на консоль содержимое строки, заданной в списке параметров.

Пример

```
extern function PrintTest()
{
    var Name1 = "Matt";
    var Name2 = "Krissy";
    var Name3 = "Tommy";
    Console.print (Name1);
    Console.print (Name2);
    Console.print (Name3);
}
```

Результат выполнения кода

```
Console.print
Press OK to test the Console print function
```

Phone information

```
cache miss: <HTTP://127.0.0.1/wapps/print.wmls>
net request: <HTTP://127.0.0.1/wapps/print.wmls>
HTTP GET Request: HTTP://127.0.0.1/wapps/print.wmls
DATA SIZE
Uncompiled data from HTTP is 207 bytes.
...found Content-Type: text/vnd.wap.wmlscript.
[xlateWMLScript] [unknown subscriber] Compiling WMLScript
[xlateWMLScript] WMLScript was successfulCompiled WAP binary is 174
bytes.
Matt Krissy Tommy
```

Функция *PrintLn*

Функция выводит на консоль сообщение построчно.

Пример

```
extern function PrintLnTest()
{
    var Name1 = "Matt";
    var Name2 = "Krissy";
    var Name3 = "Tommy";
    Console.println (Name1);
    Console.println (Name2);
    Console.println (Name3);
}
```

Результат выполнения кода

```
Console.println
Press OK to test the Console println function
```

Phone information

```
DATA SIZE
Uncompiled data from HTTP is 595 bytes.
...found Content-Type: text/vnd.wap.wml.
Compiled WAP binary is 252 bytes.
cache miss: <HTTP://127.0.0.1/wapps/println.wmls>
net request: <HTTP://127.0.0.1/wapps/println.wmls>
HTTP GET Request: HTTP://127.0.0.1/wapps/println.wmls
DATA SIZE
Uncompiled data from HTTP is 215 bytes.
...found Content-Type: text/vnd.wap.wmlscript.
```

```
[xlateWMLScript] [unknown subscriber] Compiling WMLSkript
[xlateWMLSkript] WMLSkript was successfulCompiled WAP binary is 176
bytes.
```

Matt

Krissy

Tommy

Библиотека *Debug*

Функция *Openfile*

Функция открывает файл с именем `filename` в режиме, указанном параметром `mode`. Параметр `mode` может принимать значения:

- ☐ `r` — открыть файл для чтения;
- ☐ `w` — открыть файл для записи;
- ☐ `a` — открыть файл для пополнения.

Функция *Closefile*

Функция закрывает файл.

Функция *PrintLn*

Функция выводит строку, заданную параметром, в файл, открытый для записи.

Коды наборов символов

В табл. 3.9 приведены коды наборов символов, используемые для указания таблиц кодировки.

Таблица 3.9. Коды наборов символов

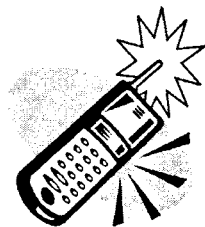
Набор символов	Назначенное число	IANA MIBЕnum-значение
BIG5	0x07EA	2026
ISO-10646-UCS-2	0x03E8	1000
ISO-8859-1	0x04	4
ISO-8859-2	0x05	5
ISO-8859-3	0x06	6
ISO-8859-4	0x07	7
ISO-8859-5	0x08	8

Таблица 3.9 (окончание)

Набор символов	Назначенное число	IANA MIBЕпит-значение
ISO-8859-6	0x09	9
ISO-8859-7	0x0A	10
ISO-8859-8	0x0B	11
ISO-8859-9	0x0C	12
SHIFT_JIS	0x11	17
US ASCII	0x03	3
UTF-8	0x6A	106
GSM-алфавит по умолчанию	Еще не определено	Еще не определено

Глава 4

Справочник по HDML



Тег <HDML>

Значение <html> задает HDML-колоду, о которой и содержит основную информацию.

Синтаксис

```
<HDML VERSION = html_version
    TTL = cache_time
    MARKABLE= boolean
    PUBLIC= boolean
    ACCESSDOMAIN= url
    ACCESSPATH= path>
</HDML>
```

Атрибуты

- ☐ **version** — определяет версию HDML, использованного в колоде. Вы должны обязательно указать номер версии. Номера версии могут быть от 0.1 до 3.0.
- ☐ **TTL** — определяет время (в секундах), в течение которого колода будет находиться в памяти телефона (по умолчанию 30 дней). Если установить нулевое значение — колода не будет запоминаться.
- ☐ **markable** — указывает, можно ли отметить карту в колоде в качестве закладки. Параметр **markable** уровня карты имеет более высокий приоритет по сравнению с тем же параметром уровня колоды.
- ☐ **public** — указывает на необходимость управления доступом для колоды. Если указано значение **true**, любая карта может ссылаться на другую карту в колоде. Если указано **false**, ссылаться можно только на карты из колоды с доменным именем, заданном параметрами **accessdomain** и **accesspath**.
- ☐ **accessDomain** — указывает домен колод, на которые можно ссылаться из текущей колоды карт. По умолчанию текущий домен колоды.
- ☐ **accessPath** — указывает путь колод, на которые можно ссылаться из текущей колоды, при параметре **public**, установленном в значение **false**. По умолчанию имеет значением корневой каталог текущей колоды.

- action — определяет действие для всех карт в колоде.
- cards — одна или несколько карт.

Тег <A>

Ссылка, привязанная к текстовой строке. При прокручивании телефон показывает символ ">" с последующим текстом и изменяет метку на клавише <ACCEPT> на некоторое заранее заданное значение (если оно задано). Если пользователь нажимает клавишу <ACCEPT>, телефон исполняет задачу, связанную со ссылкой.

Синтаксис

```
<A TASK= task_type
  LABEL= accept_key_label
  ACCESSKEY= key_num
  REL=NEXT
  DEST= dest_URL
  VARS= var_pairs
  RECEIVE= var_list
  NEXT= next_dest
  CANCEL= cancel_dest
  FRIEND= boolean
  SENDREFERER= boolean
  RETVALS= value_list
  CLEAR= boolean
  POSTMETHOD= get_or_post
  POSTDATA= data_to_post
  ACCEPT-CHARSET= char_set
  NUMBER= number>
```

text

Атрибуты

- label — метка, которая идентифицирует задачу с механизмом пользовательского интерфейса. Например, если вам нужна задача, связанная ключом accept, устройство показывает это значение в качестве метки функциональной клавиши.
- accesskey — число от 0 до 9, которое появляется с левой стороны экрана перед ссылкой. Если пользователь нажимает соответствующую клавишу на терминале, последний будет выполнять задачу, назначенную ссылкой.

- ❑ `task` — задает задачу, которая должна быть исполнена телефоном при выборе текущей ссылки. Задача `task` может быть любой из следующих:
 - `go` — запрашивает URL, указанный в `Dest`;
 - `gosub` — запрашивает URL, указанный параметром `dest`;
 - `return` — возвращает предыдущую активность из вложенной активности;
 - `cancel` — завершает текущую вложенную активность и возвращает управление к предыдущей активности;
 - `prev` — показывает предыдущую карту из стека посещенных карт;
 - `call` — помещает голосовой вызов с номером телефона, заданном в параметре `number`;
 - `noop` — ничего не исполняет, обычно используется для блокирования действий, исполняемых по умолчанию.

- ❑ `dest` — указывает URL для запроса в задачах `go` и `gosub`.

В задачах `return` или `cancel` во вложенной активности `dest` указывает на URL, который должен запрашиваться после возврата вызываемой активности. Браузер игнорирует эту опцию, если вызывающая активность пометила текущую активность как "friendly".

- ❑ `Rel=Next` — говорит телефону предварительно извлечь URL, заданный в `dest`. Телефон будет затем пытаться загрузить следующий URL в кэш, пока пользователь просматривает текущую карту. Если пользователь запрашивает кэшированный URL, браузер может показать данные без ожидания завершения загрузки.
- ❑ `method` — имеет значение `GET` (по умолчанию) или `POST`. Описывает метод подстановки для HTTP-протокола. Указание "post" приведет к тому, что WAP-шлюз будет декодировать данные, представляющие значения переменных в набор символов, описанных HTTP-заголовком. Следует предусмотреть это преобразование, если устройство передает не ASCII-набор символов (например, UTF-8). Информация об используемых кодировках и HTTP-заголовках может быть найдена, например, в руководстве по разработке компании Phone.com.
- ❑ `postdata` — задает передаваемые данные, если параметр `method` описан, как `post`. Если данные содержат более одного аргумента, необходимо разделить их знаками амперсанда.
- ❑ `accept-Charset` — указывает процедуру кодирования, которую ваше приложение будет принимать во внимание при выполнении. Устройство использует этот атрибут, чтобы преобразовывать данные, указанные в элементе `<postfield>`.

Вы можете также опустить этот атрибут, если указали вашу кодировку в заголовке HTTP-запроса.

- ❑ `vars` — описывает список имен переменных и значений при выполнении задачи `go` или `gosub`. Этот список должен иметь следующий формат:
`var1=value1&:var2=value2`
Значения должны соответствовать соглашениям, принятым при работе с URL. Телефон затем будет разбирать последовательности прежде чем их использовать.
- ❑ `receive` — задает разделенный запятыми список имен переменных, которые телефон использует для сохранения значений, возвращаемых в задаче `gosub`.
- ❑ `retvals` — задает разделенный запятыми список значений, возвращаемый в вызывающую задачу. Этот параметр можно установить только с задачей `return`.
- ❑ `next` — указывает URL для запроса после того, как вложенная активность возвращает управление в вызывающую активность. Этот параметр имеет более низкий приоритет по сравнению с аналогичным параметром из вложенной активности.
- ❑ `cancel` — указывает URL, который нужно запросить после завершения вызываемой задачи `gosub`.
- ❑ `friend` — указывает, должна ли вложенная активность `gosub` работать в доверительном режиме. Доверительная активность может использовать параметры `dest` и `clear` в опции `return` и `cancel`. По умолчанию `false`.
- ❑ `sendreferer` — имеет значение `true` или `false` (по умолчанию). Описывает, следует ли устройству включать URL-колоды в URL-запрос. Указание `true` приведет к тому, что устройство установит заголовок `HTTP_REFERER` на относительный URL запрашиваемой колоды.
Если вы хотите ограничить доступ к защищенным сервисам, колоды которых запрашивают конкретный URL, то должны устанавливать эту опцию в значение `true`.
- ❑ `clear` — указывает, может ли задача `return` или `cancel` из доверительной вложенной активности сбросить все переменные вызывающей задачи. Телефон не будет игнорировать параметр `clear` для всех задач, вызываемых с параметром `true`. По умолчанию `false`.
- ❑ `number` — указывает номер телефона для задачи `call`.
- ❑ `text` — указывает текст, который браузер будет показывать в скобках для ссылки.

Тег <ACTION>

Действие связывает задачу с функциональной клавишей телефона, такой как <ACCEPT>, <PREV>, <HELP>, <SEND>, <DELETE>, <SOFT1> или

<SOFT2>. Когда пользователь нажимает функциональную клавишу, телефон исполняет назначенную задачу.

Синтаксис

```
<ACTION TYPE= key_name
    TASK= task_type
    LABEL= key_label
    REL=NEXT
    DEST= dest_URL
    VARS= var_pairs
    RECEIVE= var_list
    NEXT= next_dest
    CANCEL= cancel_dest
    FRIEND= boolean
    SENDREFERER= boolean
    RETVALS= value_list
    CLEAR= boolean
    POSTMETHOD= get_or_post
    POSTDATA= data_to_post
    ACCEPT-CHARSET= char_set
    NUMBER= number
    SRC= image_url
    ICON= icon_name>
```

Атрибуты

- ❑ **label** — указывает метку для клавиши, в соответствии с ее типом назначенной активности. Длина метки не должна превышать пяти символов. По умолчанию для клавиши принята метка "OK".

Задание метки для клавиши <АКЦЕПТ> необязательно. Если вы назначили метку для `prev`, она игнорируется. Любой другой ключ требует метки.

- ❑ **key** — задает ключ на телефоне, который будет активировать задачу. Разрешенные активности: <ACCEPT>, <HELP>, <PREV>, <SOFT1>, <SOFT2>, <SEND>, <DELETE>.

- ❑ **task** — определяет задачу, которая будет выполнена после выбора пользователя. Задача может быть одной из следующих:

- `go` — запрашивает URL, указанный в `dest`;
- `gosub` — запрашивает URL, указанный параметром `dest`;
- `return` — возвращает предыдущую активность из вложенной активности;

- `cancel` — завершает текущую вложенную активность и возвращает управление к предыдущей активности;
- `prev` — показывает предыдущую карту из стека посещенных карт;
- `call` — помещает голосовой вызов с номером телефона, заданном в параметре `number`;
- `noop` — ничего не исполняет, обычно используется для блокирования действий, исполняемых по умолчанию.

❑ `dest` — определяет URL для запроса в задачах `go` и `gosub`.

❑ `rel=Next` — говорит телефону предварительно извлечь URL, заданный в `dest`. Телефон будет затем пытаться загрузить следующий URL в кэш, пока пользователь просматривает текущую карту. Если пользователь запрашивает кэшированный URL, браузер может показать данные без ожидания завершения загрузки.

❑ `method` — имеет значение `get` (по умолчанию) или `post`. Описывает метод подстановки для HTTP-протокола. Указание `post` приведет к тому, что WAP-шлюз будет декодировать данные, представляющие значения переменных в набор символов, описанных HTTP-заголовком. Следует предусмотреть это преобразование, если устройство передает не ASCII-набор символов (например, UTF-8). Информация об используемых кодировках и HTTP-заголовках может быть найдена, например, в руководстве по разработке компании Phone.com.

❑ `postdata` — задает передаваемые данные, если параметр `method` описан как `post`. Если данные содержат более одного аргумента, необходимо разделить их знаками амперсанда.

❑ `accept-Charset` — указывает процедуру кодирования, которую ваше приложение будет принимать во внимание при выполнении. Устройство использует этот атрибут, чтобы преобразовывать данные, указанные в элементе `<postfield>`.

Вы можете также опустить этот атрибут, если указали вашу кодировку в заголовке HTTP-запроса

❑ `vars` — описывает список имен переменных и значений при выполнении задачи `go` или `gosub`. Этот список должен иметь следующий формат:

```
var1=value1&:var2=value2
```

Значения должны соответствовать соглашениям, принятым при работе с URL. Телефон затем будет разбирать последовательности, прежде чем их использовать.

❑ `receive` — задает разделенный запятыми список имен переменных, которые телефон использует для сохранения значений, возвращаемых в задаче `gosub`.

- ❑ `retvals` — указывает список значений, возвращаемых задачей `gosub`, в вызывающую активность. Параметр `retvals` используется только для задачи `return`.
- ❑ `next` — указывает URL для запроса после того, как вложенная активность возвращает управление в вызывающую активность. Этот параметр имеет более низкий приоритет по сравнению с аналогичным параметром из вложенной активности.
- ❑ `cancel` — указывает URL, который нужно запросить после завершения вызываемой задачи `gosub`.
- ❑ `friend` — указывает, должна ли вложенная активность `gosub` работать в доверительном режиме. Доверительная активность может использовать параметры `dest` и `clear` в опции `return` и `cancel`. По умолчанию `false`.
- ❑ `sendreferer` — имеет значение `true` или `false` (по умолчанию). Описывает, следует ли устройству включать URL-колоды в URL-запрос. Указание `true` приведет к тому, что устройство установит заголовок `HTTP_REFERER` на относительный URL запрашиваемой колоды.

Если вы хотите ограничить доступ к защищенным сервисам, колоды которых запрашивают конкретный URL, то должны устанавливать эту опцию в значение `true`.

- ❑ `clear` — указывает, может ли задача `return` или `cancel` из доверительной вложенной активности сбросить все переменные вызывающей задачи. Телефон не будет игнорировать параметр `clear` для всех задач, вызываемых с параметром `true`. По умолчанию `false`.
- ❑ `number` — указывает номер телефона при выполнении задачи `call`.
- ❑ `src` — задает URL графического объекта, который будет использоваться браузером при задании метки для горячей клавиши. Эта возможность игнорируется, если корректное имя описано в параметре `icon`. Так как горячие клавиши не имеют меток по умолчанию, необходимо также задать параметр `label` при использовании `src`, так, чтобы телефон мог показать текст, если он не в состоянии разместить заданный разработчиком значок.
- ❑ `icon` — указывает имя локального графического объекта для горячей клавиши. Если браузер не в состоянии это исполнить из своей памяти, он будет пытаться получить его с текущего шлюза UP.Link. Так как горячие клавиши не имеют меток по умолчанию, необходимо использовать параметр `label` на случай невозможности показа желаемого графического объекта.

Тег <ENTRY>

Показывает подсказку и позволяет пользователю ввести данные.

Синтаксис

```
<ENTRY NAME= card_name
    MARKABLE= boolean
    TITLE= card_title
    BOOKMARK= mark_URL>
    FORMAT= format_specifier
    DEFAULT= default_value
    KEY= var
    NOECHO= boolean
    EMPTYOK= boolean>
```

actions

text

</ENTRY>

Атрибуты

- ❑ **name** — определяет имя карты. Другие карты могут ссылаться на данную карту, используя это имя как цель.
- ❑ **markable** — определяет, может ли карта использоваться в качестве источника для хранения закладки. По умолчанию **markable** для колоды.
- ❑ **title** — задает имя по умолчанию, которое будет использовано при сохранении закладки.
- ❑ **bookmark** — описывает URL, который телефон будет добавлять к закладкам, если пользователь захочет это сделать для текущей карты. При отсутствии значения используется URL текущей карты.
- ❑ **format** — в качестве значений атрибута **format** могут использоваться следующие метасимволы:
 - **A** — любой алфавитно-цифровой символ в верхнем регистре **A—Z** или знак пунктуации;
 - **a** — любой алфавитно-цифровой символ в нижнем регистре **a—z** или знак пунктуации;
 - **N** — любая цифра;
 - **X** — любой алфавитно-цифровой символ в верхнем регистре **A—Z**, цифры **0—9** или знак пунктуации;
 - **x** — любой алфавитно-цифровой символ в нижнем регистре **a—z**, цифры **0—9** или знак пунктуации;
 - **M** — любой символ;
 - **m** — любой символ, по умолчанию первый символ в нижнем регистре.
- ❑ **default** — описывает текст, который появится в поле ввода, когда браузер впервые покажет карту. Пользователь может редактировать этот текст.

- ❑ `key` — задает имя переменной, которая хранит текст, введенный пользователем. Если описываемая переменная имеет значение, оно используется в качестве начального значения при вводе. Если переменная имеет значение, то оно имеет более высокий приоритет по сравнению с заданным в параметре `default`.
- ❑ `noecho` — указывает, следует ли телефону скрывать текст, который вводит пользователь. Если значение параметра установлено в значении `true`, то телефон будет заменять каждый введенный символ текста на звездочку. По умолчанию `false`.
- ❑ `emptyok` — указывает, может ли пользователь принять пустую строку при вводе переменной.
- ❑ `actions` — описывает действия, которые телефон будет выполнять при нажатии пользователем соответствующей клавиши.
- ❑ `text` — текст заголовка, который браузер будет показывать выше вводимой строки.

Тег <CHOICE>

Отображает текст, помещенный в списке, из которого пользователь может выбрать только один пункт. Инструкция <CHOICE> должна содержать одно или несколько утверждений <CE...>, которые определяют пункты в списке.

Синтаксис

```
<CHOICE NAME= card_name
      MARKABLE= boolean
      TITLE= card_title
      BOOKMARK= mark_URL
      KEY= varname
      IKEY= varname
      METHOD= choice_method
      DEFAULT= default_val
      IDEFAULT=defaultnum>
```

```
actions
text
    <CE ...> text1
...
</CHOICE>
```

Атрибуты

- ❑ `name` — определяет имя карты. Другие карты могут ссылаться на данную карту, используя это имя как цель.

- ❑ `markable` — определяет, может ли карта использоваться в качестве источника для хранения закладки. По умолчанию `markable` для колоды.
- ❑ `title` — задает имя по умолчанию, которое будет использовано при сохранении закладки для карты.
- ❑ `bookmark` — описывает URL, который телефон будет добавлять к закладкам, если пользователь захочет это сделать для текущей карты. При отсутствии значения используется URL текущей карты.
- ❑ `key` — задает имя переменной, которая получает значение, когда пользователь выбирает элемент из списка.
- ❑ `key` — указывает имя переменной, которая первоначально содержит индекс выбора по умолчанию. Если значение не специфицировано, значение из параметра `idefault` становится активным. Если оно также не имеет значения, используется первый элемент из списка.
- ❑ `method` — указывает может ли браузер нумеровать список.
 - Если значение — `number`, телефон автоматически нумерует список. При этом пользователь может выбирать из списка, просто нажимая клавиши с цифрами на клавиатуре.
 - Если значение — `alpha`, то список не нумеруется, и пользователь должен выполнять прокрутку и затем нажимать клавишу <АССЕРТ>.
- ❑ `default` — определяет значение переменной из списка выбора, принятое по умолчанию. Если переменная в параметре `key` не имеет значения в момент показа карты, оно назначается на указанное значение. Если переменная имеет значение, то оно не изменяется.
- ❑ `idefault` — указывает `index`, принятый для списка выбора по умолчанию.
- ❑ `actions` — указывает действия, которые выполняются при нажатии на функциональные клавиши телефона.
- ❑ `text` — текст, который задает заголовок для браузера, который будет показан над списком выбора. Если текст напечатан в режиме <"line">, то текст будет усекаться в случае необходимости.
- ❑ <CE...>`text` — задает элемент в списке, который пользователь может выбрать, и текст, описывающий строку выбора.

Тег <CE>

Задаёт элементы списка "CHOICE", из которого пользователь может выбрать нужный элемент.

Синтаксис

```
<CE VALUE= choice_value  
      TASK= task_type
```

```

LABEL= accept_key_label
REL=NEXT
DEST= dest__URL
VARS= var_pairs
RECEIVE= var_list
NEXT= next_dest
CANCEL= cancel_dest
FRIEND= boolean
SENDREFERER= boolean
RETVALS= value_list
CLEAR= boolean
POSTMETHOD= get_or_post
POSTDATA= data_to_post
ACCEPT-CHARSET= char_set
NUMBER= number>

```

text

Атрибуты

- ☐ value — задает дополнительное значение, которое телефон будет связывать с выбранной строкой меню в <CE...> .
- ☐ label — определяет метку клавиши <АКЦЕПТ> при выборе ее пользователем. Длина метки не должна превышать пяти символов. По умолчанию OK.
- ☐ task — определяет задачу, которая будет выполнена, после выбора пользователя. Задача может быть одной из следующих:
 - Ggo — запрашивает URL, указанный в dDest;
 - gosub — запрашивает URL, указанный параметром dest;
 - return — возвращает предыдущую активность из вложенной активности;
 - cancel — завершает текущую вложенную активность и возвращает управление к предыдущей активности;
 - prev — показывает предыдущую карту из стека посещенных карт;
 - call — помещает голосовой вызов с номером телефона, заданном в параметре number;
 - noop — ничего не исполняет, обычно используется для блокирования действий, исполняемых по умолчанию.
- ☐ dest — определяет URL для запроса в задачах go и gosub.
- ☐ rel=Next — говорит телефону предварительно извлечь URL, заданный в dest. Телефон будет затем пытаться загрузить следующий URL в кэш, пока пользователь просматривает текущую карту. Если пользователь за-

прашивает кэшированный URL, браузер может показать данные без ожидания завершения загрузки.

- ❑ `method` — имеет значение `GET` (по умолчанию) или `POST`. Описывает метод подстановки для HTTP-протокола. Указание `"post"` приведет к тому, что WAP-шлюз будет декодировать данные, представляющие значения переменных в набор символов, описанных HTTP-заголовком. Следует предусмотреть это преобразование, если устройство передает не ASCII-набор символов (например, UTF-8). Информация об используемых кодировках и HTTP-заголовках может быть найдена, например, в руководстве по разработке компании Phone.com.
- ❑ `postdata` — задает передаваемые данные, если параметр `method` описан как `post`. Если данные содержат более одного аргумента, необходимо разделить их знаками амперсанда.
- ❑ `accept-charset` — указывает процедуру кодирования, которую ваше приложение будет принимать во внимание при выполнении. Устройство использует этот атрибут, чтобы преобразовывать данные, указанные в элементе `<postfield>`.

Вы можете также опустить этот атрибут, если указали вашу кодировку в заголовке HTTP-запроса.

- ❑ `vars` — описывает список имен переменных и значений при выполнении задачи `go` или `gosub`. Этот список должен иметь следующий формат:

```
var1=value1&:var2=value2
```

Значения должны соответствовать соглашениям, принятым при работе с URL. Телефон затем будет разбирать последовательности прежде чем их использовать.

- ❑ `receive` — задает разделенный запятыми список имен переменных, чьи значения телефон должен сохранить при возвращении из `gosub`.
- ❑ `retvals` — указывает список значений, возвращаемых задачей `gosub` в вызывающую активность. Параметр `retvals` используется только для задачи `return`.
- ❑ `next` — указывает URL для запроса после того, как вложенная активность возвращает управление в вызывающую активность. Этот параметр имеет более низкий приоритет по сравнению с аналогичным параметром из вложенной активности.
- ❑ `cancel` — указывает URL, который нужно запросить после завершения вызываемой задачи `gosub`.
- ❑ `friend` — указывает, должна ли вложенная активность `gosub` работать в доверительном режиме. Доверительная активность может использовать параметры `dest` и `clear` в опции `return` и `cancel`. По умолчанию `false`.

- ❑ `sendreferer` — имеет значение `true` или `false` (по умолчанию). Описывает, следует ли устройству включать URL-колоды в URL-запрос. Указание `true` приведет к тому, что устройство установит заголовок `HTTP_REFERER` на относительный URL запрашиваемой колоды. Если вы хотите ограничить доступ к защищенным сервисам, колоды которых запрашивают конкретный URL, то должны устанавливать эту опцию в значение `true`.
- ❑ `clear` — указывает, может ли задача `return` или `cancel` из доверительной вложенной активности сбросить все переменные вызывающей задачи. Телефон не будет игнорировать параметр `clear` для всех задач вызываемых с параметром `true`. По умолчанию `false`.
- ❑ `number` — описывает номер вызываемого в задаче `call` телефона.
- ❑ `text` — указывает текст, который браузер будет показывать в скобках для ссылки.

Тег **<DISPLAY>**

Отображает форматированный текст.

Синтаксис

```
<DISPLAY NAME= card_name
      MARKABLE= boolean
      TITLE= card_title
      BOOKMARK= mark_URL>
```

actions

display_text

</DISPLAY>

Атрибуты

- ❑ `name` — определяет имя карты. Другие карты могут использовать это имя для ссылки на данную карту.
- ❑ `title` — задает имя по умолчанию для случая, когда пользователь будет запоминать карту.
- ❑ `markable` — определяет, может ли пользователь запомнить текущую карту. По умолчанию значение `markable` определено как "да" для всей колоды.
- ❑ `bookmark` — описывает URL, который телефон будет добавлять к своей закладке в списке ресурсов, если пользователь хочет запомнить карту. Если задано значение "нет", то используется URL текущей карты.
- ❑ `actions` — задает действия на телефоне, которые будут исполняться, когда пользователь нажмет клавишу.
- ❑ `text` — определяет форматированный текст, который отображает карта.

Тег <NODISPLAY>

Инструкция <NODISPLAY> используется для определения переменных. Эта карта немедленно выполняется без отображения на экране телефона.

Синтаксис

```
<NODISPLAY NAME= card_name>
```

```
actions
```

```
</NODISPLAY>
```

Атрибуты

- ☐ name — определяет имя карты. Другие карты могут использовать это имя для ссылки на данную карту.
- ☐ actions — определяет действие, которое необходимо выполнить, если пользователь ссылается на данную карту.

Тег

Задаёт графический объект, который появится в форматированном тексте, если телефон способен показывать графику. Изображение может быть помещено в любое место строки форматированного текста, такое как текст в области списка, экран или карты ввода данных. Графические образы могут также использоваться в качестве меток для обозначения запрограммированных клавиш.

Синтаксис

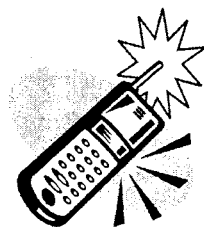
```
<IMG ALT= alt_text SRC= image_url ICON= icon_name>
```

Атрибуты

- ☐ alt — определяет поведение устройства в случае, если оно не поддерживает графику или не может найти предписанный для показа графический объект.
- ☐ src — задаёт URL графического объекта, который должен быть показан. Если задан корректный значок в атрибуте icon, то устройство должно игнорировать этот атрибут.
- ☐ Icon — имя известного значка. Если устройство не в состоянии найти значок в ROM-памяти (Read Only Memory, память только для чтения), оно должно попытаться извлечь ее с сервера WML-содержания.

Глава 5

Голосовые диалекты XML



Телефонная сеть как платформа для разработки приложений

Мировая паутина и телефонные сети подвержены конвергенции. В настоящее время телефонная сеть становится программируемой средой и может в самом ближайшем будущем снова изменить наше представление о возможностях Интернета.

С появлением платформ для голосовых приложений создание услуг для обычных телефонов вскоре станет таким же простым и понятным делом, каким сегодня является создание Web-странички. Это станет возможным благодаря большому количеству новых услуг, особенно для пользователей беспроводных сетей.

Зачем нужна платформа разработки?

Почему Web выросла так быстро? Большинство специалистов обращают внимание на то, как легко было получить доступ в Интернет и как просто с помощью обычного браузера было получить информацию из Web. Однако не менее, если не более, важным был процесс создания этого содержания и его публикация в Интернете, а также создание динамических приложений в Web. HTML, в качестве языка разметки Web, оказался достаточно простым, что дало возможность его использования даже не программистами.

Благодаря простому графическому интерфейсу и возможности размещения информации на таких сайтах, как GeoCities (сегодня это часть Yahoo!), почти каждый мог построить собственную страничку и сделать ее доступной для мировой аудитории. Даже более сложные сайты, такие как online-склады, требовали только небольшого количества времени профессиональных программистов. Провайдеры Web-хостинга предоставляли возможность физического размещения серверов и связанным с ним включением в Web, освобождая разработчиков от рутины и направляя их активность непосредственно на процесс создания услуг, которые нужно оказывать на серверах.

В результате тысячи компаний и миллионы индивидуальных стали способны создавать свои собственные уникальные службы, основанные на Web-присутствии.

Программируемость позволяет проводить эксперименты. Инновационные команды специалистов, которые создали такие компании, как eBay и Amazon.com, не нуждались в централизованном управлении при реализации всех своих идей, связанных с ведением бизнеса. Они просто публиковали свои сайты в Сети и переделывали их время от времени в соответствии с изменившимися взглядами и появившимся опытом.

В отличие от интернет-приложений, телефонные приложения и телефонные сети исторически работают в более замкнутом окружении. Только операторы и связанные с ними компании со значительными техническими и финансовыми ресурсами могли создавать пользовательские услуги. Современные услуги, такие как Internet Call waiting и Internet Call forwarding тесно связаны с возможностями, жестко прошитыми в центральных коммутационных системах оператора связи.

Бизнесмены, которые хотят использовать услуги голосовой почты и быстрого набора номера, должны либо поставить дорогое оборудование PBX, либо заказать услуги Centrex, предлагаемые их телефонной компанией. Использование платформы компьютерной телефонии от таких компаний, как Dialogic, является дорогостоящим, трудноконфигурируемым и не полностью стандартизованным процессом. Как следствие, реализация вновь создаваемых услуг происходит крайне медленно. Исключение составляют самые крупные предприятия, которые имеют собственные подразделения, отвечающие за обеспечение связи во всех офисах компаний.

Поскольку для конечного пользователя никогда не существовало возможности разработки и использования телефонных услуг, возникла тенденция игнорировать потенциал телефонных сетей как платформы для работы приложений. В то же время в мире уже существует более миллиарда телефонов. Телефонные услуги имеют более чем 90-процентное проникновение в большинстве западных стран. Добавьте к этому сотни миллионов абонентов сотовой связи и тенденцию их роста до 1.4 миллиарда к 2004 году.

Все абоненты сотовой связи подключены к сети и все способны принимать речь в качестве входной информации.

Доступ в Интернет имеет другую особенность. В мире существует 200 миллионов пользователей Интернета, которые осуществляют доступ в Интернет при помощи персональных компьютеров. Сами пользователи в момент получения доступа также находятся в месте расположения этих ПК.

В отличие от фиксированного доступа в Интернет, абоненты сотовых телефонных сетей всегда подключены к ним и получают услуги в произвольном месте расположения абонента.

Принимая во внимание данные факторы, существуют огромные возможности привнесения программируемых услуг в мир телефонии. Несколько компаний, такие как Tellme и Voxeo, в настоящее время пытаются реализовать эти возможности. Голосовые порталы, ориентированные на пользователя, можно теперь создавать гораздо проще, а создание единой среды исполнения для поддержания системы таких услуг становится все более актуальным.

Основные возможности

Голосовые порталы (Tellme, BeVocal, AOL — первоначально Quack и Hey Anita) предоставляют на любой телефон, с которого выполняется звонок пользователя, различную информацию, например, цены на акции, спортивные новости и т. п. При этом используется технология распознавания речи.

С технологической точки зрения процесс предоставления таких услуг скорее выглядит похожим на процесс обслуживания посетителя Web-сайта, чем на услугу телефонной системы. Таким образом, намечаются основные возможности для дальнейшего развития. Приложения голосового портала, так же как и приложения на Web-сайте, просто заменять и модифицировать.

Основатели Tellme, которые пришли из Netscape и Microsoft, с самого начала понимали, что создание платформы будет иметь гораздо больше возможностей для бизнеса, чем просто одиночный портал сам по себе. Они предложили пользователям набор инструментов для создания собственных приложений и предоставили им возможность разместить приложения на платформе собственного производства — Tellme.

Tellme сделала большой шаг в предоставлении и заказе локальных телефонных линий, соединяя эти телефонные сети с обрудованием компьютерной телефонии в центрах данных, управляя трафиком этих сетей и доставляя согласованные голосовые пользовательские интерфейсы. В ответ разработчики получают возможность создавать собственные персонализированные приложения и делать их доступными всем желающим пользователям обычного телефона.

Tellme также привлекает пользователей к собственным услугам голосового портала, который помогает оплачивать телефонную инфраструктуру и дает независимым разработчикам подготовленную аудиторию.

Поскольку Tellme в основном была активна в области создания самой платформы вместе с пользовательскими сервисами, другие компании, такие как BeVocal также работали в этом направлении. Еще один шаг вперед: Voxeo "тихо разработал" чисто голосовую инфраструктуру для телефонных услуг, делая платформу разработки своим основным предложением.

Для этих компаний скорость появления платформы имела экономическое значение. В отличие от пользовательских голосовых порталов, которые обычно бесплатны и спонсируются рекламодателями, разработка и услуги хостинга могут генерировать постоянные ежемесячные платежи, которые растут по мере увеличения использования платформы.

Пример приложения

Что такое телефонное приложение? Представим себе, что вы представитель малого бизнеса (медицинский центр или салон-парикмахерская), и вы хотите

24 часа в сутки принимать от клиентов заявки на обслуживание. Оплата диспетчера, который будет сидеть на телефоне в течение всех суток, — дорогое удовольствие. Аренда call-центра, где операторы принимают звонки от клиентов, также дорога и не обеспечивает занесение информации о возможной встрече в ваш электронный ежедневник. Вы ограничены автоответчиком или голосовой почтой. Если у вас есть свой сайт (разработка которого не требует больших финансовых затрат), то можно получать заказы онлайн, однако необходимо, чтобы потенциальный клиент имел доступ в сеть.

Теперь представьте, что вы создали телефонное приложение исключительно средствами вашего сайта. Посетители могут звонить по вашему телефонному номеру прямо в приложение — расписание приема — с использованием автоматического распознавания речи. Если вы хотите добавить дополнительные возможности, такие как объявления или каталог продуктов компании, то это можно сделать минимальными усилиями.

Компании United Airlines, Home Shopping Network и семь крупнейших брокерских фирм в США уже используют телефонные службы с автоматическим распознаванием речи для уменьшения затрат на содержание операторов в call-центрах.

Новые разработки позволяют создавать такие службы почти любому абоненту телефонной сети, и приложение по формированию расписания — только одна из возможностей. Система Unified messaging (Unified messaging, Интегрированная система обработки сообщений), например, как OneBox, позволяет обращаться ко всем входящим сообщениям (телефонным звонкам, автоответчику и факсимильной связи) с любого телефона.

Продукты компании Etrieve делают возможным прослушивание сообщений электронной почты по телефону. С появлением голосовых платформ, таких как Tellme или Voxeo, стало еще проще создавать системы, подобные Unified messaging. Обе эти компании предлагают графические инструменты для компоновки голосовых приложений и генерации соответствующих XML-кодов автоматически.

При использовании подобных платформ отдельно стоящая система Unified messaging может быть привлекательна и в тех случаях, когда система должна выдерживать большую нагрузку при незначительных изменениях конфигурации оборудования или нет возможности сопровождать собственное оборудование. Рынок компаний, создающих собственные приложения, которые будут удовлетворять их уникальные потребности, в конце концов, будет шире. Для того чтобы понять это — посмотрите на размер рынка серверов Web-приложений от таких компаний, как BEA Systems, Microsoft, IBM, Allaire и Sun.

Стандартизация VoiceXML и CallXML

Еще одной гранью технологий создания голосовых приложений являются стандарты. Web вырос на HTML, динамические услуги business-to-business

основаны на XML, а услуги беспроводного доступа к данным построены на WAP и i-mode. Во всех случаях первоначально разрабатывался общий подход к предлагаемому решению, который в дальнейшем приводил к появлению новых стандартов. Эти стандарты в конце концов становятся основой для возникновения рынка подобных приложений.

В настоящее время существует две версии стандарта языка разметки для голосовых услуг — VoiceXML и CallXML. Стандарт VoiceXML появился в 1999 году, а версия 1.0 была выпущена в марте 2000 года. VoiceXML объединил усилия компаний IBM (SpeechML), Motorola (VoxML), Lucent и AT&T (проект PhoneWeb в Bell Labs). Группа расширила список своих членов с 4 компаний до более чем 140 участников. Ведущие голосовые порталы, такие как Tellme, широко используют диалекты языка VoiceXML.

VoiceXML является специализированным диалектом XML. Он позволяет строить голосовые приложения в виде системы документов, в которых определены диалоги с использованием стандартных тегов разметки. Этот подход позволит упростить создание новых приложений и даст возможность приложениям и пользователям мигрировать между различными платформами.

Значительное количество компаний разрабатывает серверы VoiceXML и браузеры, создавая новые потребности и возможности расширения существующих стандартов. Эти разработки будут создавать отклонения от стандартов, некоторые из которых уже начинают появляться.

Язык CallXML, который был предложен компанией Voxeo, предназначен для решения задач, отличных от декларируемых создателями стандарта VoiceXML. VoiceXML покрывает уровень содержания голосовых услуг, такие как меню, команды пользователя, проигрываемые файлы и логика приложения.

CallXML сфокусирован на управлении течением вызова, таком как информация о маршруте, набор конкретного номера, управление конференцией и так далее.

Платформы голосовых приложений позволят разработчикам выполнять инновации гораздо быстрее, чем сегодня. Это будет особенно важно для беспроводных операторов, где персонализированные услуги и способность взаимодействия с локальными приложениями и устройствами являются фундаментальными.

Интернет Сервис Провайдеры (Internet service providers — ISP) и Web-порталы не контролируют путь поступления запроса, который пришел на сайт. Этот факт используется ими для разгрузки своих серверов за счет перенаправления пользователей к партнерам или дочерним сайтам. Аналогично пользователи PalmPilot отдаляются от сообщества независимых разработчиков создания программного обеспечения для этой платформы. Помимо прочего операторы имеют тенденцию контролировать большинство существующих услуг доступа к данным по беспроводным каналам связи.

Принятие телефона в качестве прикладной платформы будет освобождать пользователей от тирании операторов беспроводной связи, потому что телефонные услуги будут способны комбинировать речь, тоновые и беспроводные механизмы передачи данных в новых типах приложений.

Оба языка — VoiceXML и CallXML — основаны на XML, том же самом протоколе обмена данными, что и у Web-приложений и услуг. Со временем можно ожидать пересечения Web и телефонных приложений.

Позволяя технологии интернет-телефонии маршрутизировать голосовые вызовы в сетях с интернет-протоколом (IP), замкнутая модель традиционной телефонии становится частью прошлого.

Роль мета-языка XML

Читатель этих строк наверняка слышал о новых стандартах беспроводного доступа к данным на основе таких инструментов, как WAP, WML, WMLScript и других связанных с ними технологий. Может быть вы даже пытались их использовать в своей текущей деятельности и нашли, что старые проблемы совместимости различных браузеров Navigator и Internet Explorer ничто по сравнению с несовместимостью браузеров для мобильных устройств. Но открытие ваших существующих Web-приложений для легионов беспроводных абонентов является сильной мотивацией расширения сферы вашего бизнеса.

Что если существует путь предоставления существующих ресурсов Web-сайта не только интернет-пользователям, но также и любым абонентам телефонных сетей любого сорта? Что если такая технология может быть основана на открытых стандартах прямо сегодня?

Ответом является положительный ответ с использованием CallXML, VoiceXML и компании Voxeo, помогающей предоставлять доступ к информации вашего сайта всем владельцам телефонных аппаратов любого типа.

Основы Web-телефонии

Для того чтобы начать пользоваться телефонными голосовыми приложениями, размещенными в Web, необходим только браузер с функцией голосовой поддержки, который может соединяться с Web-сервером, и страничка на этом сервере, которую этот браузер будет интерпретировать.

Технически превращение существующих Web-приложений, написанных на одном из языков сценариев (Perl, CFML, PHP, JSP, ASP), в голосовые аналоги на основе WML обычно требует соединения беспроводного устройства с Web-сервером и вывод существующего контента в формате WML вместо HTML.

Но что делать людям, которые не имеют сотового телефона или пользуются услугами оператора, который не дает доступа к WAP-шлюзу.

Использование телефона в роли браузера Web-приложений будет тогда требовать только соединения с Web-сервером и некоторого способа превращения содержания к такому виду, который будет понятен телефонному браузеру.

Конечно, в случае стандартного телефонного аппарата, метод соединения был бы ограничен набором телефонного номера, а единственным браузером была бы комбинация уха абонента, голоса и телефонной клавиатуры. Именно такой способ взаимодействия поддерживают современные технологии и стандарты, которые помогают обеспечить разработку телефонных Web-приложений.

На стороне браузера WT приложение выполняет прямое взаимодействие с пользователем посредством звука и голоса, при этом сам сценарий взаимодействия находится на стороне сервера.

В настоящее время существует по крайней мере три различных языка создания таких сценариев:

❑ **VoiceXML.** VoiceXML (<http://www.voicexml.org/>) представляет собой попытку решить одну из самых частых задач, возникающих при создании сервиса в телекоммуникациях: проблему голосового взаимодействия, что дает возможность говорить "два" вместо того, чтобы нажимать на кнопку (общее название для подобных технологий — IVR, Interactive Voice Response). Кроме того, VoiceXML имеет и другую цель — он позволяет использовать для голосового наполнения содержимое Web. Однако у VoiceXML есть один существенный недостаток — он не контролирует "течение звонка" (call flow). Под термином "течение звонка" подразумевается то, как ведется взаимодействие с пользователем на другом конце провода во время звонка. VoiceXML просто определяет приглашения, меню и опции, с которыми имеет дело пользователь. С помощью VoiceXML можно написать простейшую программу с технологией IVR, но ему по плечу только простые голосовые взаимодействия. С ростом числа элементов программа усложняется и становится слабо визуализируемой. Для более сложных диалогов, в том числе и для управления "течением звонка", необходим более общий подход. VoiceXML основан на стандарте XML и управляется организацией VoiceXML FORUM, основанной Lucent, Motorola, IBM и AT&T. Эта организация направлена на поддержку создания голосовых систем автоматического ответа и включает системы распознавания речи, так же как и системы преобразования текста в речь (text-to-speech, tts).

❑ **CallXML.** CallXML нацелен на то, где VoiceXML проявляет свои слабости. Этот XML-диалект, разработанный компанией Voxeo (<http://community.voxeo.com/>), пользуется успехами, достигнутыми VoiceXML, но также позволяет контролировать "течение звонка" (call flow). В отличие от

VoiceXML, CallXML может управляться событиями, т. е. определенные блоки CallXML запускаются, когда случаются те или иные события. Допустим, мы голосом отдаем CallXML распоряжение взять трубку и ответить на входящий звонок. Но что будет, если при ответе случится ошибка? Элемент запустит событие `onError`, и оно может быть "поймано" в любом месте CallXML. Другими словами, CallXML использует модель событий, чтобы управлять различными путями, по которым может идти звонок. CallXML обеспечивает полный контроль в "течении звонка", применим для Web и прост в использовании. Язык CallXML проще, чем VoiceXML для традиционных телефонных приложений с использованием тоновых телефонных аппаратов.

- ❑ **Microsoft WTE**, набор модулей, который поставляется как часть Windows 2000. Это COM-объект, который позволяет строить WT-приложения и сценарии на любом отдельно стоящем компьютере или на Web, используя ASP.
- ❑ **XHTML**. XHTML (eXtensible Telephony Markup Language, расширяемый язык телефонной разметки), предложенный компанией Pactolus Communications Software (<http://www.pactolus.com/>), — наверное, наиболее богатый, полный и практичный из представленных диалектов. Особенность XHTML — модульный подход к проблеме. Течение звонка и IVR-взаимодействие это лишь пара его модулей. Как и в CallXML, в основе XHTML лежит модель событий. Сначала определяются события, интересующие нас в данном документе XHTML, а затем внутри документа создаются кусочки XHTML, называемые функциями, которые имеют дело с этими событиями. Гибкость XHTML увеличивают возможности стандартных расширений, позволяя делать такие полезные вещи, как включение фрагментов на Java или C++ и ведение баз данных звонков. И все это делается прямо во время работы телефонной программы. Например, после того как пользователь вводит свой PIN-код на оплаченную услугу, он должен быть проверен по базе данных. XHTML позволяет вести подобные взаимодействия прямо в "течении звонка" (call flow). Телекоммуникации нового поколения — динамичная индустрия. Все время предлагаются новые протоколы, запускается разное оборудование, обсуждаются новые идеи. Создатели XHTML учли эту динамичность и построили свою схему вокруг некоторого функционального ядра, способного расширяться по мере того, как потребности будут меняться. Кто угодно может придумать и добавить новые события в программный сервер и затем ссылаться на них в своих XHTML-документах; правда, все это несколько усложняет язык.

Каждый из этих инструментов может быть использован для построения приложений, но это только одна часть общей картины. Трудной частью по-прежнему является соединение телефона с Web-приложением. Эта часть может оказаться очень дорогой и включать модемы, серверы доступа, теле-

фонные пулы, программное обеспечение, обрабатывающее голосовые сообщения, и связанные технологии. Альтернативой является использование аутсорсинга с помощью компаний типа Voxeo.

Все, что нужно сделать разработчику, используя услуги компании Voxeo, — это связать телефонный номер с URL собственного приложения, в котором обеспечивается генерация текста на языке разметки. Всю инфраструктуру соединения обеспечивает провайдер. Что является самым важным — так это то, что в настоящее время услуга предоставляется совершенно бесплатно.

Все это выглядело бы как предложение с ограниченным сроком годности, если бы их лицензионная политика не была построена на абонентской плате за порт, при которой стоимость аренды сравнима с аутсорсингом Web-сайта.

Для того чтобы сделать свое предложение еще более привлекательным, компания Voxeo предоставляет некоторое количество CallXML и VoiceXML приложений, которые доступны с их сайта и скоро будут доступны на SourceForge. Они также предоставляют некоторое количество VoiceXML и CallXML обучающих программ, включающих примеры по использованию ColdFusion, JSP, ASP и PHP для построения интерактивных приложений. Чтобы закончить набор, они предоставляют объектно-ориентированную среду разработки (Voxeo Designer) для построения WT приложений.

Провайдеры приложений

Демонстрация инструментов компании Voxeo была проведена на конференции Allaire разработчиков. В этой демонстрации поражает простота, с которой можно выполнять нужные действия. Ниже приведена последовательность шагов при создании телефонного приложения.

- ☐ Первый шаг. Регистрация члена общества разработчиков Voxeo. С этого момента главными вещами, занимающими разработчика, являются управление отображением URL и использование инструментов для отладки приложений.
- ☐ URL Mappings позволяет зарегистрировать телефонный номер в одном из регионов США и указать на соответствующий URL. Единственная проблема в том, что как только вы указали на тип используемого на сайте языка (CallXML, VoiceXML, MS-WTE), его уже невозможно изменить для данного номера. Как только номер связан с URL, можно строить приложение на основе этих языков. Теперь самое время запустить приложение. Все, что нам действительно нужно сделать — это взять в руки телефон и опробовать его в действии. При этом можно сэкономить много времени и стоимость звонков, если следовать некоторым простым правилам.
- Проверить XML. Самый простой способ — это пропустить пакет через Internet Explorer и убедиться, что он правильно сформирован.

- Проверить функционирование вашего кода путем исполнения запроса к базе, если это требуется.
- Убедиться, что динамическая генерация также создает корректный XML документ.

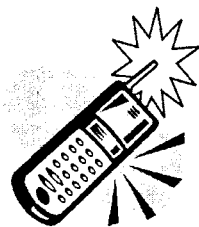
Заключительные замечания

В информационной экономике любой способ доставки существующей информации по новым каналам может стать источником дополнительных доходов. В то время как WAP, WML и весь остальной беспроводной мир выглядят поразительными, старые телефоны используются более чем 1.5 миллиардами людей. Приложения Web-телефонии являются прекрасной возможностью развития существующих Web-приложений в телефонном мире, а такие компании, как Voxeo, могут оказать на этом пути существенную помощь.

Складывается впечатление, что в скором будущем появится большое количество провайдеров, подобных Voxeo, что приведет к тому, что инструменты и оборудование для создания собственных шлюзов будет более дешевым и простым в использовании. Новые платформы для доставки WT-приложений, такие как the IBM WebSphere Voice Server SDK, позволяют разработчикам строить свою инфраструктуру. Их открытые стандарты, похожие на VoiceXML и CallXML, позволят разработчикам строить приложения Web-телефонии и затем мигрировать к будущим платформам очень простым способом.

В последующих главах будут представлены спецификации XML-диалектов для разработки телефонных приложений CallXML и VoiceXML.

Глава 6



Обзор языка VoiceXML

Обзор языка

VoiceXML является декларативным языком, описывающим иерархию элементов, которые определяют способ обработки телефонного вызова, подсказки пользователю относительно возможных методов ввода информации, обработку речи и событий DTMF, а также прямое голосовое обслуживание в соответствии с информацией, полученной от пользователя.

На рис. 6.1 изображена иерархия элементов языка VoiceXML.

Цели языка

VoiceXML является языком программирования, используемым для описания обработки телефонного вызова в интерактивных голосовых приложениях. Конструкции языка VoiceXML обеспечивают ясный и простой способ выполнения следующих действий:

- ☐ проигрывания аудио;
- ☐ распознавание речи и (DTMF) последовательностей;
- ☐ управление процессом обработки телефонного вызова.

VoiceXML является диалектом Extensible Markup Language (XML).

Основные понятия языка XML

XML представляет собой стандартный формат для задания структурированных документов и данных на Web. XML позволяет программистам определять произвольный словарь, формально названный схемой, с использованием стандартного хорошо определенного и просто выполняемого синтаксиса. Одна XML-схема может определять информацию о пользователе, другая может описывать математическое уравнение, а третья может описывать рецепт приготовления шоколадного печенья.

XML легко передается по каналам World Wide Web (WWW) посредством существующих интернет-протоколов, таких как HTTP. Специальные инструменты не требуются для создания таких документов, тем не менее их очень просто писать и модифицировать с использованием существующих программных средств. Все это делает XML идеальным языком для передачи данных между приложениями.

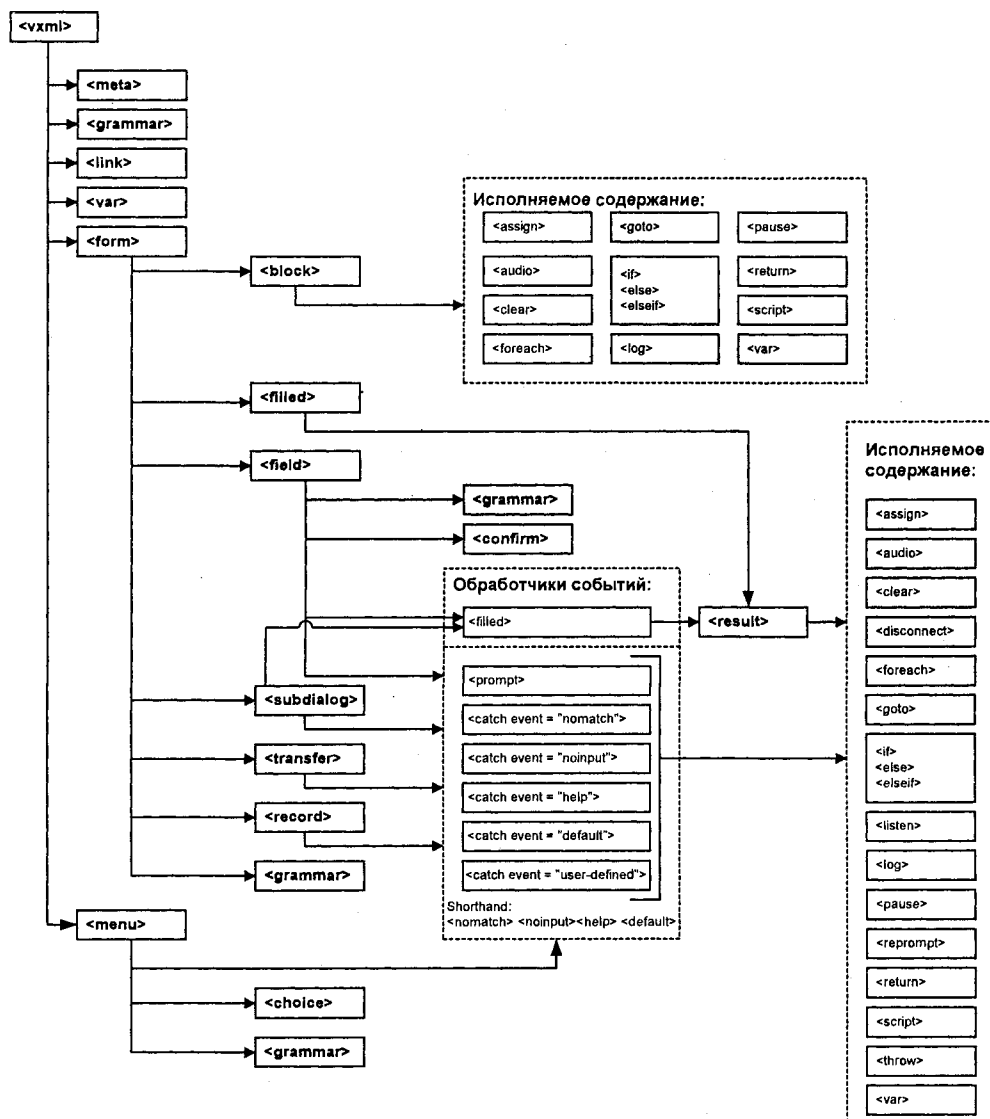


Рис. 6.1. Иерархия элементов языка VoiceXML

Пример XML-документа

XML-документ включает один или более именованных элементов, организованных в иерархический документ. Каждый элемент состоит из открывающегося тега, некоторых данных и закрывающегося тега. Тег состоит из имени с предшествующим знаком "меньше чем" (`<`) и последующим знаком

"больше чем" (>). Для каждого конкретного элемента имя открывающегося тега должно соответствовать имени закрывающего тега. Закрывающий тег идентичен открывающему за исключением того, что после открывающей скобки всегда стоит знак прямого слэша (/). Все теги пишутся маленькими буквами. Несмотря на то, что нижеприведенные примеры написаны маленькими буквами, допустимо применение как больших, так и маленьких букв. Единственным правилом должно быть точное соответствие открывающего и закрывающего тегов.

```
<welcome>Welcome to Tellme University</welcome>
```

Если элемент не содержит данных, открывающий и закрывающий теги могут быть скомбинированы в одном. Пример такого объединения приведен ниже

```
<welcome/>
```

Элемент может содержать произвольное количество атрибутов или не иметь ни одного из них. Атрибут состоит из имени и значения. Значения атрибутов должны содержаться в двойных или одиночных кавычках, например:

```
<salutations>
```

```
  <welcome type="texan">Howdy, partner</welcome>
```

```
  <welcome type="australian">G'day, mate</welcome>
```

```
</salutations>
```

Элементы могут составлять иерархическую структуру с неограниченным количеством уровней вложенности, но только один элемент в документе может быть назначен в качестве корневого. Корневым элементом считается первый элемент документа.

За исключением элемента комментария, который будет описан ниже, все элементы документа должны быть порожденными (подчиненными) элементами корневого элемента.

В нижеприведенном примере <customer> является корневым элементом документа customer с тремя подчиненными элементами. Подчиненный элемент <address> в свою очередь имеет еще 4 порожденных элемента.

```
<customer id="C1234">
```

```
  <lname>Smith</lname>
```

```
  <fname>John</fname>
```

```
  <address type="biz">
```

```
    <street>1310 Villa Street</street>
```

```
    <city>Mountain View</city>
```

```
    <state>CA</state>
```

```
    <zip>94041</zip>
```

```
  </address>
```

```
</customer>
```

XML-комментарии

Комментарии необходимы в любом хорошо оформленном документе. Это особенно справедливо для документов, включающих программный код.

Комментарии помогают понять, как этот код работает. Обычно комментарий начинается с комбинации символов "<!--" и заканчивается комбинацией символов "-->". Комментарии могут также появиться в качестве дочернего элемента XML-документа. Они также могут появиться перед или после корневого документа приложения. Комментарии могут занимать несколько строк, но не могут образовывать вложенные структуры. Следующий пример демонстрирует возможности комментирования документов.

```
<assembly>
  <part>A451</part> <!-- headset -->
  <part>C200</part> <!-- crank -->
  <part>X999</part> <!-- derailleur -->
</assembly>
```

Проверка корректности XML-документа

Схема XML-документа может быть формально определена с использованием Document Type Definition (DTD). DTD определяет элементы и атрибуты, которые разрешены в документе. Следующий DTD формально задает схему вышеприведенного документа customer.

```
<!ELEMENT customer (lname, fname, address+)>
<!ATTLIST customer id ID #REQUIRED>
<!ELEMENT lname (#PCDATA)>
<!ELEMENT fname (#PCDATA)>
<!ELEMENT address (street, city, state, zip)>
<!ATTLIST address type (biz | home) #REQUIRED>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
```

XML-интерпретатор может быть сконфигурирован таким образом, чтобы выполнять проверку корректности XML документа в соответствии с указанным DTD. В этом случае DTD должен быть указан для данного XML документа объявлением DOCTYPE. Объявление DOCTYPE может появиться в верхней части документа XML до появления корневого элемента. В следующем объявлении документ, содержащий корневой элемент <customer>, проверяется в соответствии с DTD, помещенном в файле cust.dtd.

```
<!DOCTYPE customer SYSTEM "cust.dtd">
```

Зарезервированные символы и разделы *CDATA*

XML резервирует некоторые знаки типа " меньше чем " (<), "больше чем" (>) и амперсанд. Чтобы выразить эти знаки в данных, использующих эквивалентный знак:

```
<equation>x &lt; 2</equation>
```

```
<rhyme>Jack &amp; Jill</rhyme>
```

Но данные могут быть защищены от XML-обработки тегом <CDATA>.

```
<comedians>
```

```
<comedian><![CDATA[Abbott & Costello]]></comedian>
```

```
<comedian><![CDATA[Laurel & Hardy]]></comedian>
```

```
<comedian><![CDATA[Amos & Andy]]></comedian>
```

```
</comedians>
```

Как будет показано в дальнейшем, VoiceXML построен на основе базовых концепций и правил, заимствованных у XML, и определяет словарь для описания интерактивных голосовых приложений.

Сравнение XML и HTML

Некоторые разработчики более знакомы с языком HTML, чем с XML. Табл. 6.1 иллюстрирует основные различия в использовании этих языков.

Таблица 6.1. Основные различия XML и HTML

XML	HTML
Может быть использован для определения произвольного словаря при решении любого семейства задач	Определяет конкретный словарь, который используется для создания визуального представления документов
Все теги должны быть закрыты	Некоторые теги, такие как <code>IMG</code> и <code>INPUT</code> не обязательно закрывать
Все атрибуты должны иметь значения	Некоторые атрибуты могут не иметь точного значения
Все значения атрибутов должны быть окружены кавычками	Значения атрибутов заключаются в кавычки только в тех случаях, когда имеют внутри вложенные пробелы
Сущности, кроме <code>&</code> , <code><</code> и <code>></code> , должны быть точно объявлены внешним объявлением как <code>ENTITY</code> в DTD	Большинство сущностей жестко определены внутри интерпретатора
Интерпретатор обычно ничего не прощает. Основные правила обработки тегов и атрибутов строго управляются DTD	Интерпретатор предельно много прощает автору документа. И часто явно забытые теги вставляются в документ по умолчанию

Таблица 6.1 (окончание)

XML	HTML
В тех случаях, когда задан DTD, нераспознанные теги отбрасываются, а процесс интерпретации завершается	Нераспознанные теги игнорируются
Может содержать только один корневой элемент	Следует включать только один корневой элемент (HTML), но интерпретатор может простить отклонение от этого правила, если элемент HTML опущен

Где найти дополнительную информацию?

Нижеприведенные серверы в Интернете могут служить источником дополнительной информации по HTML, XML и VoiceXML:

- ☐ <http://www.w3.org/MarkUp/>
- ☐ <http://www.w3.org/XML/>
- ☐ <http://www.voiceXML.org/>
- ☐ <http://www.xml.com/pub/98/10/guide1.html>

Основы VoiceXML

В предыдущем разделе были представлены основные понятия языка XML как мета- языка, используемого для описания данных. В этом разделе будет представлено описание VoiceXML, диалекта XML, который не только описывает данные (такие как аудиоподсказка), но также описывает процесс обработки телефонного вызова в голосовом приложении.

Структура голосового приложения, созданного с использованием языка VoiceXML

Голосовое приложение, построенное на основе языка VoiceXML, состоит из одного или нескольких документов VoiceXML. VoiceXML документ включает один или несколько диалогов. Каждый диалог определяет состояние вызова. Состояние вызова может быть интерактивным или информационным. В интерактивном состоянии вызова голосовое приложение подсказывает звонящему возможности последующего ответа, и этот ответ либо распознается, либо отбрасывается. Последующие уроки подробно описывают состояния интерактивного вызова. Информативные состояния вызова гораздо проще и описываются здесь для демонстрации базового уровня функциональности.

В минимальном варианте VoiceXML документ состоит из следующих элементов:

- ❑ `<vxml>` — определяет верхний (корневой) элемент документа. Все другие элементы подчиняются прямо или косвенно этому корневому документу;
- ❑ `<form>` — определяет интерактивный или информативный тип диалога, известный как состояние вызова;
- ❑ `<block>` — определяет контейнер для исполняемых элементов;
- ❑ `<audio>` — проигрывает звуковой файл или синтезированную речь (TTS);
- ❑ `<goto>` — указывает навигацию внутри текущего документа или к другому VoiceXML-документу.

Чтобы создать приложение нужно все элементы представить в виде иерархической структуры в соответствии с правилами, определенными в VoiceXML DTD. Ниже приведен пример простейшего голосового приложения:

```
<vxml>
  <form>
    <block>
      <audio>Добро пожаловать в Университет Tellme</audio>
      <goto next="_home"/>
    </block>
  </form>
</vxml>
```

Приложение использует проигрывание синтезированного сообщения и возвращает управление в головное меню платформы Tellme.

Следующее приложение построено на основе предыдущего и состоит из двух диалогов, выполняемых последовательно в заданном в документе порядке.

```
<vxml>
  <form>
    <block>
      <audio>Welcome to Tellme University</audio>
      <pause>1000</pause>
    </block>
  </form>
  <form>
    <block>
      <audio>Thanks for visiting Tellme U</audio>
      <goto next="_home"/>
    </block>
  </form>
</vxml>
```

Описание элемента `<audio>`

В предыдущем примере, элемент `<audio>` используется для проигрывания синтезированной речи (TTS). Однако `<audio>` может быть использован также для поддержки проигрывания предварительно записанного звука. Обычно TTS используется на ранних стадиях создания голосовых приложений и при создании прототипов, использование записанного звука переносит приложение на другой уровень профессионального качества. Все приложения Tellme используют звук, записанный профессионалами.

Для того чтобы проиграть записанный звук, нужно описать атрибут `src` элемента `<audio>` и установить его значение на URL записанного файла:

```
<audio src="welcome.wav">
  Welcome to Tellme University.
</audio>
```

Как видно из текста приложения звуковой файл и TTS могут быть заданы одновременно. В случае, когда записанный звуковой файл не может быть извлечен VoiceXML-браузером, вместо него используется механизм TTS. В то время как профессиональный звук более предпочтителен синтезированному звуку, последний явно предпочтительнее полной тишины.

Управление процессом обработки вызова

Голосовое приложение является набором диалогов. Процесс обработки вызова задает порядок, в котором эти диалоги представляются вызывающему абоненту. По умолчанию, интерпретатор VoiceXML переходит от диалога к диалогу в порядке, в котором они приведены в документе.

Разработчик может также задать другой порядок интерпретации диалогов в документе с использованием элемента `<goto>`, установив значение атрибута `next` на URL нужного документа, или диалог внутри текущего документа. К диалогу можно обратиться через значение атрибута `id`. Поэтому каждый диалог внутри VoiceXML-документа должен быть уникальным. Для того чтобы сослаться на диалог внутри текущего документа, нужно в URL указать `"#"` с последующим значением атрибута `id` элемента `<form>`. Для перехода к другому документу нужно указать в URL абсолютный или относительный путь этого документа, символ и окончательно `id` нужного элемента.

В нижеприведенном примере диалог `"welcome"` в основном VoiceXML-документе выполняет переход к диалогу `"get_course"` в документе `reg.vxml`, который помещен в том же самом каталоге. Диалог `"get_course"` выполняет переход к `"get_details"`. Диалог `"get_details"` переходит к `"farewell"` в документе `main.vxml`. Диалог возвращает управление обратно к главному сервис-меню.

```
<!-- main.vxml -->
```

```
<vxml>
  <form id="welcome">
    <block>
      <goto next="#get_course"/>
    </block>
  </form>
  <form id="farewell">
    <block>
      <goto next="_home"/>
    </block>
  </form>
</vxml>
<!-- reg.vxml -->
<vxml>
  <form id="get_course">
    <block>
      <goto next="#get_details"/>
    </block>
  </form>
  <form id="get_details">
    <block>
      <goto next="main.vxml#farewell"/>
    </block>
  </form>
</vxml>
```

Совет

Атрибут `id` элемента `<form>` используется для целей, аналогичных атрибуту `NAME` в элементе `<anchor>` языка HTML.

Верификация и отладка приложений

Совершенно точно так же как программисты используют компиляторы для проверки синтаксической корректности своего кода, голосовое приложение может использовать VoiceXML Syntax Checker для проверки своего VoiceXML.

После того как ошибки периода компиляции будут обнаружены и исправлены, могут появиться ошибки периода исполнения программы. Программа Tellme Studio Debugger может помочь для поиска последних.

Верификация VoiceXML-документов

Как уже отмечалось, XML документ может быть протестирован на корректность указанному в нем DTD. Документ DTD определяет правила, которым должны удовлетворять элементы интерпретируемого документа, отношения между главными и порожденными элементами документа и атрибуты самих элементов. В ряде случаев также допускается указание разрешенных значений атрибутов, описанных в DTD.

DTD для голосовых приложений, запускаемых с помощью Tellme Studio, находится по адресу <http://resources.tellme.com/toolbox/vxml-tellme.dtd>.

Для того чтобы обеспечить для Tellme Studio Syntax Checker проверку корректности голосового приложения, необходимо добавить следующее объявление DOCTYPE до корневого элемента документа VoiceXML:

```
<!DOCTYPE vxml PUBLIC
"-//Tellme Networks//Voice Markup Language 1.0//EN"
"http://resources.tellme.com/toolbox/vxml-tellme.dtd">
```

Для того чтобы запустить Syntax Checker, убедитесь, что страничка MyStudio описывает URL нужного документа VoiceXML и выберите пункт **<check syntax>** главного меню.

Ошибки времени исполнения

Опытные программисты знают, что даже синтаксически правильные программы могут при запуске выдавать неожиданные результаты. Это справедливо и для приложений на VoiceXML.

Программа Tellme Studio Debug Log может помочь увидеть, что случилось с приложением при его выполнении. При этом доступна следующая информация:

- ☐ call flow — подробности исполнения процесса интерпретации документа VoiceXML, такие как навигация и исполнение блока инструкций. В процессе интерпретации приложения VoiceXML интерпретатор отчитывается о тех элементах, с которыми он встречается;
- ☐ errors — детальный синтаксис ошибок в VoiceXML, грамматиках и сценариях JavaScript, ошибок вызова документов, необработанных событий и ошибок JavaScript времени исполнения;
- ☐ events — детальная спецификация событий, связанных с распознаванием образов и событий, определенных пользователем;
- ☐ field information — подробности информации, полученной от пользователя, в некотором поле в процессе диалога;
- ☐ variable information — детали чтения или записи переменных, заданных пользователем, или переменных сеанса. Анализируя log-файл отладки,

доступный в MyStudio, можно быстро определить источник ошибки и исправить ее.

Управление выводом отладочной информации

Прежде чем начать использовать интерактивный отладчик, программисты, работающие на языке С, традиционно добавляли в текст приложений предложения `printf`, для того чтобы направить отладочную информацию на стандартное устройство вывода, обычно консоль. Это помогает понять, что происходит в отлаживаемом коде. Даже сегодня Web-программисты включают вызовы промежуточных информаторов в свои клиентские сценарии на JavaScript. Разработчики голосовых приложений используют аналогичные методы отладки. Эта дополнительная информация выводится в отладочный log-файл вместе с данными, отправляемыми туда по умолчанию с сервисной платформы. Интерпретатор Tellme VoiceXML обеспечивает разработчику два механизма — использование элемента `<log>` и функции `vxmlllog`. Элемент `<log>` может быть использован внутри любого исполняемого блока. Функция `vxmlllog` может быть использована внутри любого блока JavaScript. Следующий пример демонстрирует использование обоих подходов.

```
<vxml>
<meta name='scoping' content='new' />
<script type="text/javascript"><![CDATA[
  function GetAd(nSeed, nMaxAds)
  {
    var nAd = Math.round(Math.random(nSeed)*100) %
    nMaxAds;
    var sAd = "ad" + nAd + ".wav";
    vxmlllog("GetAd(" + nSeed + ") returning '" + sAd +
    "'");
    return sAd;
  }
}]>
</script>
<form id="Init">
  <block>
    <!-- get random ad, and play it -->
    <var name="sOpeningAd" expr="GetAd(1, 10)"/>
    <log>Queueing {sOpeningAd}...</log>
    <audio expr="'ui/ads/' + sOpeningAd"/>
  </block>
</form>
<!-- fill in application-specific dialogs here -->
```

```
<form id="Term">
  <block>
    <goto next="_home"/>
  </block>
</form>
</vxml>
```

Приведенный пример определяет функцию GetAd, которая создает случайное рекламируемое имя. Прежде чем вернуть это имя, функция выводит его отладочный log-файл.

Первый диалог вызывает функцию GetAd и сохраняет результаты в переменной. Затем код выводит это имя в отладочный log-файл, используя элемент `<log>`, и загружает рекламный ролик в очередь на проигрывание.

Следующий диалог "Term" просто передает управление в главное меню системы.

Прослушивание голосового сообщения от входящего телефонного вызова

Этот раздел описывает, как строить состояния интерактивного вызова, в котором голосовое приложение подсказывает вызывающему абоненту и получает от него ответ.

Проверяя ответ, имеется возможность принимать решения о том, куда дальше передавать управление. При этом также имеется возможность выполнить некоторый внешний процесс.

Необходимые параметры для создания интерактивности

Как было сказано ранее, тег `<form>` является основным элементом при построении интерактивных или информативных приложений, однако, для того, чтобы вызов был полностью интерактивен, необходимо указать ряд параметров:

- ☐ `<field>` — задает место, в котором нужно хранить ответы вызывающего абонента;
- ☐ `<prompt>` — посвящен тому, что нужно сказать абоненту;
- ☐ `<grammar>` — задает, что конкретно абоненту разрешено сказать;
- ☐ `<response-handling code>` — определяет, что делать в ответ на правильный ввод информации от абонента;
- ☐ `<error-handling code>` — задает, что делать при получении некорректного ввода.

Язык VoiceXML определяет один или несколько элементов для представления вышеприведенной информации в приложении. Они детально описаны ниже.

Создание резерва памяти для ответа звонящего

В интерактивном диалоге обычно звонящему что-то сначала говорят.

Если его ответы понимаются системой распознавания речи, возвращается значение из грамматики, которая доступна программисту в элементе `<field>` из тега `<form>`.

```
<form id="class_yes_no">
  <field name="class_by_name">
    </field>
  </form>
```

Значение атрибута `name` определяет пространство памяти в виде некоторой переменной, значением которой будет возвращаемое значение грамматики.

Подсказка звонящему перед вводом информации

В предыдущем уроке было изучено, что аудиоэлемент может быть использован для проигрывания записанного звукового файла или синтезированной речи.

В случае интерактивного диалога, аудиоэлемент должен содержаться внутри элемента подсказки. Интерпретатор VoiceXML использует элемент `<prompt>` в качестве указателя, куда нужно вернуться в ответ на возникновение события, которое случится при встрече элемента `<reprompt>`. Сам элемент `<reprompt>` будет описан позднее в разделе, посвященном обработке ошибок.

```
<form id="class_yes_no">
  <field name="class_by_name">
    <prompt>
      <audio>Do you know the name of the class you want?
        Say yes or no.</audio>
    </prompt>
  </field>
</form>
```

В соответствии со спецификацией VoiceXML, диалог может содержать несколько полей ввода. Элемент `<prompt>`, содержащийся внутри поля `field`, связывает подсказку с конкретным полем. В настоящее время платформа Tellme поддерживает только одно поле для одного диалога.

Ограничение ввода

Эффективное распознавание речи требует, чтобы говорящий выделял свою речь определенными рамками, которые обычно заданы в виде грамматик.

Грамматики определяются внутри VoiceXML документа посредством тегов `<grammar>`.

Синтаксис грамматики

Несмотря на то, что текущая спецификация VoiceXML определяет элемент `<grammar>`, его синтаксис оставлен на усмотрение поставщика интерпретатора VoiceXML.

Платформа Tellme поддерживает Grammar Specification Language (GSL) для своего синтаксиса грамматики. Дополнительные языки описания грамматик будут поддерживаться в будущих версиях платформы.

Синтаксис грамматики лучше всего описывается на простых примерах. Например, нижеприведенная грамматика задает набор разрешенных реакций в ответ на вопрос, который имеет ответы "да" и "нет".

```
[
{yes sure yeah okay check affirmative (ten four)}
  {<option "yes">}
{no nope negative}
  {<option "no">}
]
```

Для того чтобы ответить "да" абонент может сказать "yes", "sure", "yeah", "okay", "check", "affirmative" или "ten four". Для ответа "нет" он может сказать "no", "nope" или "negative". Заметим, что все слова внутри грамматики пишутся с маленькой буквы. Прописные буквы зарезервированы для определения имени и ссылки на подграмматику. Подграмматика — это строительный блок, определяющий сложную грамматику.

Когда абонент высказывает ответ, который может быть одним из этих двух подмножеств, механизм распознавания голоса возвращает значение, связанное с действием, которое соответствует этому высказыванию. Платформа Tellme устанавливает значение атрибута `name` активного поля в активной форме на значение, возвращаемое грамматикой.

В этом событии позитивного высказывания абонента строка "yes" хранится в атрибуте `name` активного поля в активном диалоге. В случае негативного высказывания сохраняется строка "no".

Квадратные скобки определяют условие "или". То есть внутри набора квадратных скобок только одно слово или фраза может быть высказана. В следующем примере абонент может сказать "bail" или "drop me", чтобы отказаться от курса. Он также может сказать "enroll me" или "sign me up", чтобы подписаться на курс.

```
[
  {bail (drop me)} {<option "drop">}
]
```

```
{(enroll me) (sign me up)} {<option "enroll">}  
}
```

Скобки определяют условие "и". То есть внутри круглых скобок все произнесенные слова или фразы произносятся для того, чтобы их вместе распознавали. Например, абонент должен сказать оба слова "enroll" и "me" или слова "sign," "me" и "up" для того, чтобы предназначенное действие начало выполняться.

Условия "и" и "или" могут комбинироваться и вкладываться один в другой. Нижеприведенный пример показывает как условия "и" вкладываются внутрь условий "или".

```
{([enroll add] me) (sign me up)}
```

После компиляции фрагмент будет генерировать следующие предложения:

```
enroll me  
add me  
sign me up
```

Вариации ответа абонента поддерживаются посредством использования дополнительных операторов языка.

Эти операторы могут быть предпосланы определенным словам или фразам для указания тех слов, которые могут встречаться ноль или один раз (?), ноль или более раз (*) и один или более раз (+).

Символ "?"

Символ "?" используется перед словом или фразой, когда они полностью необязательны, то есть, от абонента не требуется говорить их. Такие слова или фразы обычно являются модификаторами.

В следующем примере слово "me" полностью вспомогательное, потому что абонент может его говорить или нет. Высказывание абонента не должно отбрасываться просто потому, что он не произнес слово "me".

```
enroll ?me  
sign ?me up
```

Символ "*"

Символ "*" используется перед словом или фразой, когда они необязательны, но не могут встречаться только один раз. Аналогично "?", ответ абонента не должен отбрасываться только потому, что кто-то не сказал слово или фразу более одного раза.

В нижеследующем примере грамматика описывает поведение абонента, который действительно хочет подписаться на курс. Он может сказать слово "definitely" столько раз, сколько ему нравится или не сказать его вовсе.

```
*definitely sign ?me up
```

Символ "+"

Символ "+" используется перед словом или фразой, когда они могут встречаться по меньшей мере один раз, но могут появиться более чем единожды. Обычно не используется, но реализован для полноты поддерживаемых операций.

Внимание!

Использование вышеописанных операторов может плохо сказаться на производительности используемого механизма распознавания, так как она обычно падает с увеличением размера используемой грамматики.

Разрешение ввода DTMF в грамматике

Традиционные системы IVR действуют, инструктируя абонента, какие клавиши на телефоне, поддерживающем тоновый режим, нужно нажимать. Этот режим известен также как ввод DTMF-последовательностей. GSL поддерживает такой ответ. Для того чтобы связать клавишу тонового телефона с опцией меню в грамматике, просто указывается значение с литерной строкой "dtmf-", заключенной в кавычки. Следующий пример разрешает абоненту набрать "d", "e," и "v" или сказать слово "development" для того чтобы вернуть слово "development" из грамматики.

```
[ (dtmf-3 dtmf-3 dtmf-8) development ]
<option "development">
```

Для того чтобы описать клавишу "звездочка" в грамматике, укажите "dtmf-*":

```
[ (dtmf-* dtmf-*) quit ] <option "quit">
```

Комментирование грамматики

Грамматики могут быть достаточно сложными. Для того чтобы помочь разработчикам при сопровождении продукта, желательно использовать комментарии, которые задаются в GSL следующим образом:

1. Комментарии начинаются с двоеточия и завершаются вместе с символом конца строки.
2. Многострочные комментарии не поддерживаются в языке GSL. Ниже приведен пример комментария.

```
; grammar used after a specific class is selected
[
  [bail (drop me)] {<option "drop">}
  [(enroll me) (sign me up)] {<option "enroll">}
]
```

Интеграция грамматик в VoiceXML

Так как синтаксис GSL использует символы, которые зарезервированы в XML ("**<**", "**>**"), встроенные грамматики должны быть защищены от XML анализатора включением их в секцию CDATA. Нижеприведенный пример показывает использование встроенной грамматики.

```
<form>
  <field>
    <grammar>
      <![CDATA[
        [
          [bail (drop me)] {<option "drop">}
          [(enroll me) (sign me up)] {<option "enroll">}
        ]
      ]]>
    </grammar>
  </field>
</form>
```

Грамматики могут быть определены внутри документа VoiceXML. Они также могут быть определены как внешний документ и вмонтированы в VoiceXML-документ с использованием атрибута `src` элемента `<grammar>`. В этом случае можно получить следующие преимущества.

- ☐ Грамматики могут создаваться динамически (например, с использованием CGI-сценариев или Active Server Pages).
- ☐ Грамматики могут быть использованы в разных приложениях и диалогах.
- ☐ Грамматики могут быть заменены или перемещены без изменения текстов голосового приложения.

Нижеприведенный диалог ссылается на грамматику, размещенную в `yesno.gsl`, как к файлу внешней грамматики, размещенному в каталоге грамматик. URL для `yesno.gsl` является относительным адресом этого каталога, в котором размещен документ VoiceXML, содержащий ссылающийся на грамматику диалог.

Интерпретатор VoiceXML использует протокол HTTP для извлечения грамматики тем же самым методом, каким извлекается сам VoiceXML-документ.

```
<form>
  <field>
    <grammar src="grammars/yesno.gsl"/>
  </field>
</form>
```

Введение в события VoiceXML

Событие является важнейшим понятием, описывающим поведение системы или приложения. В настольном приложении события возникают как результат действий пользователя компьютера (такие как нажатие клавиш или щелчок мыши). Событие также возникает из-за активности программ системы, такие как выключение системы или даже ее отказ. При возникновении этих событий система дает возможность приложениям отреагировать на их появление.

Обычно приложения отвечают на события выполнением некоторой полезной работы. Этот процесс называется обработкой ошибочных ситуаций. Аналогичные действия выполняются разработчиками Web-приложений при создании сценариев, работающих на стороне клиента, когда тот взаимодействует с Web-браузером.

Разработчики голосовых приложений имеют возможность обработки событий, используя специальные теги описания разрешенных событий. Некоторые теги описывают обработку специфических часто возникающих событий. Элемент `<catch>` является общим методом такой обработки, который будет описан позднее.

Теги VoiceXML для обработки общих событий в голосовом приложении включают:

- ❑ `filled` — обрабатывает правильное высказывание от абонента;
- ❑ `nomatch` — обрабатывает неверное высказывание абонента;
- ❑ `noinput` — обрабатывает тишину внутри состояния взаимодействия с абонентом;
- ❑ `help` — обрабатывает запрос на получение помощи.

Обработка корректного ввода

Если высказывание вызывающего ограничено рамками активной грамматики, интерпретатор VoiceXML подхватывает заполненное событие, и программа приступает к обработке заполненного элемента внутри активного поля формы.

Если активное поле не содержит заполненного элемента, программа переходит к следующему заполненному элементу. Внутри заполненного элемента программист может определить высказывание звонящего, проверяя значение переменной, связанной с атрибутом `name` активного поля. Значение сохраненной переменной будет содержать значение, которое было подготовлено активной грамматикой.

Для малых грамматик простейший способ обработки корректного высказывания внутри заполненного элемента заключается в том, чтобы каждый заполненный элемент имел специальное значение, возвращаемое граммати-

кой. Значение атрибута `name` элемента `<result>` соответствует значению, возвращаемому грамматикой.

В следующем примере диалога приложение запрашивает звонящего имя интересующего класса. Если он говорит "yes" или "o'kay", приложение переходит в состояние, которое запрашивает имя этого класса и получает класс по имени `get_class_by_name`. В случае ответа "no" или "nope," приложение переходит в состояние вызова, которое запрашивает область интереса, получает `get_class_by_subject`.

```
<form id="class_yes_no">
  <field name="class_by_name">
    <prompt>
      <audio>Do you know...? Say yes or no.</audio>
    </prompt>
    <grammar>
      <![CDATA[
        [
          [yes okay] {<option "yes">}
          [no nope] {<option "no">}
        ]
      ]]>
    </grammar>
  </field>
  <filled>
    <result name="yes">
      <goto next="#get_class_by_name"/>
    <result name="no">
      <goto next="#get_class_by_subject"/>
    </result>
  </filled>
</form>
```

Для больших грамматик, таких как только что определенная (задает все акции на NASDAQ, главные города США или курсы конкретного университета), элемент `<result>` является непрактичным. В этом случае более вероятно, что возвращаемые данные будут храниться в переменной, а приложение будет выполнять навигацию к следующему состоянию, независимо от полученных данных.

Так как данные поддерживаются приложением в форме строки, продвинутые разработчики могут быть наняты для динамической модификации процесса обработки звонка. В приложении, управляемом данными, данные, помещенные в активное поле, могут быть отправлены к HTTP-серверу,

который бы в свою очередь выполнил просмотр базы данных и динамически сгенерировал диалог VoiceXML.

Обработка некорректного ввода

В идеале каждый диалог в приложении может заставить абонента сказать что-то из активной грамматики. Содержание заполненного элемента тогда бы исполнялось, а интерпретатор VoiceXML переводил бы программу в подходящее состояние вызова. К сожалению, это не всегда возможно. Иногда звонящий говорит что-то такое, что находится за пределами грамматики или не говорит ничего.

В любом случае вызывающий не может быть распознан адекватно. Если он не может запомнить ключевое слово, которое перевело бы его в главное меню, он почти всегда не будет распознан корректно.

К счастью, VoiceXML дает возможность программисту обработать эту ситуацию с помощью обработчиков "nomatch" и "noinput".

Обработка ввода, не отображенного в грамматике

Когда говорящий говорит что-то за пределами грамматики, интерпретатор VoiceXML обрабатывает событие nomatch. Приложение в этом случае имеет возможность обработать это событие, выполняя <nomatch> уровня формы или уровня поля. Обработчик может принять следующие формы.

```
<nomatch>
  <!-- additional coding logic goes here -->
</nomatch>

<catch event="nomatch">
  <!-- additional coding logic goes here -->
</catch>
```

Поведение каждого из этих элементов идентично. Последнее имеет более общий синтаксис для обработки любого события, подхваченного VoiceXML интерпретатором.

Для того чтобы помочь звонящему получить состояние диалога через состояние вызова, необходимо обеспечить обратную связь для оповещения о неспособности системы понять то, что было сказано, чтобы дать звонящему шанс попытаться сказать фразу снова.

Следующий пример демонстрирует это.

```
<field name="what_todo">
  <prompt>
    <audio>Would you like to enroll, drop,
      or learn more about the class?</audio>
  </prompt>
```

```

<grammar src="active_class.gsl" />
  <nomatch>
    <audio>I'm sorry. I didn't understand that.</audio>
    <reprompt/>
  </nomatch>
</field>

```

Вот что при этом происходит.

- ❑ Приложение проигрывает подсказку.
- ❑ Звонящий что-то говорит за пределами грамматики "active_class".
- ❑ Приложение говорит: "I'm sorry. I didn't understand that."
- ❑ Приложение заново проигрывает звонящему первоначальную подсказку, предоставляя возможность нового ответа.

Этот код будет работать в том случае, когда ответ звонящего успешно отделен от фоновых шумов. В других ситуациях отсутствия совпадения вызывающий был бы отделен от диалога до тех пор, пока он не повесит трубку.

Вместо того чтобы заново проигрывать первоначальную подсказку, обработчик несоответствий может обеспечить полностью новую подсказку, в точности описывающую сложившуюся ситуацию. Вместо того чтобы использовать элемент <reprompt> внутри <nomatch>, используйте элемент <listen>. Этот элемент инструктирует браузер VoiceXML ожидать сообщения от звонящего без проигрывания подсказки в активном поле. Следующий пример демонстрирует использование элемента <listen> в обработчике <nomatch>.

```

<field name="what_todo">
  <prompt>
    <audio>Would you like to enroll, drop, or learn more about the
class?</audio>
  </prompt>
  <grammar src="active_class.gsl" />
  <nomatch>
    <audio>I'm sorry. I didn't understand that.
To learn more about the class, please say tell me more.
<pause>300</pause>
To enroll in the class, please say enroll.
<pause>300</pause>
To drop the class, please say drop.
<pause>300</pause>
  </audio>
  <listen/>
</nomatch>
</field>

```

Вот что при этом происходит:

- ❑ Приложение проигрывает подсказку.
- ❑ Вызывающий что-то говорит из области, не включенной в грамматику.
- ❑ Приложение говорит: "I'm sorry. I didn't understand that. To learn more about the class, please say "tell me more". To enroll in the class, please say "enroll". To drop the class, please say "drop".
- ❑ Приложение дает звонящему другую возможность ответа.

Обработка события "отсутствие ввода"

В случае, когда звонящий не говорит ничего, VoiceXML-интерпретатор запускает событие `noinput`. Это приложение имеет возможность обрабатывать событие на уровне поля и на уровне формы. Описываемый обработчик может принимать одну из следующих форм.

```
<noinput>
  <!--здесь возможно дополнительное кодирование логики -->
</noinput>
<catch event="noinput">
  <!--здесь возможно дополнительное кодирование логики -->
</catch>
```

Поведение каждого из этих элементов идентично.

Последний вариант имеет более общую форму для обработки событий, создаваемых VoiceXML-интерпретатором. Обрабатывать данное событие можно аналогично тому, как обрабатывается событие `nomatch` из предыдущего раздела. Ниже приведен пример.

```
<field name="what_todo">
  <prompt>
    <audio>Would you like to enroll, drop,
or learn more about the class?</audio>
  </prompt>
  <grammar src="active_class.gsl" />
  <noinput>
    <audio>I'm sorry. I didn't here you.
To learn more about the class, please say tell me more.
<pause>300</pause>
To enroll in the class, please say enroll.
<pause>300</pause>
To drop the class, please say drop.
<pause>300</pause>
```

```

</audio>
<listen/>
</noinput>
</field>

```

Обработка события *help*

Если звонящий испытывает затруднения при работе с приложением, он может сказать слово "help". В этом случае VoiceXML-интерпретатор порождает событие *help*, которое голосовое приложение может подхватить на уровне поля или на уровне формы с использованием элемента *<help>*. Для того чтобы обеспечить поддержку *help* приложением, необходимо сделать следующее:

- ❑ добавить ключевое слово "help" в грамматику диалога. Если помощь предусмотрена в приложении, можно поместить это слово в корневой документ приложения;
- ❑ добавить обработчик *<help>* в каждое меню взаимодействия. При разработке приложения нужно добавить этот обработчик в корневой документ для целей отладки. Обработчик должен посылать сообщение для указания в отладочном log-файле.

Эта помощь будет обрабатываться общим обработчиком, а не специфическим модулем.

Если обработчик *<help>* не выполняется, интерпретатор VoiceXML выполняет обработчик *<filled>* для активного диалога. Ниже приведен пример.

```

<field name="what_todo">
  <prompt>
    <audio>Would you like to enroll, drop,
or learn more about the class?</audio>
  </prompt>
  <grammar src="active_class.gsl" />
  <grammar><![CDATA[
[
[help (?dtmf-star dtmf=0)] {<option "help">}
]
]]></grammar>
  <help>
    <audio>

```

To learn more about the class, say tell me more.

```
<pause>300</pause>
```

To enroll in the class, say enroll.

```

<pause>300</pause>
To drop the class, say drop.
<pause>300</pause>
  </audio>
  <reprompt/>
</help>
</field>

```

Обработка направляющих подсказок

Как минимум, необходимо выполнить обработку событий `<nomatch>` и `<noinput>`, чтобы проинформировать вызывающего, почему распознавание может не состояться, и попросить повторить попытку. С каждой успешной попыткой отключения при заполнении активного поля, вам подсказывают добавить дополнительные детали или альтернативное высказывание, чтобы помочь пользователю продолжить диалог.

Модифицируя уровень детализации, VoiceXML позволяет использовать несколько элементов `<prompt>`, `<noinput>`, `<nomatch>` и `<help>` внутри одного поля. Если, например, задано два обработчика `<noinput>` для поля, первый обработчик будет исполняться, когда событие `noinput` будет поймано в поле первый раз, второй обработчик будет исполняться при втором появлении события `noinput`, и так далее. Если число событий определенного вида превышает число активных обработчиков конкретного типа, интерпретатор VoiceXML просто заново исполняет программу последнего обработчика. Если пользователь направляется на другой диалог и позднее возвращается обратно, интерпретатор VoiceXML начинает с первого обработчика в порядке, заданном в документе.

Ниже приведен иллюстрирующий пример с несколькими обработчиками `<noinput>`.

```

<field name="what_todo">
  <prompt>
    <audio>Would you like to enroll, drop,
or learn more about the class?</audio>
  </prompt>
  <grammar src="active_class.gsl" />
  <noinput> <!-- 1 -->
    <audio>I'm sorry. I didn't here you.</audio>
  <reprompt/>
</noinput>
  <noinput> <!-- 2 -->
    <audio>I'm sorry. I still didn't here you.
To learn more about the class, please say

```

```
tell me more.  
<pause>300</pause>  
To enroll in the class, please say enroll.  
<pause>300</pause>  
To drop the class, please say drop.  
<pause>300</pause>  
</audio>  
<listen/>  
</noinput>  
</field>
```

Грамматики

Обычно грамматики являются большим компонентом любого голосового приложения. Грамматики определяют словарь, используемый звонящими. В этом уроке будет представлено введение и описаны некоторые методы построения более сложных и многократно используемых грамматик.

Описание грамматик

Грамматики могут быть описаны в нескольких местах голосового приложения:

- ☐ как порожденный элемент элемента `<vxml>`;
- ☐ как порожденный элемент элемента `<form>`;
- ☐ как порожденный элемент элемента `<field>`;
- ☐ как порожденный элемент элемента `<link>`.

Определение грамматики как порожденного элемента верхнего уровня, такого как `<vxml>`, приводит к тому, что грамматика наследуется элементами нижнего уровня. Когда грамматика описана, как порожденный элемент поля, она является активной только при обработке данных данного поля.

Обычно разработчики приложений определяют универсальные команды в грамматике, размещенной в качестве порожденного элемента корневого документа.

Поскольку диалоги содержат только одно поле, нетипично задавать грамматики уровня диалога.

В следующем примере грамматики `"document_grammar"`, `"dialog1_grammar"` и `"field1_grammar"` являются активными, когда система распознавания собирает информацию для поля `"field1"`, а грамматики `"dialog2_grammar"` и `"field2_grammar"` являются активными, когда система распознает информацию для поля `"field2"`.

```
<vxml>
  <grammar name="document_grammar" />
  <form name="dialog1">
    <grammar name="dialog1_grammar" />
    <field name="field1">
      <prompt/>
      <grammar name="field1_grammar" />
    </field>
  </form>
  <form name="dialog2">
    <grammar name="dialog2_grammar" />
    <field name="field2">
      <prompt/>
      <grammar name="field2_grammar" />
    </field>
  </form>
</vxml>
```

Повторное использование грамматик Tellme

Использование платформы Tellme определяет большой набор грамматик, который может быть повторно использован в голосовых приложениях. Эти грамматики применимы к широкому классу различных приложений и протестированы так, чтобы разработчик мог сфокусироваться на других аспектах диалога. Полный список библиотеки грамматик можно найти по адресу <http://studio.tellme.com/library/grammar/>.

Для того чтобы сослаться на любую из этих грамматик, необходимо просто указать ее имя. Например, чтобы сослаться на грамматику YES_NO, нужно просто создать элемент `<grammar>` следующего типа:

```
<grammar>YES_NO</grammar>
```

Для того чтобы написать код обработки диалога, необходимо знать диапазон значений, возвращаемых грамматике. Эта информация доступна на Web-странице, связанной с грамматикой, включенной в библиотеку Grammar Library. Грамматика YES_NO возвращает "yes", если пользователь сказал положительно распознанный текст и "no" при отрицательном высказывании.

Переменные

Переменные являются базовыми компонентами любого языка программирования. Переменные обеспечивают программисту механизм создания и хранения данных заданного типа, который позволяет обращаться к этим

данным по имени. Основные типы данных включают строковые, числовые и булевы значения. Большинство языков программирования поддерживают создание наборов или списков значений одного типа, известных как массивы. Более сложные языки разрешают программисту создавать новые типы данных, основываясь на группировании данных разных типов. Эти типы известны как структуры.

В отличие от детального кодирования строк и чисел в программе, переменные дают полезную альтернативу, которая помогает увеличить повторное использование данных в программе.

Язык VoiceXML поддерживает объявление, присваивание и извлечение значений переменных.

Объявление переменных

Спецификация VoiceXML требует, чтобы переменные были объявлены до того момента, когда они будут использоваться.

Чтобы объявить переменную, нужно использовать элемент `<var>` или установить атрибут `name` элемента `<field>`. Элемент `<var>` поддерживает два атрибута: `name` и `expr`. Атрибут `name` используется для именования переменной, а атрибут `expr` является дополнительным для назначения начального значения переменной. Использование этих элементов показано в приведенном примере.

```
<var name="iStudentID"/>
<var name="bLoggedIn" expr="false"/>
<var name="sGreeting" expr="'Hi, there.'"/>
```

В последнем объявлении атрибут `expr` включает два набора кавычек. Внешние кавычки окружают значение атрибута. Содержание внутри этих кавычек, включающее внутренние кавычки, передается интерпретатору JavaScript. В этом случае внутренние кавычки указывают, что значение является строкой символов.

Другой пример назначает переменной некоторое значение.

```
<var name="sGreeting" expr="'Hello, '"/>
<var name="sFullGreeting"
expr="sGreeting + 'Mr. President'"/>
```

Для всех тегов языка VoiceXML, которые поддерживают атрибут `expr` или `cond`, значение должно быть правильным выражением JavaScript. Программисты, не знакомые с JavaScript, должны ознакомиться со спецификацией ECMAScript или с другой открытой информацией.

В документе VoiceXML переменные инициализируются в порядке, указанном в тексте. Это позволяет ссылаться на переменные, которые уже были объявлены в предыдущей части документа. Ниже приведен пример.

```
<!-- relative location of audio files -->
<var name="audio_dir" expr="'ui/misc/'"/>
<!-- name of audio file containing welcome prompt -->
<var name="welcome_wav" expr="'welcome.wav'"/>
<var name="audio_path"
expr="audio_dir+welcome_wav"/>
```

Объявление переменных с использованием элемента *<field>*

Атрибут `name` определяет переменную, которая хранит результат распознавания. Для того чтобы получить доступ к значению этой переменной, нужно присвоить ее значение другой переменной с областью действия всего документа. Для того чтобы получить доступ к переменной из другого документа того же самого приложения, нужно присвоить ее значение другой переменной с областью действия приложения.

Присваивание значений переменным

После объявления переменной, с ней можно работать, присваивая ей значения и обращаясь к ним. Чтобы изменить значение, нужно использовать элемент `<assign>` внутри блока JavaScript.

```
<var name="sCurrentState" expr="'welcome'"/>
<assign name="sCurrentState" expr="'enrolling'"/>
<script type="text/javascript">
  sCurrentState = "dropping";
</script>
```

В то время как элемент `<var>` разрешен в качестве порожденного для элементов `<vxml>` и `<form>` для поддержки создания переменных уровня документа и диалога, элемент `<assign>` поддерживается только внутри исполняемого содержания. Исполняемое содержание возникает внутри следующих элементов `<block>`, `<filled>`, `<result>`, `<catch>`, `<default>`, `<help>`, `<noinput>`, `<nomatch>`, `<prompt>`, `<foreach>` и `<if>`.

Описание области действия

Область действия переменных задает время их жизни в приложении VoiceXML.

VoiceXML использует пять областей действия переменных.

- `session` — область действия для режима чтения для переменных, объявленных интерпретатором VoiceXML, в рамках текущего сеанса связи. Переменные сеанса действуют в течение времени сеанса.

- ❑ `application` — область действия `read/write` включает глобальные переменные. Они должны быть объявлены в корневом документе приложения.
- ❑ `document` — область действия `read/write` включает переменные текущего документа приложения. Они должны быть объявлены в текущем документе приложения. Эти переменные иницииируются каждый раз, когда интерпретатор загружает документ, в котором объявлена переменная, и имеют область действия внутри всех диалогов данного документа.
- ❑ `dialog` — переменные уровня диалога, в котором они объявлены. Чтобы создать переменную уровня диалога, нужно объявить ее как порожденный тег желательного элемента `<form>`.
- ❑ `anonymous` — область действия `read/write` включает переменные блока. Они должны быть объявлены в блоке приложения, для которого они действуют.

Интерпретатор VoiceXML инициализирует переменные при каждом входе в блок. При выходе из блока интерпретатор разрушает значение переменной.

Системные переменные сеанса

- ❑ `session.error.code` — хранит числовой идентификатор ошибки.
- ❑ `session.error.text` — хранит текстовое описание ошибки.
- ❑ `session.error.state` — хранит ID элементов `<field>`, `<block>` или `<form>`, в которых возникла ошибка.
- ❑ `session.telephone.ani` — хранит Automatic Number Identification вызывающего телефона. Эта информация доступна только тогда, когда включен сервис автоматического определения номера.
- ❑ `session.telephone.dnis` — хранит Dialed Number Identification Service.
- ❑ `session.telephone.iidigits` — хранит Information Indicator Digit, информация о линии, по которой звонят, такая как вызов с таксофона, сотовый сервис, специализированный оператор или тюремный телефон. Следует заметить, что в настоящее время платформа Tellme не поддерживает этот сервис.
- ❑ `session.telephone.uui` — хранит User to User Information. Следует заметить, что в настоящее время платформа Tellme не поддерживает это сервис.

Обеспечение нового поведения области действия

Поведением области действия, описанным в этом документе, является дополнение стандарта, которое было введено для платформы Tellme 1 ноября 2000 года.

```
<meta name="scoping" content="new"/>
```

Перекрывающиеся области действия переменных и деактивация переменных

В языке VoiceXML области действия переменных организованы в иерархию. Время жизни переменных некоторых областей действия перекрываются с временем жизни переменных из других областей действия.

- ☐ Переменные сеанса и переменные приложения одновременно сосуществуют с переменными, объявленными в документе, на уровне документа, диалога и формы.
- ☐ Переменные документа одновременно сосуществуют с переменными, объявленными в диалоге, блоке и форме.
- ☐ Переменные диалога одновременно сосуществуют с переменными блока.

Следующий пример демонстрирует, как эти правила влияют на состояние программы.

Выполнению программы предшествует комментарий:

"var1 существует в трех областях — документ, диалог и блок — с тремя различными значениями 1, 3 и 5 соответственно. В этом случае элемент <assign> обращается к var1, объявленной в блоке, потому что var1 объявлена в области документа и области диалога."

В соответствии с комментарием переменная var1 существует только в двух областях, документ и диалог, несмотря на то, что она доступна в анонимном блоке.

Значение var1 уровня документа по-прежнему равно 1. А значение var1 для диалога теперь равно 4. Элемент <assign> обращается к var1, объявленной в области диалога, а значение var1, объявленное в анонимном блоке, выпадает из этой области.

```
<vxml>
<var name="var1" expr="1"/>
<form id="form1">
  <var name="var1" expr="3"/>
  <block>
    <var name="var1" expr="5"/>
    <assign name="var1" expr="var1+1"/>
    <!-- A -->
  </block>
  <block>
    <assign name="var1" expr="var1+1"/>
    <!-- B -->
  </block>
```

```

</form>
<form id="form2">
  <block>
    <assign name="var1" expr="var1+1"/>
    <!-- C -->
  </block>
</form>
</vxml>

```

Обращение к неактивным переменным

Если две переменные имеют перекрывающиеся области действия и объявления с тем же самым именем, интерпретатор VoiceXML работает с ними в некоторой внутренней области действия. Программист может задать собственный механизм обработки таких переменных, который будет превалировать над стандартным поведением. Следующий пример показывает задание такой области действия переменной.

```

<vxml>
  <var name="var1" expr="1"/>
  <form id="form1">
    <var name="var1" expr="10"/>
    <block>
      <var name="var1" expr="100"/>
      <assign name="document.var1" expr="document.var1+1"/>
      <assign name="dialog.var1" expr="document.var1+10"/>
      <!-- A -->
    </block>
  </form>
</vxml>

```

Извлечение данных из переменных

Для того чтобы указать переменную в атрибутах `cond` или `expr` элемента VoiceXML, нужно просто указать имя переменной, так как эти атрибуты напрямую видны JavaScript-интерпретатору. Для того чтобы использовать переменную в качестве значения другого атрибута, имя переменной должно быть заключено в скобки. Для того чтобы использовать имя переменной во внутреннем тексте элементов `<audio>` или `<log>`, имя переменной также необходимо окружить скобками.

Эти правила иллюстрируются следующими примерами.

```

<vxml>
  <var name="audio_path"/>

```

```

<var name="welcome_wav"/>
<var name="std_pause"/>
<form id="init">
  <block>
    <assign name="audio_path"
      expr="'ui/'"/>
    <assign name="welcome_wav"
      expr="audio_path + 'intro.wav'"/>
    <assign name="std_pause"
      expr="300"/>
  </block>
</form>
<form id="welcome">
  <block>
    <audio expr="welcome_wav"/>
    <pause>{std_pause}</pause>
  </block>
</form>
</vxml>

```

В следующем примере значение, возвращаемое из грамматики, используется для задания пути к звуковому файлу, который подтверждается выбором пользователя. Диалог `get_class` принимает имя класса от пользователя. Возвращаемое значение, соответствующее имени класса, хранится в переменной `selected_class` с областью действия всего документа. Управление передает ее в диалог `confirm_class`, другой интерактивный диалог, который содержит подсказку, использующую эту переменную для ссылки на звуковой файл.

```

<vxml>
  <var name="selected_class"/>
  <form id="get_class">
    <field name="iClassID">
      <!-- Say a class name. -->
      <prompt><audio src="ui/sayclass.wav"/></prompt>
    <grammar>
      <![CDATA[
        [
          [(build ?me)] {<action "1">}
          [(design ?me)] {<action "2">}
          [(produce ?me)] {<action "3">}
          [(tune ?me)] {<action "4">}
        ]
      ]>
    </grammar>
  </form>
</vxml>

```

```

]]>
</grammar>
<filled>
  <assign name="selected_class" expr="iClassID"/>
</filled>
<default>
  <audio>Sorry. I didn't get that.</audio>
  <reprompt/>
</default>
</field>
</form>
<form id="confirm_class">
  <field name="yesno">
    <prompt>
      <!-- I heard you say XXX. Is that correct? -->
      <audio src="ui/heardsay.wav"/>
      <audio expr="'ui/classes/cn' + selected_class +
'.wav'"/>
      <audio src="ui/isincorrect.wav"/>
    </prompt>
    <grammar>YESNO</grammar>
  <filled>
    <result name="yes">
      <goto next="#class_detail"/>
    </result>
    <result name="no">
      <goto next="#get_class"/>
    </result>
  </filled>
</field>
</form>
<form id="class_detail">
  <!-- code to output detail about the class goes here -->
</form>
</vxml>

```

Обращение к аудиоданным посредством переменных

Интерпретатор VoiceXML поддерживает проигрывание предварительно записанных и синтезированных звуков посредством элемента `<audio>`. Сле-

дующий пример позволяет интерпретатору VoiceXML проигрывать звуковой файл `intro.wav` из каталога `ui`.

```
<audio src="ui/intro.wav">
  welcome to tell me university
</audio>
```

Интерпретатор Tellme VoiceXML поддерживает также использование выражений при обращении к звуковым файлам в атрибуте `expr`.

```
<var name="sPathEarcons"
expr="'http://resources.tellme.com/audio/earcons/'"/>
<audio expr="sPathEarcons + 'intellipause.wav'"/>
```

Интеграция JavaScript

Язык JavaScript является языком общего пользования, объектно-ориентированным языком, первоначально созданным для работы на стороне клиента, который позволял разработчикам Web-услуг выполнять превращение HTML содержания в браузерах Netscape Navigator и Microsoft Internet Explorer. Проверка ввода на стороне клиента является стандартной задачей, выполняемой с использованием этого языка из-за своей начальной интеграции в Web-браузеры.

Из-за высокой популярности и используемости JavaScript был также интегрирован в VoiceXML. Использование JavaScript в голосовом приложении позволяет выполнять действия, которые в противном случае не поддерживаются в языке VoiceXML.

Кроме того, JavaScript включает ряд объектов, которые делают задачу программирования гораздо более легкой.

Следующие ресурсы дают возможность получить детальное представление о языке JavaScript.

- ❑ Flanagan, David. JavaScript: The Definitive Guide, Third Edition. O'Reilly & Associates, Inc. Sebastopol, CA. 1998.
- ❑ Standard ECMA-262 Language Specification (<http://www.ecma.ch/ecma1/stand/ecma-262.htm>).
- ❑ The JavaScript Source (<http://javascript.internet.com/>).
- ❑ JavaScript.com (<http://www.javascript.com/>).

Использование выражений JavaScript

Программисты могут использовать JavaScript везде, где разрешено использование атрибута `expr` или атрибута `cond`.

Например, элементы `<var>` и `<assign>` поддерживают `expr` и ожидают выражения JavaScript. Вот почему в предыдущем параграфе нужно было включать строку в кавычки.

Следующие примеры показывают использование переменных, которые уже объявлены.

```
<assign name="ui_path" expr="'ui/'" />
<assign name="intro_path"
expr="ui_path + 'welcome.wav'" />
```

Следующий пример использует внутреннее математическое выражение JavaScript для генерации случайного файла подсказок.

Файлы 0.wav и 9.wav должны существовать (ниже в программе задается условие выбора произвольного файла с именем от 0.wav до 10.wav).

```
<assign name="class_prompt"
      expr="ui_path + Math.floor(Math.random()*10) + '.wav'" />
<audio expr="class_prompt" />
```

Следующий пример использует выражение JavaScript в условиях определения, какой файл нужно проигрывать midnight, noon, am или pm.

```
<block>
  <var name="d" expr="new Date()" />
  <var name="iHour" expr="d.getHours()" />
  <var name="iMin" expr="d.getMinutes()" />
  <if cond="iHours == 0 && iMin == 0">
    <audio src="ui/time/midnight.wav" />
  <elseif cond="iHours == 12 && iMin == 0">
    <audio src="ui/time/noon.wav" />
  <elseif cond="iHour < 12">
    <audio src="ui/time/am.wav" />
  <else/>
    <audio src="ui/time/pm.wav" />
  </if>
</block>
```

Определение блоков JavaScript

Использование JavaScript часто является более простым, чем использование регулярного выражения.

Использование элемента `<script>` для определения блока инструкций

Так как несколько операторов JavaScript (`<`, `>`, `&`, `&&`) имеют специальное значение в XML, нужно заключить скрипт в раздел CDATA. Кроме того,

нужно установить атрибут `type` на имя языка. В настоящее время платформа Tellme поддерживает только JavaScript, так что атрибут `type` должен указывать на `"text/javascript"`.

В любом случае переменные сценария JavaScript не нужно объявлять в VoiceXML. Прежде чем использовать переменные JavaScript нужно их объявить.

Пример.

```
<vxml>
<var name="utc_month"/>
<var name="utc_day"/>
<var name="utc_year"/>
<script type="text/javascript"><![CDATA[
    // explicitly declare d using var
    var d = new Date();
    // store the month
    utc_month = d.getUTCMonth();
    // store the day of the month
    utc_day = d.getUTCDate();
    // store the year including the century
    utc_year = d.getUTCFullYear();
]]>
</script>
```

Кроме того, определяя блоки внутри JavaScript, можно также определять сценарии для повторного использования, помещая эти сценарии во внешние файлы.

Например:

```
// Is the date/time after noon
function IsPM(d)
{
    return (d.getHours() > 12 ? true : false);
}

// Is the date/time noon?
function IsNoon(d)
{
    return (12 == d.getHours() && 0 == d.getMinutes());
}

// Is the date/time midnight?
function IsMidnight(d)
{.
```

```

    return (0 == d.getHours() && 0 == d.getMinutes());
}
// Return the hour between 1 and 12
function GetNormalizedHour(d)
{
    var iHour = d.getHours();
    if (0 == iHour)
    {
        return 12;
    }
    else if (iHour > 12)
    {
        return iHour-12;
    }
    else
    {
        return iHour;
    }
}

```

Если скрипт содержится во внешнем файле, его код не должен быть заключен в CDATA. Например:

```

<vxml>
<script type="text/javascript"
    src="scripts/dtlib.js"/>
<form>
    <block>
        <audio src="ui/time_is.wav"/>
        <var name="dNow" expr="new Date()"/>
        <if cond="IsMidnight(dNow)">
            <audio src="ui/time/midnight.wav">
        <elseif cond="IsNoon(dNow)"/>
            <audio src="ui/time/noon.wav"/>
        <else/>
            <var name="iHour"
                expr="GetNormalizedHour(dNow)"/>
            <var name="iMinutes"
                expr="dNow.getMinutes()"/>
            <audio expr="'ui/time/' + iHour"/>
            <if cond="iMinutes != 0">
                <audio expr="'ui/time/' + iMinutes"/>

```

```
</if>
<if cond="IsPM(dNow)">
  <audio src="ui/time/pm.wav"/>
<else/>
  <audio src="ui/time/am.wav"/>
</if>
</if>
</block>
</form>
</vxml>
```

Доступ к переменным VoiceXML из JavaScript

Переменная, объявленная в VoiceXML, доступна в сценарии JavaScript как родная переменная JavaScript. Следующий пример устанавливает переменную VoiceXML для числового эквивалента текущего месяца. Месяц проигрывается пользователю, ссылаясь на звуковой файл. Разработчик приложения просто создает звуковые файлы с именами, соответствующими номерам, возвращаемым методом `getUTCMonth` объекта `Date` от 0 до 11.

```
<block>
  <var name="iCurrentMonth" />
  <script type="text/javascript">
    <![CDATA[
      var d = new Date();
      iCurrentMonth = d.getUTCMonth();
    ]]>
  </script>
  <!-- The current month is... -->
  <audio src="curmonth.wav"/>
  <audio expr="'ui/months/' + iCurrentMonth + '.wav'"/>
</block>
```

Область действия блока `<script>`

Элемент `<script>` может содержать функции JavaScript и блоки немедленно исполняемых сценариев. Когда интерпретатор VoiceXML встречает элемент `<script>`, он передает содержание этого элемента интерпретатору JavaScript.

Если документ находится в корневом каталоге приложения, функции JavaScript доступны везде в приложении. В нижеприведенном примере `app_root.vxml` представляет корневой документ приложения, `app_doc1.vxml` является другим документом приложения.

Ниже приведен список комментариев.

- ❑ Переменная `g1`, определенная в `app_root.vxml`, может быть доступна в любом элементе `<script>`, выражении или условии `app_root.vxml` и `app_doc1.vxml`. `g1` имеет область действия всего приложения.
- ❑ Функция `GlobalFunc`, определенная в `root.vxml`, также имеет область действия всего приложения. Функция `DocFunc` имеет область действия документа `app_doc1.vxml`.
- ❑ Функция `LocalFunc`, определенная в диалоге `dialog1` приложения `app_doc1.vxml`, имеет область действия диалога.

```
<!-- app_root.vxml -->
<vxml>
  <var name="g1" expr="0"/>
  <script type="text/javascript">
    <![CDATA[function GlobalFunc() { return g1++; } ]]>
  </script>
</vxml>

<!-- app_doc1.vxml -->
<vxml application="root.vxml">
  <var name="d1"/>
  <script type="text/javascript"><![CDATA[
    function DocFunc() { return d1++; }
  ]]>
  </script>
  <form id="dialog1">
    <var name="l1"/>
    <var name="l2"/>
    <script type="text/javascript"><![CDATA[ function LocalFunc() { return
++l1; } ]]>
    </script>
    <block>
      <script type="text/javascript"><![CDATA[
        l1 = GlobalFunc();
        l2 = LocalFunc();
      ]]>
    </script>
    </block>
  </form>
</vxml>
```

Обработка событий

Спецификация VoiceXML описывает два типа событий — предопределенные события и события, заданные приложением.

Предопределенные события — это события, подхватываемые системой и разбиваемые на две категории: стандартные события и события ошибочных ситуаций.

События, заданные приложением, являются специфическими для данного приложения, создаются и воспринимаются только в рамках текущего приложения.

Следующий список показывает стандартные события обработки ошибок, поддерживаемые платформой Tellme.

- ❑ `help` — подхватывается, когда абонент запрашивает помощь.
- ❑ `noinput` — возникает внутри интерактивного состояния вызова, когда пользователь ничего не сказал в течение заданного периода ожидания ввода.
- ❑ `nomatch` — возникает, когда абонент что-то сказал за пределами активной грамматики.

Список предопределенных событий обработки ошибок:

- ❑ `error.semantic` — возникает при обнаружении семантической ошибки при работе интерпретатора. В настоящее время возникает только тогда, когда абонент говорит что-то за пределами грамматики;
- ❑ `error.badfetch` — возникает, когда HTTP-запрос на извлечение VoiceXML-документа внешней грамматики или внешнего файла сценария не может быть выполнен.

Использование стандартных событий обработки ошибок

Чтобы обработать ошибку, нужно использовать элемент `<catch>` и установить атрибут `event` на имя желаемого события. Например, приведенный обработчик события `<nomatch>` проигрывает некоторый звуковой файл и затем слушает ввод от абонента:

```
<catch event="nomatch">
  <audio>
    I'm sorry. I didn't get that.
    Please say the name of a sport.
    For example, say baseball.
  </audio>
```

```
</listen/>
</catch>
```

Для того чтобы обработать общее предопределенное событие — `help`, `noinput` и `nomatch` — спецификация VoiceXML определяет соответствующие теги с теми же самыми именами. Например:

```
<noinput>
  <audio>I'm sorry. I didn't hear you</audio>
  <reprompt/>
</noinput>
```

Использование событий, определенных приложением

Определенные приложением события произвольно задаются разработчиком и обычно используются для согласования работы приложения с требуемым поведением пользователя.

Отслеживание событий

Для того чтобы заставить платформу отслеживать события, заданные приложением, используется элемент `<throw>`. Этот элемент поддерживает атрибут `event`.

Любому такому событию можно присвоить имя, которое не конфликтует со стандартными именами предопределенных событий. Таких конфликтов можно избежать путем задания имен в следующей нотации:

```
event.<company-name>.<app-name>.<event-name>
```

Следующий пример показывает использование поддиалогов. Поддиалог пытается проверить ID пользователя и пароль. Если проверка успешна, элемент `<subdialog>` возвращает событие `event.mycomp.myapp.valid`, а соответствующий обработчик в вызывающем диалоге инструктирует интерпретатор VoiceXML о необходимости перехода к документу `getsched.vxml`. Если проверка неуспешна, поддиалог возвращает событие `event.mycomp.myapp.invalid`, а соответствующий обработчик в вызывающем диалоге проверяет счетчик попыток и запускает событие, определенное приложением `event.mycomp.myapp.onmaxpasswordretries`, если этот счетчик превышен.

```
<form>
  <subdialog src="validate_credentials.pl"
    namelist="id password">
    <catch event="event.mycomp.myapp.valid">
      <goto next="getsched.vxml"/>
    </catch>
    <catch event="event.mycomp.myapp.invalid">
```

```
<assign name="iRetries" expr="iRetries+1"/>
<if cond="iRetries > iMaxPwdRetries">
  <throw event="event.mycomp.myapp.onmaxpasswordretries"/>
<else/>
  <goto next="#get_id"/>
</if>
</catch>
</subdialog>
</form>
```

Обработка событий

Для обработки событий, определенных приложением, необходимо использовать элемент `<catch>`. Нижеприведенный пример ловит событие `event.mycomp.myapp.onfatalerror`. Это событие не отслеживается системой. Поэтому приложение может отследить его самостоятельно с помощью элемента `<throw>` из различных точек, где это может случиться.

```
<catch event="event.mycomp.myapp.onfatalerror">
  <audio>I'm sorry, but we're experiencing
    technical difficulties. Now connecting
    you to a Tellme University representative
  </audio>
  <transfer dest="{giCallCenter}">
  <default/>
</transfer>
<disconnect/>
</catch>
```

Если приложение не обеспечивает детальную обработку таких событий с помощью собственных обработчиков или обработчиков по умолчанию, интерпретатор VoiceXML будет завершать работу такого приложения.

Область действия события

Язык VoiceXML обеспечивает разработчику гибкость в отслеживании событий внутри любого из следующих элементов: `<vxml>`, `<form>`, `<field>`, `<menu>`, `<block>`, `<record>`, `<subdialog>`, `<transfer>` и `<confirm>`.

При появлении события интерпретатор VoiceXML ищет соответствующий обработчик.

Если такой обработчик не может быть найден, интерпретатор VoiceXML выполняет обратный поиск ближайшего доступного обработчика.

Например, при возникновении события `nomatch` интерпретатор сначала пытается выполнить обработчик `<nomatch>`, размещенный внутри поля, в кото-

ром возникло событие. Если он не задан, то интерпретатор VoiceXML ищет обработчик в области диалога. Если такой не существует, интерпретатор ищет его в области действия документа. Если это корневой документ, интерпретатор ищет его там. Если интерпретатор не может его там найти, он завершает приложение.

Событие не может быть активировано из области действия анонимного блока. Элементы, которые определяют область анонимного блока, включают `<block>`, `<filled>` или обработчик событий.

Пример демонстрирует использование обработчика событий.

```
<!-- app_root.vxml -->
<vxml>
  <catch event="event.mycomp.myapp.onnomatch">
    <assign name="iNoMatches" expr="iNoMatches+1"/>
    <reprompt/>
  </catch>
</vxml>

<!-- doc.vxml -->
<vxml>
  <form>
    <var name="iNoMatches" expr="0"/>
    <field name="sport">
      <prompt><audio>Pick a sport</audio></prompt>
      <grammar src="sports.gsl"/>
      <nomatch>
        <script src="errorlib.js"
          type="text/javascript"/>
        <var name="var1"/>
        <throw event="event.mycomp.myapp.onnomatch"/>
      </nomatch>
    </field>
  </form>
</vxml>
```

Стандартная обработка события

Элемент `<default>` позволяет обрабатывать события `nomatch`, `noinput` и `filled` единообразно. На практике эти события будут обрабатываться разными обработчиками, однако на фазе создания прототипа удобно пользоваться единым интерфейсом обработки событий.

Главный недостаток обработчика `<default>` заключается в том, что разработчик не способен точно определить, какое событие произошло.

Как все обработчики событий, за исключением `<filled>`, обработчик `<default>` может выполняться в приложении, документе, диалоге или поле. Следующий пример показывает такое выполнение на уровне приложения.

```
<!-- sport_root.vxml -->
<vxml>
  <default>
    <log>default handler</log>
    <audio>Sorry. I didn't get that.</audio>
    <reprompt/>
  </default>
</vxml>

<!-- sport.vxml -->
<vxml application="sport_root.vxml">
  <form id="get_sport">
    <field name="sport" timeout="1.0">
      <prompt><audio>Pick a sport.</audio></prompt>
      <grammar><![CDATA[
        [
          [baseball] {<option "baseball">}
          [football] {<option "football">}
          [hockey] {<option "hockey">}
          [soccer] {<option "soccer">}
          [help] {<option "help">}
        ]
      ]]>
    </grammar>
    <help>
      <audio>Say the name of a sport. For example, say baseball.</audio>
    </listen/>
    </help>
    <filled>
      <goto expr="gsSport + '.vxml'"/>
    </filled>
  </field>
</form>
</vxml>
```

Выполнение нескольких обработчиков одного и того же события

Эффективное голосовое приложение всегда обрабатывает трудности, с которыми встречается абонент в конкретной части диалога.

Трудности распознаются приложением, когда интерпретатор VoiceXML обнаруживает множественные события `nomatch`, `noinput` и `help` в рамках одного диалога. Вместо повторения той же самой информации, приложение может помочь абоненту несколькими вариантами разных подсказок, которые проигрываются соответствующими обработчиками.

VoiceXML следит за количеством обрабатываемых событий каждого типа внутри одного диалога и позволяет выполнять разные обработчики событий с помощью атрибута `count`.

```
<form id="command">
  <prompt count="1">
    <audio>Okay, now pick a fruit.</audio>
  </prompt>
  <prompt count="2">
    <audio>Please say the name of a fruit.
      For example, say banana.
    </audio>
  </prompt>
  <grammar src="fruits.gsl"/>
  <nomatch count="1">
    <audio>I'm sorry. I didn't get that.</audio>
    <reprompt/>
  </nomatch>
  <nomatch count="2">
    <audio>I'm sorry. I still didn't get that.</audio>
    <reprompt/>
  </nomatch>
  <nomatch>
    <audio>I'm sorry. I'm having trouble
understanding. Type the first few letters of
the sport. Press pound when you're done.
    </audio>
    <listen/>
  </nomatch>
  <noinput count="1">
    <audio>I'm sorry. I didn't hear you.</audio>
    <reprompt/>
  </noinput>
  <noinput count="2">
    <audio>I'm sorry. I still didn't hear you.</audio>
```

```
<reprompt/>
</noinput>
<noinput><goto next="_home"/></noinput>
</form>
```

Если выполняется всего один обработчик, атрибут `count` описывать необязательно.

Построение VoiceXML-приложений

Голосовое приложение является набором одного или нескольких документов VoiceXML, каждый из которых состоит из одного или более диалогов.

С помощью элемента `<audio>` диалог обычно представляет информацию абоненту в форме записанного или синтезированного звукового файла.

С помощью элементов `<field>`, `<prompt>`, `<grammar>` и `<filled>` диалог может также получать информацию от абонента.

VoiceXML обеспечивает одну точку входа в приложение. Точкой входа в приложение является первый документ приложения, который интерпретатор VoiceXML загружает при старте приложения. Все другие документы ссылаются прямо или косвенно на точку входа.

Документы голосового приложения обычно одновременно используют корневой документ, который может содержать объявления переменных, сценарии, ссылки, грамматики и обработчики событий, доступные в других частях приложения.

Рис. 6.2 описывает простое голосовое приложение (состоящее из главного документа "doc1") как точку входа корневой документа "app_root", который задает разделяемые приложением ресурсы и два дополнительных документа VoiceXML, "doc2" и "doc3".

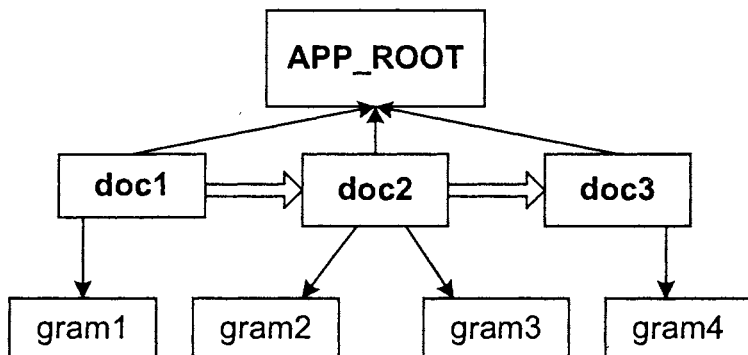


Рис. 6.2. Схема голосового приложения

- ❑ "doc2" ссылается на внешние грамматики, размещенные в файлах "gram2" и "gram3".
- ❑ "doc3" ссылается на внешнюю грамматику, размещенную в файле "gram4".
- ❑ Приложение работает проходя по документам "doc1" и "doc2", а также от "doc2" к "doc3".

Рис. 6.3 показывает более сложное приложение.

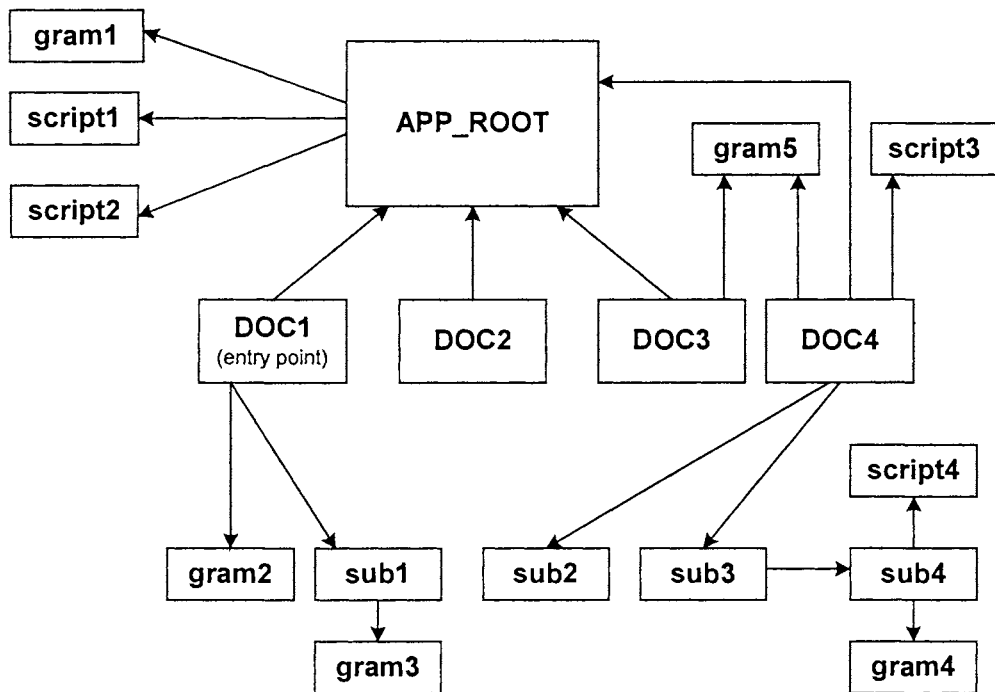


Рис. 6.3. Схема сложного голосового приложения

- ❑ "app root" представляет корневой документ приложения. Он ссылается на внешнюю грамматику "gram1" и внешние сценарии "script1" и "script2".
- ❑ "doc1" представляет точку входа в приложение. В этом приложении "doc1" ссылается на внешнюю грамматику "gram2". В процессе выполнения "doc1" вызывает поддиалог, содержащийся в документе VoiceXML "sub1". Поддиалог "sub1" ссылается на внешнюю грамматику "gram3".
- ❑ "doc2", "doc3" и "doc4" представляют другие документы VoiceXML приложения. Все три документа ссылаются на корневой документ "app root" и имеют доступ к ресурсам, заданным в нем. Навигация между этими документами не показана на рисунке.
- ❑ "doc3" ссылается на внешнюю грамматику "gram5". В процессе выполнения он вызывает поддиалог, размещенный в документе "sub2".

- ❑ "doc4" ссылается на внешнюю грамматику "gram5" и на внешний сценарий script3. При выполнении "doc4" вызывает поддиалоги из "sub2" и "sub3". Поддиалог из "sub3" вызывает поддиалог в "sub4". Этот поддиалог "sub4" ссылается на внешнюю грамматику "gram4" и на внешний сценарий "script4".

Приложение протекает по документам от "doc1" до "doc2", туда и обратно между "doc2" и "doc3" и окончательно от "doc3" к "doc4".

Задание корневого документа приложения

При построении голосового приложения полезно определить его разделяемые ресурсы. Переменные, объявленные в корневом приложении, имеют область действия всего приложения, позволяя передавать состояние между разными документами приложения. То же самое относится к другим ресурсам приложения.

Ниже приведен пример корневого документа приложения, который содержит следующие ресурсы.

- ❑ Одна переменная указывает, что абонент уже авторизован в приложении, другая хранит student-ID; третья хранит URL следующего диалога в приложении.
- ❑ Элемент `<link>` задает грамматику, указывая, что когда пользователь говорит "start over" приложение должно перескочить к диалогу welcome в документе tellmeu.vxml.
- ❑ Элемент `<script>` задает две функции. Первая возвращает значение, указывающее авторизован абонент или нет. Вторая иницирует предварительно описанные переменные.
- ❑ Обработчик событий ловит события, определенные пользователем, "event.onlistmyclasses". При возникновении события обработчик назначает URL переменной уровня приложения gsNextDialog и переходит к документу login.vxml.
- ❑ Элемент `<property>` устанавливает значение тайм-аута на одну секунду для всех полей приложения, за исключением тех, которые имеют специальное значение тайм-аута.

```
<!-- tellmeu_root.vxml -->
<vxml>
  <var name="gbLoggedIn" expr="false"/>
  <var name="giStudentID" />
  <var name="gsNextDialog" />
  <link next="tellmeu.vxml#welcome">
  <grammar>[(start over)]</grammar>
</link>
```

```

<script type="text/javascript"><![CDATA[
  function IsLoggedIn() { return application.gbLoggedIn; }
  function InitGlobals()
  {
    application.gbLoggedIn = false;
    application.giStudentID = null;
    application.gsNextDialog = null;
  }
  ]>]]>
</script>
<catch event="event.onlistmyclasses">
  <assign name="application.gsNextDialog"
    expr="'my_classes.asp'"/>
  <goto next="login.vxml"/>
</catch>
<property name="timeout" value="1.0"/>
</vxml>

```

Ссылка на корневой документ приложения

Ссылка на корневой документ выполняется заданием атрибута приложения VoiceXML.

```

<!-- tellmeu.vxml -->
<vxml application="tellmeu_root.vxml">
  <!-- document scoped variables, scripts, and event handlers and dialogs
  go here -->
</vxml>

```

Когда вы ссылаетесь на корневой документ, интерпретатор VoiceXML создаст область действия приложения, за исключением поддиалогов. Если голосовое приложение переходит к документу, который ссылается на другое приложение, область действия приложения разрушается. Такое поведение можно продемонстрировать на следующем примере.

Точка входа в приложение — doc1.vxml. Когда выполняется диалог form1, он увеличивает значение переменной gVar на 10 и переходит к диалогу form3 в doc2.vxml, который переходит к диалогу form3 в doc1.vxml.

```

<!-- app_root.vxml -->
<vxml>
  <meta name="scoping" content="new"/>
  <var name="gVar" expr="1"/>
</vxml>
<!-- doc1.vxml -->

```

```
<vxml application="app_root.vxml">
  <meta name="scoping" content="new"/>
  <form id="form1">
    <block>
      <assign name="gVar" expr="gVar+10"/>
      <goto next="doc2.vxml#form3"/>
    </block>
  </form>
  <form id="form2">
    <block>
      <log>In form2, gVar is <value expr="gVar"/></log>
    </block>
  </form>
</vxml>
<!-- doc2.vxml -->
<vxml>
  <meta name="scoping" content="new"/>
  <form id="form3">
    <block>
      <goto next="doc2.vxml#form3"/>
    </block>
  </form>
</vxml>
```

Использование вложенных диалогов

Поддиалог является самосодержательным компонентом VoiceXML многократного использования. Для своего создания он не требует обращения к исполняемому содержанию от вызывающего абонента. Исполняемое содержание поддиалога включает переменные, определенные в вызывающем диалоге, и ресурсы уровня приложения.

Данные в поддиалогах могут быть разделены для взаимного использования двумя путями — посредством передачи параметров и использованием серверной технологии, такой как Perl или Active Server Pages (ASP). Поддиалог может передать данные обратно звонящему, используя атрибут `namelist`.

Замечание

Тег `<param>` и атрибут `namelist` поддерживаются только, когда атрибут `content` тега `<meta>` установлен в `new`.

Передача параметров к вложенному диалогу

Использует тег `<param>` для передачи параметра поддиалогу. Тег `<param>` поддерживает атрибуты `name` и `expr`. Атрибут `name` именуется параметр. Атрибут `expr` определяет значение для параметра и принимает любое правильное ECMAScript-выражение.

Следующий пример передает параметры "iAnswer" и "bPrompt" к поддиалогу, содержащемуся в документе "picknumber.vxml". Поддиалог объявляет переменные с теми же самыми именами. Когда поддиалог вызывается, интерпретатор VoiceXML инициализирует переменные со значениями, соответствующими параметрам.

```
<vxml>
  <var name="iAnswer"
        expr="Math.floor(Math.random(1) *100)"/>
  <var name="bPrompt" expr="true"/>
  <form id="call_pick_number">
    <subdialog name="oResult" src="picknumber.vxml">
      <param name="iAnswer"
            expr="document.iAnswer"/>
      <param name="bPrompt"
            expr="document.bPrompt"/>
    <filled>
      <if cond="oResult.iRetVal">
        <audio>You should head to Vegas</audio>
        <goto next="_home"/>
      <else/>
        <if cond="oResult.iPick > oResult.iAnswer">
          <audio>Try a number lower than
            <value expr="oResult.iPick"/>.
          </audio>
        <else/>
          <audio>Try a number higher than
            <value expr="oResult.iPick"/>.
          </audio>
        </if>
        <assign name="bPrompt" expr="false"/>
        <goto next="#call_pick_number"/>
      </if>
    </filled>
  </subdialog>
```

```
</form>
</vxml>
<!-- picknumber.vxml -->
<vxml>
  <form>
    <var name="iAnswer"/>
    <var name="bPrompt"/>
    <var name="iRetval" expr="0" />
    <field name="iPick">
      <grammar>NATURAL_NUMBER_THRU_99</grammar>
      <prompt count="1">
        <if cond="bPrompt">
          <audio>Pick a number</audio>
        </if>
      </prompt>
      <prompt count="2">
        <audio>Please pick a number</audio>
      </prompt>
      <default>
        <audio>Sorry.</audio>
        <reprompt/>
      </default>
      <filled>
        <if cond="iPick == iAnswer">
          <assign name="iRetval" expr="1"/>
        </if>
        <return namelist="iAnswer iPick iRetval"/>
      </filled>
    </field>
  </form>
</vxml>
```

Использование сценариев на стороне сервера для передачи данных к поддиалогу

Существует также возможность передачи данных к поддиалогу, выполняя поддиалог с использованием технологии на стороне сервера (Perl или Active Server Pages (ASP)) и передавая данные как часть строки запроса или атрибутом `namelist` тега `<subdialog>`. Этот подход особенно полезен, если нужно выполнить некоторый скрытый процесс подтверждения корректности данных пользователя.

Возврат данных звонящему

Для возврата данных звонящему нужно использовать элемент `<return>`. Вы можете вернуть информацию абоненту, используя атрибут `event` или атрибут `namelist`. `namelist` является набором переменных, разделенных пробелами и определенных внутри поддиалога. Каждая переменная в списке становится свойством объекта, на которую можно сослаться по имени.

В следующем примере значение `oCC` именуется переменной, на которую ссылаются с помощью свойств `sNumber` и `dExpDate`. Подходящее выполнение поддиалога взаимодействовало бы с абонентом с целью получения номера кредитной карты.

В этом примере поддиалог иницирует `sNumber` с 16-символьной строкой и `dExpDate` с объектом ECMAScript, который хранит текущую дату.

```
<subdialog name="oCC" src="#get_ccinfo">
  <filled>
    <debug>card number is
      <value expr="oCC.sNumber"/>
    </debug>
    <debug>expiration is
      <value expr="(oCC.dExpDate.getMonth()+1)
+ ' ' +oCC.dExpDate.getFullYear()"/>
    </debug>
  </filled>
</subdialog>
<form id="get_ccinfo">
  <var name="sNumber" expr="'5424000000000000'"/>
  <var name="dExpDate" expr="new Date()"/>
  <return namelist="iNumber dExpDate"/>
</form>
```

Для того чтобы поймать событие, возникшее в поддиалоге, нужно добавить один или более элементов `<catch>` в элемент `<subdialog>` и установить атрибут `event` для каждого обработчика на имя события, возвращаемого в поддиалог.

Если элемент `<catch>`, соответствующий событию, в поддиалоге не описан, интерпретатор VoiceXML будет останавливать приложение при возникновении этого события. Для того чтобы предотвратить эту ситуацию, лучше задать обработчик событий по умолчанию в качестве порожденного элемента `<subdialog>` или диалога, или документа. Если поддиалог не возвращает события, нужно использовать элемент `<filled>` в качестве порожденного элемента `<subdialog>` для обработки любых значений возвращаемых поддиалогом.

Исполнение поддиалога на стороне сервера

В следующем примере `iStudentID` и `iStudentPIN` определены в области действия документа. Диалог `"get_id"` собирает `student ID` и хранит значение в переменной `iStudentID`. Диалог `"get_pin"` собирает `student PIN` и хранит значение в переменной `iStudentPIN`. Передача контролируется диалогом `"validate_proxy"` и использует атрибут `namelist` элемента `<subdialog>` для передачи `student ID` и `student PIN` посредством Perl-скрипта.

Сценарий PERL использует CGI-модуль, чтобы протолкнуть `student ID` и `PIN` через `http`-запрос. Затем он вызывает гипотетическую функцию `ValidateStudent`, которая выполняет просмотр базы данных для определения корректности введенных данных.

Если сценарий Perl возвращает `"event.invalid"`, интерпретатор VoiceXML инструктируется запустить событие, определенное пользователем `"event.retry_login"`. Обработчик этого события делает следующее:

- ☐ информирует абонента об ошибке корректности данных;
- ☐ увеличивает счетчик попыток, проверяет, что граничное значение счетчика не достигнуто;
- ☐ возвращает управление в диалог, который собирает `student ID`, если счетчик не превышен, или стартует другое определенное пользователем событие `"event.login_retries_exceeded"` уровня приложения. Обработчик этого события не показан, но по функциональности он передает абоненту ответ, используя элемент передачи.

```
<!-- login.vxml -->
<vxml application="tellmeu_root.vxml">
  <var name="iStudentID"/>
  <var name="iStudentPIN"/>
  <var name="iRetries" expr="1"/>
  <catch event="event.retry_login">
    <audio>Invalid i d or password</audio>
    <assign name="iRetries" expr="iRetries+1"/>
    <if cond="iRetries == giMaxLoginRetries">
      <throw event="event.login_retries_exceeded"/>
    </if>
    <goto next="#get_id"/>
  </catch>
  <form id="get_id">
    <!-- retrieve the student's id -->
    <!-- then navigate to #get_pin -->
  </form>
  <form id="get_pin">
```

```

<!-- retrieve the student's pin -->
<!-- then navigate to #validate_proxy -->
</form>
<form id="validate_proxy">
  <subdialog src="validate_id.pl"
    namelist="iStudentID iStudentPIN">
    <catch event="event.valid">
      <!-- the id and PIN were valid. Proceed. -->
      <assign name="giStudentID" expr="iStudentID"/>
      <assign name="giStudentPIN" expr="iStudentPIN"/>
      <goto expr="gsNextDialog"/>
    </catch>
    <catch event="event.invalid">
      <!-- the user id or password was invalid.
      Inform the user; try again.
      -->
      <throw event="event.retry_login"/>
    </catch>
    <catch event="event.error">
      <!-- an error occurred -->
      <throw event="event.abort"/>
    </catch>
  </subdialog>
</form>
# -----
# validate_id.pl
# -----
use CGI;
$o = new CGI;
$id = $o->param("iStudentID");
$pin = $o->param("iStudentPIN");
# Determine whether or not the student id/pin is valid.
$bValid = ValidateStudent($id, $pin);
$sEvent = ($bValid ? "event.valid" : "event.invalid");
print "Content-Type: text/xml\n\n";
print <<"EOF";
<vxml>
<form>
  <block>
    <return event="$sEvent"/>

```

```
</block>
</form>
</vxml>
EOF
```

Данные приложения и JavaScript

Платформа Tellme имеет встроенные средства интеграции с программными модулями, написанными с использованием языка JavaScript. Эти средства основаны на реализации функции `vxmlllog`, которая позволяет описать вывод в log-файл внутри элемента `<script>`.

Замечание

При использовании подходящей области действия не требуется использования объекта `vxmldata`.

Функция `vxmlllog`

Печатает вывод в Debug Log. Можно задавать строки, целые числа, числа с плавающей точкой (например, 2.495), выражения с числами с плавающей точкой (например, $2/5 \times 34.5$) и несколько аргументов ("a", "b", "c"). Все аргументы преобразуются в строки, но другие объекты нет. Эта функция эквивалентна элементу `<debug>`.

Использование: `vxmlllog("JS_expression")`.

Пример

```
vxmlllog("*** Debug", 2+3, " is the result.")
```

производит печать следующих данных

```
*** Debug 5 is the result.
```

Объект `vxmldata`

При использовании правильной области действия вам нет необходимости использовать объект `vxmldata`. Обратите внимание, что использование объекта `vxmldata` в JavaScript, ссылочная переменная, которая не существует и оценивается как ноль.

Метод `vxmldata.get`

Извлекает переменную и возвращает ее значение как строку. Если переменная является списком строк, возвращается только первый элемент.

Использование: `vxmldata.get("variable")`.

Пример

```
vxmldata.get("document.myapp.color")
```

возвращает значение переменной `document.myapp.color`. Обратите внимание, что вы также можете использовать синтаксис:

```
vxmldata["variable"]
```

Метод `vxmldataset`

Устанавливает переменную и возвращает `boolean`. Метод эквивалентен элементу `<assign>`.

Использование: `vxmldata.set("variable", "string").`

Метод `vxmldatagetList`

Извлекает все элементы переменной и возвращает массив JavaScript. Если переменная не существует или происходит ошибка исполнения `vxmldata.getList()`, возвращается `"null"`.

Использование: `vxmldata.getList("variable").`

Пример

```
<vxml application="http://resources.tellme.com/lib/universals.vxml">
  <form>
    <script type="javascript">
      <![CDATA[
        vxmldata.set("document.myapp.phone", "'650-815-1234'");
        vxmldata.append("document.myapp.phone", "'650-917-5555'");
        vxmlllog("document.myapp.phone=",
        vxmldata["document.myapp.phone"]);
        phones = vxmldata.getList("document.myapp.phone");
        if(phones)
        {
          vxmlllog("phones length= ", phones.length);
          for(i =0;i <phones.length;i++)
            { vxmlllog("item[" ,i,"]=" , phones[i]); }
        }
      ]]>
    </script>
  </form>
</vxml>
```

порождает следующий вывод в Debug Log

```
document.myapp.phone=650-815-1234
```

```
phones length= 2
```

```
item[0]=650-815-1234
```

```
item[1]=650-917-5555
```

Метод *vxmldataclear*

Очищает все элементы переменной. Эквивалентно элементу `<clear>`.

Использование: `vxmldata.clear("variable").`

Пример

```
vxmldata.clear("document.myapp.phone")
```

Метод *vxmldataappend*

Добавляет значение к существующей переменной.

Использование: `vxmldata.append("variable", "string").`

Пример 1

```
vxmldata.append("document.myapp.color", "green");
```

```
vxmldata.append("document.myapp.color", "yellow");
```

```
vxmldata.append("document.myapp.color", "red");
```

добавляет значения "green", "yellow" и "red" к существующей информации переменной. Если значение переменной `document.myapp.color` было предварительно "blue", новое значение будет списком строк: "blue green yellow red".

Пример 2

```
vxmldata.set("document.myapp.fruits", "apple");
```

```
vxmldata.append("document.myapp.fruits", "orange");
```

```
vxmldata.append("document.myapp.fruits", "banana");
```

```
vxmldata.set("document.myapp.fruits_2",
```

```
vxmldata["document.myapp.fruits"]);
```

создает новую переменную документа `document.myapp.fruits_2` и назначает ей первую строку в `document.myapp.fruits`.

Метод *vxmldataremove*

Удаляет значение из существующей переменной. Нужно точно указать удаляемое значение.

Использование: `vxmldata.remove("variable", "string").`

Пример

```
vxmldata.remove("document.myapp.color", "yellow")
```

Метод *vxmldatanext*

Передаёт управление заданному URL. Этот метод аналогичен элементу `<goto>`, за исключением `vxmldata.next` и не приводит к переключению содержания до тех пор, пока не завершится исполнение текущего блока JavaScript.

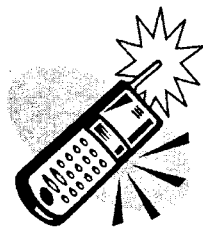
Использование: `vxmldata.next = "URL".`

Пример

```
<vxml application="http://resources.tellme.com/lib/universals.vxml">
  <form>
    <block>
      <script type="javascript">
        var theURL = "http://myserver/next_document.vxml#init";
        vxmldata.next = theURL;
      </script>
      <audio>This audio never plays</audio>
    </block>
  </form>
</vxml>
```

Глава 7

Описание тегов VoiceXML



Эта глава описывает все элементы языка, исполняемые на платформе сети компании Tellme. Информация об основных отличиях от стандарта представлена в *приложении 1*.

Теги

Для удобства ознакомления и использования в справочных целях в этом разделе все основные теги языка сгруппированы в соответствии с их главным способом использования или выполнения в составе приложения VoiceXML.

Теги документа

- ☐ `<vxml>` — содержит последовательности диалогов взаимодействия с пользователем.
- ☐ `<meta>` — задает общую информацию о VoiceXML-документе.
- ☐ `<link>` — описывает одну или более грамматик и путь назначения или событие.
- ☐ `<grammar>` — задает разрешенный словарь для данного взаимодействия с абонентом.
- ☐ `<var>` — объявляет локальную переменную.
- ☐ `<form>` — содержит последовательность элементов поля, которые должны быть заполнены в процессе взаимодействия с абонентом.
- ☐ `<menu>` — представляет список выборов абонента и транзакции, которые должны быть исполнены для выбранной информации.

Теги формы

- ☐ `<grammar>` — задает разрешенный словарь для текущего взаимодействия с абонентом.
- ☐ `<block>` — описывает блок директив, которые выполняются в заданном документом порядке.

- ☐ `<dtmf>` — определяет разрешенные DTMF-клавиши для текущего взаимодействия с абонентом.
- ☐ `<filled>` — задает действие, которое должно быть выполнено, когда переменной `<field>` присваивают некоторое значение.
- ☐ `<field>` — описывает ввод, полученный от абонента.
- ☐ `<record>` — записывает результат ввода абонента.
- ☐ `<subdialog>` — переход на URL для последующего выполнения приложения.
- ☐ `<transfer>` — перенаправляет звонящего на другой телефонный номер.

Теги поля

- ☐ `<grammar>` — определяет разрешенный словарь для данного взаимодействия абонента.
- ☐ `<confirm>` — предлагает абоненту подтвердить ответ.
- ☐ `<dtmf>` — определяет разрешенные DTMF-последовательности для данного взаимодействия абонента.
- ☐ `<filled>` — описывает действие, которое нужно выполнить, когда переменной `<field>` назначается некоторое значение.
- ☐ `<result>` — описывает действие, которое нужно выполнить для распознавания, соответствующего содержанию `<filled>`.

Теги меню

- ☐ `<grammar>` — определяет разрешенный словарь для текущего диалога абонента.
- ☐ `<choice>` — определяет одну строку в меню.

Обработчики событий:

- ☐ `<prompt>` — проиграть звуковой файл абоненту и слушать ответ.
- ☐ `<nomatch>` — обработать ввод абонента, который не распознается как часть активной грамматики.
- ☐ `<noinput>` — обрабатывает случаи, где абонент не говорит.
- ☐ `<help>` — обрабатывает случай, когда абонент говорит "help".
- ☐ `<default>` — подхватывает все события, не обработанные ни одним из описанных обработчиков событий.
- ☐ `<catch>` — ловит события, заданные в VoiceXML-приложении или Tellme Platform.
- ☐ `<filled>` — описывает действие, вызываемое в момент назначения переменной `<field>` некоторого значения.

Исполняемые теги

- ❑ `<assign>` — назначает значение переменной.
- ❑ `<audio>` — проигрывает звуковой файл или конвертирует текст в речь в составе подсказки.
- ❑ `<clear>` — очищает значение переменной.
- ❑ `<disconnect>` — завершает телефонный вызов абонента.
- ❑ `<foreach>` — выполняет повторение по списку элементов.
- ❑ `<goto>` — выполняет переход к заданному URL.
- ❑ `<if><elseif/><else/>` — выполняет процедуру условного исполнения.
- ❑ `<listen>` — заставляет платформу слушать ввод от абонента.
- ❑ `<log>` — пишет вывод в log-файл Tellme Studio Debug Log.
- ❑ `<pause>` — добавляет заданный период времени неактивности и затем продолжает исполнение приложения.
- ❑ `<reprompt>` — заново пригласывает предварительно проигранную подсказку.
- ❑ `<return>` — возвращает управление последнему документу, добавленному в стек `<subdialog>`.
- ❑ `<script>` — включает в документ блок кода на языке JavaScript.
- ❑ `<submit>` — получает новый документ с использованием запросов HTTP GET или POST.
- ❑ `<throw>` — связывает системное или определенное пользователем событие с процедурой его обработки.
- ❑ `<var>` — объявляет локальную переменную.

Тег `<assign>`

Присваивает значение некоторой переменной.

Синтаксис

```
<assign name="строка" expr="JS_выражение"/>
```

где `name` — имя переменной (обязательное поле), `expr` — новое значение переменной. Если переменная имеет тип строка, нужно заключить ее в одинарные кавычки:

```
expr="'значение строки'"
```

Элемент `<assign>` присваивает значение переменной. Переменная должна быть объявлена с использованием элемента `<var>`.

Вызывающие теги: `<block>`, `<filled>`, `<if>`, `<prompt>`, `<catch>`, `<foreach>`, `<noinput>`, `<result>`, `<default>`, `<help>`, `<nomatch>`.

Порожденные теги: нет.

Пример

```
<vxml>
<!--variable with document scope-->
<var name="mycost"/>
  <form id="test">
    <block>
<!--variables with dialog scope-->
      <var name="flavor"/>
      <var name="mycost" expr="4"/>
      <assign name="flavor" expr="'chocolate'"/>
      <assign name="document.mycost" expr="mycost + 14"/>
      <goto next="_home"/>
    </block>
  </form>
</vxml>
```

Тег **<audio>**

Проигрывает звуковой файл или преобразует текст в речь в составе подсказки.

Синтаксис

```
<audio src="URL"
      data="variable"
      expr="JS_expression">
  Text-to-Speech text
</audio>
```

где

- ❑ **src** — URL звукового файла (необязательный атрибут).

Замечание

Если не задавать URL несуществующего файла и не обеспечивать альтернативного текста, механизм генерации речи TTS (Text-to-Speech) выдаст сообщение об ошибке.

- ❑ **data** — переменная, которая хранит звуковые данные, записанные с использованием элемента `<record>` (необязательный атрибут, нельзя использовать в сочетании с **src**):


```
<audio data="recordvar"/>
```
- ❑ **expr** — выражение языка JavaScript, которое необходимо выполнить (оценить, получить значение). Результат такой оценки используется в месте, указанном в атрибуте **src** (необязательный параметр).

Замечание

Атрибут `expr` эквивалентен элементу `src` языка JavaScript. Он не влияет на очередность исполнения звука TTS, так что его можно проиграть позднее. Например, нижеприведенный код ничего не проигрывает:

```
<assign name="audio" expr="'This is a test'"/>
<audio expr="audio"/>
```

Элемент `<audio>` проигрывает звук абоненту — предварительно записанный звуковой файл или синтезированную речь. Если атрибуты `src` или `data` указывают на корректный звуковой файл, то любой текст, заданный в тексте тега, игнорируется. Если звуковой файл не найден, то указанный текст проигрывается абоненту.

Платформа проталкивает звуковые сообщения в стек подсказок до тех пор, пока не встретит директиву `<prompt>` или `<listen>`. В этот момент звуковое сопровождение проигрывается последовательно в том порядке, в котором оно было отправлено в стек.

Текущая реализация платформы понимает, как извлекать и проигрывать стандартные wav-файлы следующих форматов записи:

- ☐ моно или стерео;
- ☐ 8 кГц, 11,025 кГц, 16 кГц, 22,05 кГц, или 44,1 кГц;
- ☐ 8 или 16 bit linear Pulse Coded Modulation (PCM).

В дальнейшем платформа Tellme будет поддерживать новые форматы записи звуковых файлов, включая потоковые звуковые форматы. Независимо от формата записи звукового файла, система будет преобразовывать его в моно 8 кГц. Качество звука не улучшается от использования новых типов форматов.

Замечание

Текущая реализация платформы преобразует текст внутри элементов `<block>` и `<prompt>` в речь, не требуя, чтобы он находился внутри тегов `<audio>`. Тем не менее, очень рекомендуется, чтобы теги `<audio>` в дальнейшем использовались более согласованно.

Вызывающие теги: `<block>`, `<filled>`, `<if><else/><elseif/>`, `<prompt>`, `<catch>`, `<foreach>`, `<noinput>`, `<result>`, `<default>`, `<help>`, `<nomatch>`.

Порожденные теги: `<value>`.

Пример

В нижеприведенном примере приложение создает Text-to-Speech (TTS) версию строки "Welcome to the Bird Seed Emporium", и полученный файл направляется в стек.

```
<vxml>
  <form>
```

```

<block>
  <audio>Welcome to the Bird Seed Emporium</audio>
  <audio
    src="http://resources.tellme.com/audio/earcons/tellmelogo.wav">
    Tellme Logo Audio File</audio>
    <goto next = "_home"/>
  </block>
</form>
</vxml>

```

Эти примеры извлекают, кэшируют и выталкивают в стек звуковой файл, описанный в атрибуте `src`. Если файл не найден, версия TTS строки "Welcome to the Bird Seed Emporium" создается, а полученный файл выталкивается в стек. При этом не возникает никаких условий исполнения обработки ошибочных состояний.

```

<vxml>
  <form>
    <block>
      <audio>Welcome to the Bird Seed Emporium</audio>
      <goto next = "_home"/>
    </block>
  </form>
</vxml>
<vxml>
  <form>
    <block>
      <audio
        src="http://resources.tellme.com/audio/earcons/tellmelogo.wav">
        Tellme Logo Audio File
      </audio>
      <goto next = "_home"/>
    </block>
  </form>
</vxml>

```

Тег **<block>**

Описывает блок директив, которые выполняются в порядке, приведенном в документе.

Синтаксис

`<block>` подчиненные или порожденные элементы `</block>`

Элемент `<block>` описывает директивы блока для исполнения в порядке, указанном в текущем документе.

Замечание

Платформа Tellme исполняет интерпретацию текста прямо внутри элемента `<block>` как TTS текст. Нет необходимости заключать этот текст в теги `<audio>`. Тем не менее, рекомендуется, чтобы правила языка соблюдались.

Вызывающие теги: `<form>`.

Порожденные теги: `<assign>`, `<goto>`, `<pause>`, `<submit>`, `<audio>`, `<foreach>`, `<prompt>`, `<var>`, `<clear>`, `<if><else/><elseif/>`, `<return>`, `<disconnect>`, `<log>`, `<script>`.

Пример

Нижеприведенный пример использует элемент `<block>` для того, чтобы присвоить значение переменной.

```
<vxml>
  <form>
    <block>
      <var name="company" expr="'test'"/>
      <assign name="company" expr="'Tellme'"/>
    </block>
  </form>
</vxml>
```

Тег `<catch>`

Ловит события, заданные в приложении VoiceXML, или для платформы Tellme.

Синтаксис

```
<catch event="event"
       count="integer">
  child elements
</catch>
```

где

- ❑ `event` — событие, которое отлавливается (обязательный атрибут).
- ❑ `count` — количество раз, которое событие отлавливается в текущей форме до того момента пока начнет использоваться обработчик этого события. По умолчанию этот счетчик устанавливается равным счетчику предыдущего элемента `<catch>` плюс 1 (необязательный атрибут).

Элемент `<catch>` отлавливает события, определенные пользователем или предопределенные системные события, и указывает, какое действие необхо-

димо выполнить следующим. Имена событий определяются в соответствии с префиксом, так что элемент `<catch>`, определяющий `myevent.test`, будет отслеживать появление события `myevent.test`, `myevent.test.whatever1`, `myevent.test.whatever2` и т. д.

Если обработчик события не приводит к изменению управления процессом звонка, такому как использование директив `<goto>`, `<reprompt>` или `<throw>`, управление передается к следующей форме. Необходимо заметить, что такое поведение является альтернативным, а будущая версия описываемой платформы будет возвращать управление скорее в точку, где событие возникло, чем к следующей форме.

Используйте атрибут `count` в тех случаях, когда вы хотите изменить ответ на событие, которое может обрабатываться много раз. Если элемент `<catch>` имеет счетчик 1, например, платформа использует указанный обработчик событий первый раз. Если `<catch>` имеет счетчик 2, платформа использует его, когда событие обрабатывается второй раз. В общем случае, платформа находит наивысший доступный счетчик, который меньше или равен числу возникновений события в стеке событий текущей формы.

Вызывающие теги: `<field>`, `<menu>`, `<transfer>`, `<form>`, `<subdialog>`, `<vxml>`.

Порожденные теги: `<assign>`, `<goto>`, `<prompt>`, `<throw>`, `<audio>`, `<if><else/><elseif/>`, `<reprompt>`, `<var>`, `<clear>`, `<listen>`, `<return>`, `<disconnect>`, `<log>`, `<script>`, `<foreach>`, `<pause>`, `<submit>`.

Пример

Нижеприведенный пример показывает меню, которое предлагает три выбора. Первый раз вызывающий не говорит ничего, и система повторяет подсказку. Второй раз нет никакого ввода, и система обеспечивает контекстное сообщение и еще раз повторяет подсказку абоненту. Третий раз абонент снова не обеспечивает ввод, и система отключается от него.

```
<vxml>
<form id="launch_missiles">
  <field name="password">
    <prompt>
      <audio>What is the code word?</audio>
    </prompt>
    <grammar>DAY_OF_WEEK</grammar>
    <help>It is a day of the week.</help>
    <catch event="nomatch" count="1"><reprompt/></catch>
    <catch event="noinput" count="1"><reprompt/></catch>
    <catch event="noinput" count="2">
      <audio>Sorry, I didn't hear you</audio>
      <reprompt/>
    </catch>
  </field>
</form>
```

```
</catch>
<catch event="noinput" count="3">
  <disconnect/>
</catch>
<filled/>
</field>
</form>
</vxml>
```

Тег **<choice>**

Задаёт параметры одной строки выбора из меню.

Синтаксис

```
<choice next="URL"
          dtmf="string">
  grammar fragment
</choice>
```

где

- **next** — URL документа (или указатель на внутреннюю часть текущего документа) для перехода, если ввод данных от абонента соответствует фрагменту грамматики (обязательный параметр).
- **dtmf** — число от 1 до 9, которое обслуживается, как эквивалент DTMF-последовательности в элементе **<choice>** (необязательный параметр).

Замечание

Элемент **<menu>** имеет, кроме того, атрибут **dtmf**, и если вы установите этот атрибут в значение **true**, элементы **<choice>** внутри меню будут получать внутренние значения от 1 до 9 в качестве эквивалента DTMF-последовательностей. Если, кроме того, описать атрибут **dtmf** для элемента **<choice>**, то этот атрибут будет превалировать над всеми числами, назначенными **<menu dtmf="true">**.

Например, если имеются пять элементов: **<choice>**, А, В, С, D, Е, и назначены **<choice dtmf="1">** для С и **<menu dtmf="true">**, вызывающий всегда переходит к выбору С при нажатии "1". В этом случае вызывающий не может попасть на выбор А, нажимая "1", а число "3" не соответствует ни одному из выборов. Нажимая "2", "4" и "5", заставляет переходить на В, D, Е соответственно.

Элемент **<choice>** задаёт речь или фрагмент DTMF-грамматики и место для передачи в том случае, если распознавание речи происходит успешно. Элемент **<choice>** работает в сочетании с элементом **<prompt>**, который подсказывает абоненту конкретный вариант ввода в форме списка выборов.

Каждая возможность в списке выбора отображается в элементе `<choice>`: текст между тегами `<choice>` и `</choice>` соответствует одному элементу в списке `<prompt>` и обслуживается как грамматический фрагмент, который является строкой, описывающей одиночное грамматическое выражение, определяющее, что абонент должен сказать, чтобы соответствовать данному выбору.

Замечание

Компилятор грамматики просматривает фрагменты многословной грамматики как одно выражение вне зависимости от того, как это задано в операторе описания грамматики. Например:

```
<choice next="myapp.vxml">green cheese</choice>
```

требует, чтобы абонент сказал "green cheese" в ответ на myapp.vxml. Слова "green" или "cheese" по отдельности не работают. Фрагменты грамматики должны быть описаны маленькими символами. Возможно также описание DTMF-последовательности и задание атрибута с числом от 1 до 9.

Вызывающие теги: `<menu>`.

Порожденные теги: нет.

Пример

Меню предлагает три выбора. Фрагмент грамматики описан для каждого элемента `<choice>` в виде одной конструкции `<prompt>`. Заметим, что пользователь должен сказать "ESPN sports", а не просто "ESPN" или "sports" для того, чтобы распознавание речи сработало в соответствии с первым пунктом меню. Абонент может сказать "Caltech news" или "news", чтобы попасть в третий пункт меню.

```
<vxml>
```

```
  <menu id="three_choice_menu">
```

```
    <prompt>
```

```
      <audio>Welcome. Say E S P N sports, weather, or Caltech news.
```

```
    </audio>
```

```
  </prompt>
```

```
  <choice next="#sports_report">
```

```
    (e s p n sports)
```

```
  </choice>
```

```
  <choice next="#weather_report">
```

```
    weather
```

```
  </choice>
```

```
  <choice next="#news_report">
```

```
    (?caltech news)
```

```
  </choice>
```

```

    <default>
      <reprompt/>
    </default>
  </menu>
<form id = "sports_report">
  <block>
    <audio>The New York Giants will win the Superbowl next year!
  </audio>
    <goto next= "_home"/>
  </block>
</form>
<form id = "weather_report">
  <block>
    <audio>It will be mostly clear and sunny today</audio>
    <goto next= "_home"/>
  </block>
</form>
<form id = "news_report">
  <block>
    <audio>Earthquake in California causes damages in the billions
  </audio>
    <goto next= "_home"/>
  </block>
</form>
</vxml>

```

Тег <clear>

Удаляет значение переменной.

Синтаксис

```
<clear namelist="variable_list"/>
```

Здесь `namelist` — имя освобождаемой переменной (обязательный параметр).

Тег <clear> обнуляет переменную, определенную внутри любой области действия. Вместо:

```

<assign name="city" expr="' '"/>
<assign name="state" expr="' '"/>
<assign name="zip" expr="' '"/>

```

можно задать:

```
<clear namelist="city state zip"/>
```

или:

```
<clear/>
```

Вызывающие теги: `<block>`, `<filled>`, `<if>`, `<prompt>`, `<catch>`, `<foreach>`, `<noinput>`, `<result>`, `<default>`, `<help>`, `<nomatch>`.

Порожденные теги: нет.

Пример

```
<vxml>
  <form>
    <block>
      <var name="answer" expr="'true'"/>
      <if cond="answer == 'true'">
        <log>answer is equal to true => passed
      </log>
      <else/>
        <log>answer is equal to true => failed
      </log>
      </if>
      <clear namelist="answer"/>
    </block>
  </form>
</vxml>
```

Тег **`<confirm>`**

Просит абонента подтвердить ответ.

Синтаксис

```
<confirm name="string">
```

child elements

```
</confirm>
```

Элемент `<confirm>` просит абонента подтвердить ответ. Для использования этого тега нужно установить следующие три атрибута в родительском элементе `<field>`:

- ☐ `confirm`
- ☐ `acceptthresh`
- ☐ `rejectthresh`

Распознаватель назначает внутренние флаги для каждого из событий, которые могут возникнуть. Атрибут `acceptthresh` задает уровень, выше которого

отклонение воспринимается как доверие. Атрибут `rejectthresh` задает уровень, ниже которого расхождение не распознается. Если результирующее доверие попадает в интервал между одним и другим, то отклонение хранится в переменной сеанса `session.confirm.value` и передается элементу `<confirm>`. Элемент `<confirm>` подсказывает абоненту предопределенный текст (который может запросить абонента проверить расхождение).

Если абонент говорит "no", указывая, что проигрывание некорректно, то платформа начинает обрабатывать событие `confirmno`, для которого необходимо определить обработчик события. Если абонент говорит "yes", расхождение запоминается в переменной `<field>` и обрабатывается обработчиком, заданным для тега `<field>`.

Замечание

Неправильно использовать тег `<confirm>` как `<catch event="confirm">`.

Вызывающие теги: `<field>`.

Порожденные теги: `<assign>`, `<goto>`, `<noinput>`, `<throw>`, `<audio>`, `<help>`, `<pause>`, `<var>`, `<clear>`, `<if><else/><elseif/>`, `<prompt>`, `<disconnect>`, `<listen>`, `<reprompt>`, `<foreach>`, `<nomatch>`, `<script>`.

Пример

```
<vxml>
<form id="main">
  <field name="choice"
    endseconds="0.75"
    timeout="2.0"
    confirm="yes"
    acceptthresh="70"
    rejectthresh="30">
    <grammar>
      DAY_OF_WEEK
    </grammar>
    <prompt>
      <audio>Say the name of your favorite day
    </audio>
    <pause>1500</pause>
    </prompt>
  <confirm>
    <prompt>
      <audio>Did you say</audio>
    <pause>200</pause>
```

```

    <audio>{session.confirm.value}</audio>
  </prompt>
</confirm>
  <catch event="confirmno">
    <audio>my mistake</audio>
    <reprompt/>
  </catch>
  <filled>
    <audio>you just said {choice}
  </audio>
  <pause>1000</pause>
  <audio>happy {choice}
  </audio>
  <reprompt/>
</filled>
<default>
  <log>other event</log>
  <reprompt/>
</default>
</fied>
</form>
</vxml>

```

Тег <debug>

Пишет вывод в Tellme Studio Debug Log.

Синтаксис

<debug> текст, который нужно писать в Debug Log </debug>

Элемент <debug> позволяет писать вывод, включая все переменные, из контейнеров исполняемого содержания (<block>, <iniccstial> и обработчики событий) в файл Tellme Studio Debug Log.

Этот тег можно использовать внутри любого элемента, порожденного тегом <vxml> за исключением <form> и <script>. Внутри элемента <script> нужно использовать vxmllog(string).

Порожденные теги: <value> .

Пример

Нижеприведенный код порождает следующие строки в файле Debug Log:

```
[06/04/2000:17:22:37 DST] DEV : 000190 JAVASCRIPT ***** 409
```

```
[06/04/2000:17:22:37 DST] DEV : 000190 JAVASCRIPT ***** 252.1313
[06/04/2000:17:22:37 DST] DEV : 000190 JAVASCRIPT /252/
[06/04/2000:17:22:37 DST] DEV : 000190 JAVASCRIPT -1
<vxml>
  <var name="myapp_number" expr="'252.1313'"/>
  <var name="myapp_badareacodes" expr="'409'"/>
  <block>
    <script type="text/javascript">
      <![CDATA[
        var number = document.myapp_number;
        var badareacodes = document.myapp_badareacodes;
        vxmlllog("***** " + badareacodes);
        vxmlllog("***** " + number);
        regexp = new RegExp(number.substr(0,3));
        vxmlllog(regexp.toString());
        vxmlllog(badareacodes.search(regexp));
      ]]>
    </script>
    <goto next="_home"/>
  </block>
</vxml>
```

Тег **<default>**

Кэширует все события, не обработанные к текущему моменту другими обработчиками событий.

Синтаксис

```
<default> child elements </default>
```

Замечание

<default> является коротким вариантом команды <catch event="default">.

Обработчик события <default> подхватывает любое событие, которое не было обработано ни одним из объявленных.

Вызывающие теги: <field>, <menu>, <transfer>, <form>, <subdialog>, <vxml>.

Порожденные теги: <assign>, <goto>, <prompt>, <throw>, <audio>, <if><else></elseif>, <reprompt>, <var>, <clear>, <listen>, <return>, <disconnect>, <log>, <script>, <foreach>, <pause>, <submit>.

Пример

Пример показывает тег `<default>`, используемый для обработки событий `nomatch` и `noinput`:

```
<vxml>
  <form id="launch_missiles">
    <field name="missiles_password">
      <prompt>
        <audio>What is the code word?
      </audio>
    </prompt>
    <grammar>DAY_OF_WEEK</grammar>
    <help>It is a day of the week.</help>
    <filled>
      <audio>you said {missiles_password}
    </audio>
    <goto next="_home"/>
    </filled>
    <default><reprompt/></default>
  </field>
</form>
</vxml>
```

Тег **`<disconnect>`**

Завершает телефонный вызов.

Синтаксис

```
<disconnect/>
```

Директива `<disconnect>` заставляет систему завершить телефонное соединение. Если для соединения используется директива `<transfer>` с целью вызвать другое приложение, которое использует директиву `<disconnect>`, управление возвращается в основное приложение.

Примечание

Тег `<disconnect>` генерирует событие `event.disconnect.hangup`, которое в текущей версии языка не может использоваться после того, как вызов завершен. Это ограничение будет преодолено в будущей версии платформы Tellme.

Вызывающие теги: `<block>`, `<filled>`, `<if><else/><elseif/>`, `<prompt>`, `<catch>`, `<foreach>`, `<noinput>`, `<result>`, `<default>`, `<help>`, `<nomatch>`.

Порожденные теги: нет.

Пример

```
<vxml>
  <form>
    <field name="myapp_none" timeout="1">
      <prompt>
        <audio>Hello</audio>
      </prompt>
      <noinput>
        <audio>Sorry, I didn't hear you. I'm hanging up now.Good-bye.
        </audio>
      </noinput>
    </field>
  </form>
</vxml>
```

Тег <dtmf>

Определяет разрешенные клавиши DTMF в текущем взаимодействии с абонентом.

Синтаксис

```
<dtmf name="string"
      src="URL">
</dtmf>
<dtmf>
  <![CDATA[
    grammar description or name of an intrinsic grammar
  ]]>
</dtmf>
```

где

□ name — имя грамматики (не обязательно).

□ src — URL внешнего файла грамматики (не обязательно).

Элемент <dtmf> позволяет определять правильные DTMF-последовательности или ссылаться на внешние грамматики, в которых заданы DTMF-последовательности. Используется для того, чтобы определить ввод пользователя, которому доступна только тоновая клавиатура. При использовании элемента <dtmf> ускоряется время обработки, так как платформе не требуется записывать голос абонента и запускать процедуру распознавания

голоса. Заметим, что любые слова, определенные внутри <dtmf>-элемента, игнорируются.

При совпадении введенных последовательностей заданным вариантам ввода, значение, заданное в директиве option, возвращается платформе. Платформа использует это значение для определения следующей последовательности событий голосового VoiceXML-приложения.

Вызывающие теги: <field>, <form>, <link>, <transfer>, <vxml>.

Порожденные теги: нет.

Пример

Нижеприведенный пример показывает, как писать внутреннюю грамматику, используя элемент <grammar> и тег CDATA. Так как грамматика определена внутри элемента <vxml>, эти высказывания имеют область действия всего документа:

```
<vxml>
  <form id="transfer">
    <transfer maxlength="60" dest="8005558355">
      <dtmf>
        <![CDATA[
          [
            [(dtmf=0 dtmf=0)] {<option "hangup">}
          ]
        ]]>
      </dtmf>
      <catch event="event.busy">
        <audio> busy </audio>
        <goto next="_home"/>
      </catch>
      <catch event="event.noanswer">
        <audio>no answer</audio>
        <goto next="_home"/>
      </catch>
      <catch event="event.transfer.maxlen.exceeded">
        <audio>max length exceeded</audio>
        <goto next="_home"/>
      </catch>
      <catch event="event.transfer.failure">
        <audio> failed </audio>
        <goto next="_home"/>
      </catch>
```

```

<default>
  <audio>done</audio>
  <goto next="_home"/>
</default>
</transfer>
</form>
</vxml>

```

Тег **<error>**

Обрабатывает ошибки, сформированные платформой или заданные в элементе `<throw>`.

Синтаксис

```
<error> child elements </error>
```

Замечание

`<error>` является коротким написанием `<catch event="error">`.

Обработчик события `<error>` подхватывает события `error`, возникшие в платформе или заданные в элементе `<throw>`. При определении события в элементе `<throw>`, нужно использовать префикс `"error."`: `<throw event="error.myevent"/>`.

В настоящее время платформа поддерживает следующие виды заданных пользователем событий:

- ☐ `error.semantic` — возникает при использовании элемента `<return>` без `<subdialog>`;
- ☐ `error.badfetch` — возникает при невозможности выполнить запрос HTTP GET.

Вызывающие теги: `<field>`, `<subdialog>`, `<menu>`, `<transfer>`.

Порожденные теги: `<assign>`, `<foreach>`, `<pause>`, `<throw>`, `<audio>`, `<goto>`, `<reprompt>`, `<var>`, `<clear>`, `<if>``<else/>``<elseif/>`, `<return>`, `<disconnect>`, `<listen>`, `<script>`.

Пример

Пример показывает, как использовать событие `<error>` в сочетании с другими обработчиками событий.

```

<vxml>
  <form>
    <field name="storename">
      <!-- The grammar source is invalid. The platform will throw
           error.badfetch. -->

```

```

<grammar src="http://badlink.txt"/>
<prompt>
  <audio>Say dogs, plants, or help</audio>
</prompt>
<nomatch>
  <audio>Sorry, I didn't get that</audio>
  <reprompt/>
</nomatch>
<noinput>
  <audio>Sorry, I didn't get that</audio>
  <reprompt/>
</noinput>
<help>
  <audio>You are in Buy Me. Please choose a type of store.</audio>
  <reprompt/>
</help>
<default/>
<error>
  <goto next="errorhandling.vxml#form_id"/>
</error>
</field>
</form>
</vxml>

```

Тег **<exit>**

Возвращает управление к последнему документу, добавленному в стек **<gosub>**.

Примечание

Для того чтобы соответствовать спецификации VoiceXML Forum Version 1.0, платформа поддерживает элемент **<return>** для использования совместно с **<subdialog>**. Этот текст описывает альтернативную версию Tellme элемента **<exit>**.

```
<exit expr="string"/>
```

где

expr — событие, возникающее для элемента **<gosub>** в стеке **gosub** (обязательный параметр).

Директива **<exit>** возвращает управление к последнему приложению, добавленному в стек **<gosub>**. Когда директива **<exit>** встречается в форме,

событие, которое было описано атрибутом `expr`, возникает в элементе `<gosub>` из стека `<gosub>`. В этом случае необходимо предусмотреть подхватывание этого события для обеспечения продолжения процесса обработки.

Вызывающие теги: `<block>`, `<filled>`, `<if><else/><elseif/>`, `<prompt>`, `<catch>`, `<foreach>`, `<noinput>`, `<result>`, `<default>`, `<help>`, `<nomatch>`.

Порожденные теги: нет.

Пример

gateway.vxml:

```
<vxml>
  <form id="intro">
    <gosub next="login.vxml#GetPin">
      <catch event="login.failed">
        <goto next="failure.vxml"/>
      </catch>
      <catch event="login.success">
        <goto next="success.vxml"/>
      </catch>
    <default>
      <goto next="unexplainedevent.vxml"/>
    </default>
  </gosub>
</form>
</vxml>
```

login.vxml:

```
<vxml>
  <form id="GetPin">
    <field name="pin">
      <grammar>Four__digits
    </grammar>
    <prompt>Please enter your pin
    </prompt>
    <noinput>
      <exit expr="login.failed"/>
    </noinput>
    <filled>
      <if cond="{pin} == 5555">
        <exit expr="login.success"/>
      </if>
```

```

    <exit expr="login.invalid"/>
  </filled>
</field>
</form>
</vxml>

```

Тег *<field>*

Описывает ввод информации абонентом.

Синтаксис

```

<field name="variable"
      confirm="yes_no"
      errorhandler="URL"
      timeout="seconds"
      timeoutondtmf="true_false"
      acceptthresh="integer"
      rejectthresh="integer"
      rejectthreshhold="integer"
      bargein="true_false"
      bargeinlevel="integer"
      endseconds="seconds"
      magicword="true_false"
      pruning="integer"
      phoneticpruning="true_false">
  child elements
</field>

```

Замечание

Используйте `<property>` для управления параметрами речи, которые более не поддерживаются альтернативными атрибутами `<field>`.

Здесь

- ❑ `name` — переменная, которая хранит результат распознавания, если распознавание помечено как успешно завершенное (обязательный параметр).
- ❑ `confirm` — эта команда включает или выключает элемент `<confirm>`. Когда установлена на "yes", элемент `<confirm>` включен. Когда "no" (по умолчанию) — элемент `<confirm>` игнорируется.

Если доверительная область обрабатываемого высказывания ниже, чем `rejectthresh`, высказывание отбрасывается и возникает событие `nomatch`. Если оно находится между `rejectthresh` и `acceptthresh`, управление под-

хватывается элементом `<confirm>`. Если оно выше `acceptthresh`, высказывание принимается, как распознанное соответствие без перехода к элементу `<confirm>` (не обязателен).

- ❑ `timeout` — интервал времени, в течение которого платформа ждет ввода от абонента, прежде чем создать событие `noinput`. По умолчанию этот интервал устанавливается платформой (не обязателен).
- ❑ `timeoutondtmf` — установка периода, в течение которого платформа ожидает DTMF-последовательность или абонент завершит ввод, начав голосовое сообщение. Когда установлен `"true"` (по умолчанию), сначала предполагается использовать межцифровой тайм-аут для DTMF прежде, чем корректная интерпретация приведет абонента к следующему состоянию диалога. Когда установлено `"false"`, любая корректная интерпретация DTMF немедленно отправляет абонента в следующее состояние диалога.

Например, если вы установили `imeoutondtmf` на `"false"` и определили ввод с клавиатуры `"0"` в качестве ответа вызывающего на проигрывание главного меню, попытка вызывающего ввести `"08873"` в качестве zip-кода немедленно приведет к возврату в главное меню. Если установлено `"true"`, пользователь ждет несколько секунд, прежде чем будет направлен в главное меню после нажатия `"0"`, но будет также время найти `"8"` на клавиатуре в случае необходимости.

- ❑ `acceptthresh` — доверительный уровень `[0...100]` сверху, который автоматически принимает высказывание пользователя для грамматического разбора. По умолчанию равен 90.

Если `confirm="no"`, этот атрибут игнорируется. Если `confirm="yes"` и этот атрибут не описан, используется правило умолчания.

- ❑ `rejectthresh` — доверительный уровень снизу `[0...100]`, который автоматически отбрасывает высказывание пользователя, как находящееся за пределами грамматики. По умолчанию 45. Если `confirm="no"`, этот атрибут игнорируется. Если `confirm="yes"` и атрибут не описан, используется правило по умолчанию.
- ❑ `rejectthreshold` — доверительный уровень поля `{0...100} <field>`. Если доверительная область для высказывания ниже `rejectthreshold`, высказывание отбрасывается.

Заметим, что этот атрибут описывает одно значение, с которым сравнивается высказывание, то есть высказывание либо отбрасывается, либо принимается.

Атрибуты `rejectthresh` и `acceptthresh` описывают диапазон, в который может попасть область действия высказывания и работает в сочетании с атрибутом `confirm`. Если результат попадает в диапазон, управление передается элементу `<confirm>`.

Примечание

Существуют значения системных умолчаний для `acceptthresh`, `rejectthresh` и `rejectthreshold`, которые оптимизируются для наиболее общих случаев.

- ❑ `bargein` — этот параметр позволяет задать, может ли пользователь прерывать систему (по умолчанию `"true"`). Если параметр имеет значение `"false"`, `bargein` выключен, а пользователь должен слушать все системные подсказки. Установка атрибута на `"false"` может использоваться для коротких подсказок, когда вы хотите, чтобы пользователь их слышал.
- ❑ `bargeinlevel` — уровень громкости, который нужен пользователю, чтобы переключить триггер `bargein`.
- ❑ `endseconds` — длительность интервала времени, в течение которого распознаватель ожидает внутри паузы говорящего, предполагая, что вызывающий абонент закончил свое сообщение (по умолчанию 1 секунда). Это очень чувствительный параметр. Если значение слишком низко, система будет усекать высказывание до того момента, как абонент завершит речь. Это может привести к неправильной активации событий в интерпретаторе по причине отсутствия достаточной информации для поиска нужной грамматики. Если уровень установлен слишком высоко, то система будет производить впечатление слишком медленной.

Краткое руководство по работе с этим атрибутом.

- ❑ Для высказываний, в процессе которых ожидается естественная пауза, например, как при вводе телефонного номера, `"650...555...1212"`, рекомендуется значение 1.0.
- ❑ Для высказываний, в процессе которых не ожидается естественной паузы между словами, рекомендуется значение параметра `endseconds` между 0,5 и 0,75.

Замечание

Атрибут `endseconds` отражает только тайм-аут речи. При ожидании ввода цифр нужно использовать атрибут `timeoutondtmf`.

- ❑ `magicword` — установка магического слова на `"on"` или `"off"`. Слово "магическое" означает способность системы продолжать проигрывание подсказки при одновременном сравнении высказывания с активной грамматикой.

Когда `magicword` установлен в значение `"true"` (по умолчанию `"false"`), `bargein` работает медленно. Старт речи не прерывается текущей подсказкой, и если высказывание не распознается, система продолжает проигрывать подсказку, а не отвечает о трудностях распознавания `<nomatch>`. Если фраза распознается, система определяет, какое состояние будет следующим, останавливает проигрывание текущей подсказки, затем предоставляет абоненту новую информацию.

`Magicword` является очень надежным вариантом общения для борьбы с шумом, однако, оно также приводит к медленной скорости распознавания речи. Его лучше использовать для доставки содержания, такого как погода или новости. Если абонент говорит что-то в течение отчета, система продолжает

проигрывать отчет, пока распознаватель сравнивает высказывание с активной грамматикой.

Система прерывает проигрывание только в том случае, когда процесс распознавания завершается успешно. Заметим, что сложность грамматики прямо влияет на скорость ответа.

Замечание

Любой ввод DTMF имеет больший приоритет над состоянием `magic word` в рамках текущей подсказки. Если ввод DTMF не соответствует активной грамматике, возникает событие `nomatch`. Грамматика области приложения является активной в процессе состояния `magic word` в дополнение к локальной грамматике.

- ❑ `pruning` — имеет значения в диапазоне [400...5000], используется в поиске Viterbi search по акустическим моделям. Параметр определяет длину пути в процессе каждой итерации и используется для управления точностью и скоростью. Так, значение параметра 5000 — наиболее точно, а значение 400 дает максимальную скорость.

Заметим, что платформа Tellme использует оптимальные установки по умолчанию для наиболее часто встречающихся сценариев (не обязателен).

- ❑ `phoneticpruning` — установка фонетического `pruning`. Когда установлен в значении "true" (по умолчанию "false"), `phoneticpruning` заставляет сервер распознавания выполнять дополнительные вычисления для того, чтобы исключить маловероятные гипотезы, касающиеся метода распознавания. Это свойство в процессе использования больших грамматик позволяет значительно сократить время распознавания при небольшом ухудшении точности.

Рекомендуется использовать эту возможность, только если выполняется работа с большой грамматикой, такой как грамматика имени города, когда можно заранее определить эффект уменьшения времени распознавания (не обязателен).

Элемент `<field>` определяет.

- ❑ Переменную с атрибутом `name`, чтобы хранить результат распознавания. Когда высказывание абонента соответствует активной грамматике, результат распознавания хранится в переменной элемента `<field>`.
- ❑ Одна или более грамматик, которые используются для интерпретации ввода пользователя. Все грамматики, определенные внутри элемента `<field>`, имеют область действия данного поля. Это означает, что они активны только в рамках конкретного поля.
- ❑ Один или более элементов `<prompt>`, описывающих TTS-файлы или звуковые файлы (.wav), для проигрывания абоненту.
- ❑ Обработчики событий для событий `<noinput>`, `<nomatch>`, `<help>` или `<filled>`. Также имеется возможность указать стандартный обработчик

событий `<default>` для неожиданных случаев. Вы можете иметь несколько сущностей `<prompt>`, `<noinput>` или `<nomatch>`. На последующих итерациях по полю (после директив `<reprompt/>`), если один из этих обработчиков включается снова, используется следующий обработчик.

Замечание

Не следует использовать несколько обработчиков типа `<default>`.

Вызывающие теги: `<form>`.

Порожденные теги: `<catch>`, `<filled>`, `<noinput>`, `<confirm>`, `<grammar>`, `<nomatch>`, `<default>`, `<help>`, `<prompt>`.

Пример

Представленный ниже диалог показывает одно из возможных взаимодействий между абонентом и платформой Tellme:

Tellme: Играет prompt 1.

Абонент: Не выполняет ввод; возникает событие `noinput`.

Tellme: Выполняет первый обработчик `<noinput>`, проигрывая prompt 2.

Абонент: Выполняет ввод, который не соответствует грамматике, и возникает событие `nomatch`.

Tellme: Выполняет первый обработчик события `<nomatch>`, проигрывая prompt 2.

Абонент: Выполняет ввод, который не соответствует грамматике, второй раз; снова возникает событие `nomatch`.

Tellme: Выполняет второй обработчик событий `<nomatch>`, проигрывая prompt 2.

Абонент: Выполняет ввод, который не соответствует грамматике третий раз; снова возникает событие `nomatch`.

Tellme: Выполняет второй обработчик событий `<nomatch>` снова, проигрывая prompt 2.

`<vxml>`

```
<form id="top">
  <field name="myapp_input">
    <grammar>
      <![CDATA[
        [
          [pet (pet store) (pet shop)] {<option "pets">}
          [flower (flower shop) (flower store)] {<option "flowers">}
        ]
      ]]>
```

```
</grammar>
<prompt><audio>prompt 1</audio></prompt>
<prompt><audio>prompt 2</audio></prompt>
<nomatch count="1"><audio>first nomatch</audio><reprompt/></nomatch>
<nomatch count="2"><audio>second nomatch</audio><reprompt/></nomatch>
<noinput><audio>first noinput</audio><reprompt/></noinput>
<filled>
  <audio> you said {myapp_input} </audio>
  <goto next="#top"/>
</filled>
</field>
</form>
</vxml>
```

Тег **<filled>**

Описывает действие, которое выполняется в том случае, когда переменной поля `<field>` назначается значение.

Синтаксис

```
<filled> child elements </filled>
```

Элемент `<filled>` содержит исполняемое содержание, когда высказывание абонента соответствует активной грамматике.

Каждый раз, когда абоненту выполняется подсказка, необходимо писать обработчик событий `<filled>` хотя бы с одной директивой, говорящей системе что делать в том случае, когда успешное распознавание завершится.

При написании пустого элемента `<filled>`, управление будет передано к следующему элементу поля в форме. Если в форме больше нет элементов, управление передается к следующей форме, однако такое поведение является временным расширением стандарта. В будущих версиях выполнение будет завершаться в тех ситуациях, когда следующая форма для передачи управления не будет задана.

Вызывающие теги: `<form>`, `<field>`, `<subdialog>`.

Порожденные теги: `<assign>`, `<goto>`, `<pause>`, `<script>`, `<audio>`, `<if><elseif><else/>`, `<prompt>`, `<submit>`, `<clear>`, `<listen>`, `<reprompt>`, `<throw>`, `<disconnect>`, `<log>`, `<result>`, `<var>`, `<foreach>`, `<goto>`, `<return>`.

Пример

Пример обработки распознавания проигрывает TTS-звуковой файл, когда высказывание соответствует грамматике и затем выполняется переход к форме `CallPhoneModule`.

```

<vxml>
  <form id="GetPhoneNum">
    <field name="phonemodule_number"
      rejectthreshold="30"
      bargeinlevel="10">
      <grammar>US_10_DIGIT_PHONE_NUMBER</grammar>
      <prompt>
        <audio>say your phone number, starting with the area code
        </audio>
      </prompt>
      <default>
        <audio>sorry?</audio>
        <reprompt/>
      </default>
      <filled>
        <audio>got it</audio>
        <pause>500</pause>
        <goto next="#CallPhoneModule"/>
      </filled>
    </field>
  </form>
</vxml>

```

Тег **<foreach>**

Выполняет итерацию по списку элементов.

Синтаксис

```

<foreach item="variable"
  in="comma_delimited_list"
  array="JS_expression">
  child elements
</foreach>

```

где

- ❑ **item** — переменная, в которой временно хранится каждый элемент из итерируемого списка (обязательный).
- ❑ **in** — разделенный запятыми список элементов, по которому выполняется итерация. Этот список может быть смешанным и состоять из других строк и переменных.

Если переменная строится с помощью метода `vxmldata.append()`, возвращаются все составляющие переменной (обязателен, когда не используется массив).

□ array — интерпретируемый массив JavaScript (обязательный).

Элемент `<foreach>` позволяет выполнять итерацию по всем элементам списка. Наиболее общее использование `<foreach>` заключается в том, чтобы проиграть серию звуковых подсказок `<audio>`.

Вызывающие теги: `<block>`, `<filled>`, `<if><elseif/><else/>`, `<prompt>`, `<catch>`, `<foreach>`, `<noinput>`, `<result>`, `<default>`, `<help>`, `<nomatch>`.

Порожденные теги: `<assign>`, `<goto>`, `<prompt>`, `<throw>`, `<audio>`, `<if><elseif/><else/>`, `<reprompt>`, `<var>`, `<clear>`, `<listen>`, `<return>`, `<disconnect>`, `<log>`, `<script>`, `<foreach>`, `<pause>`, `<submit>`.

Пример

Если `dialog.color_list` содержит "red, blue, yellow", то полный список описанный в `in` будет "purple, red, blue, yellow, blue".

```
<assign name="dialog.color_list" expr="'red, blue, yellow'"/>
<foreach item="dialog.item" in="purple, {dialog.color_list}, blue">
<audio>{dialog.item}</audio>
</foreach>
```

Этот пример показывает, как строить список с использованием JavaScript, а затем проигрывать все элементы в списке посредством применения элемента `<audio>`.

```
<vxml>
<form id="readlist">
  <block>
    <var name="mylist" expr="new Array"/>
    <script type="text/javascript">
      <![CDATA[
        mylist[0] = "Utah Jazz";
        mylist[1] = "San Antonio Spurs";
        mylist[2] = "Minnesota Timberwolves";
        mylist[3] = "Denver Nuggets";
        mylist[4] = "Dallas Mavericks";
        mylist[5] = "Houston Rockets";
        mylist[6] = "Vancouver Grizzlies";
      ]]>
    </script>
    <audio>Here are the teams in the N B A midwest conference</audio>
    <pause>1000</pause>
    <foreach item="demo_item" array="mylist">
      <audio>
        <value expr="demo_item"/>
```

```

    </audio>
  </foreach>
  <pause>1000</pause>
  <audio>That's all!</audio>
  <goto next="_home"/>
</block>
</form>
</vxml>

```

Этот код использует `<foreach>` для итерации по списку.

```

<vxml>
  <form>
    <block>
      <!-- literal string -->
      <foreach item="item1" in="red, blue, green">
        <audio><value expr="item"></audio>
      </foreach>

      <!-- comma delimited string in curly brace notation -->
      <var name="color1" expr="'red'"/>
      <var name="color2" expr="'blue'"/>
      <var name="color3" expr="'green'"/>
      <foreach item="item2" in="{color1}, {color2}, {color3}">
        <audio><value expr="item2"></audio>
      </foreach>

      <!-- combination: delimited string and multi-item list -->
      <var name="color_list" expr="newArray()"/>
      <script type="text/javascript">
        <![CDATA[
          color_list[0]='red';
          color_list[1]='blue';
          color_list[2]='green';
        ]]>
      </script>
      <foreach item="item3" array="color_list">
        <audio><value expr="item3"></audio>
      </foreach>
      <goto next="_home"/>
    </block>
  </form>
</vxml>

```

Тег **<form>**

Содержит последовательность элементов поля, которые должны быть заполнены при взаимодействии с абонентом.

Синтаксис

```
<form id="string"
      anchor="true_false"
      errorhandler="URL">
```

child elements

```
</form/>
```

где

- ❑ `id` — имя формы. Заметим, что можно использовать `id` в качестве ссылки для перехода по директиве `<goto>` (обязательный параметр).
- ❑ `anchor` — установка для передачи управления в тех случаях, когда элемент `<form>` добавляется к списку истории. Когда этот атрибут установлен в `true`, его URL добавляется в список истории. Когда установлен в `false` (по умолчанию), URL формы не добавляется к списку (не обязательный).

Внутренняя переменная `variable _lastanchor` указывает на URL последнего элемента в форме, добавленного в этот список. Чтобы перейти на этот элемент, нужно задать `<goto next="_lastanchor"/>`.

После перехода к форме, ее URL удаляется из списка, а `_lastanchor` указывает на предыдущий URL. Когда первый якорек достигается, последующий использует `<goto next="_lastanchor">` для указания на тот же самый URL.

Замечание

Если используется `<goto next="_lastanchor">` внутри формы, чей атрибут `anchor` установлен в `true`, управление передается на последний якорек, описанный прежде, чем вы вошли в текущую форму.

Элемент `<form>` содержит всю логику голосового приложения, включая грамматики, подсказки абоненту и исполняемый код, основываясь на ответе абонента. Платформа Tellme выполняет элементы `<form>` в том порядке, в каком они приведены в документе до тех пор, пока не встретится директива, изменяющая естественный процесс выполнения. Например, можно описать директиву `<goto>`, которая при успешном завершении процесса распознавания будет переходить к другой форме. Обратите внимание, что такое поведение является нестандартным. В будущем, если не будет использоваться элемент `<goto>` для указания на следующую форму, исполнение будет завершаться.

Элемент `<form>` содержит одно или более элементов `<field>`. Обычно элементы `<field>` содержат обработчики событий в ответ на ввод данных от абонента.

Можно также использовать событие `<filled>` как порожденный элемент `<form>`, чтобы поймать распознавание в соответствии с грамматиками формы, документа или всего приложения.

Замечание

Если элемент `<filled>` внутри `<form>` приходит прежде, чем элемент `<filled>` внутри поля `<field>`, то элемент `<filled>` уровня формы имеет преимущество.

Вызывающие теги: `<vxml>`.

Порожденные теги: `<block>`, `<filled>`, `<noinput>`, `<subdialog>`, `<catch>`, `<grammar>`, `<nomatch>`, `<transfer>`, `<default>`, `<help>`, `<result>`, `<var>`, `<field>`, `<initial>`, `<script>`.

Пример

Элемент `<form>` содержит один элемент `<field>`, который дает пользователю выбор слушать больше bloopers, изменять их предпочтения при прослушивании или возвращаться в главное меню платформы Tellme. Элемент `<field>` содержит два обработчика событий:

- ❑ `<filled>`, предназначен для обработки результата распознавания;
- ❑ `<default>`, служит для обработки любого другого рода, когда ввод не был распознан.

Заметим, что атрибутами элемента `<vxml>` является набор Tellme Studio универсальных URL, который определяет фразу TELLME MENU в элементе `<link>`. Элемент `<link>` обрабатывает успешное распознавание для TELLME MENU, вот почему нет необходимости использовать специальный обработчик событий в этом случае.

```
<vxml application="http://resources.tellme.com/lib/universals.vxml">
<!--... other code can be here... -->
<form id="askForMore">
  <field name="bloopers_choice"
    timeout="4.0"
    bargeinlevel="10"
    endseconds="1.0"
    rejectthreshold="30">
<grammar>
  <![CDATA[
    [
      [more] {<option "more">}
      [change] {<option "change">}
```

```
    ]
  ]]>
</grammar>
<prompt>
  <audio>
    To hear another blooper, say MORE. To change blooper preference,
    say CHANGE. If you've had your fill, say TELLME MENU.
  </audio>
</prompt>
<default>
  <audio>sorry?</audio>
</default>
<filled>
  <result name="more">
    <goto next="#personalBloopers"/>
  </result>
  <result name="change">
    <goto next="#askForPrefs"/>
  </result>
</filled>
</field>
</form>
<form id="personalBloopers">
  <block>
    <audio>This is personal bloopers.</audio>
    <!--... other code can be here... -->
    <goto next="#askForMore"/>
  </block>
</form>
<form id="askForPrefs">
  <block>
    <audio>To change blooper preference.</audio>
    <!--... other code can be here... -->
    <goto next="#askForMore"/>
  </block>
</form>
<!--... other code can be here... -->
</vxml>
```

Тег `<gosub>`

Переходит к URL для специальной обработки приложения.

Примечание

Пожалуйста, перепишите ваш код с использованием `<subdialog>` в сочетании с `<return>` вместе с `<gosub>` и `<exit>`.

Синтаксис

```
<gosub next="URL"/>
```

Здесь `next` — URL этого документа (или ссылка внутри текущего документа) для перехода на него.

Замечание

Минимально использовать обработчик событий `<default>` внутри элемента `<gosub>`.

Элемент `<gosub>` задает URL для перехода к подпрограмме, аналогичен директиве `<goto>`. Однако существует важное различие. Для того чтобы вернуться из указанного файла, необходимо создать событие "exit" внутри этого файла с использованием директивы `<exit>`. Необходимо также написать обработчик событий внутри первичного элемента `<gosub>` для того, чтобы поймать каждое событие, которое может возникнуть в вызываемом файле.

Когда платформа встречается с элементом `<gosub>`, событие кладется в стек `<gosub>`, который следит, из какого файла и когда был выполнен конкретный элемент. Затем, когда директива `<exit>` встречается в другом файле, событие, которое было описано атрибутом `expr` пойманного последнего элемента `<gosub>` добавляется в стек `<gosub>`.

Если используются элементы `<gosub>` для движения от файла к файлу, необходимо помнить порядок, в котором они добавлялись в стек `<gosub>`. Необходимо также порождать события для возврата каждого элемента `<gosub>` в обратном порядке. Также необходимо следить за тем, чтобы не перенагружать программу инструкциями `<goto>` и `<gosub>`. Если выполняется переход к файлу, который отмечен как события для выхода и не существует подходящего обработчика, определенного в последнем элементе `<gosub>`, добавленном в стек, платформа будет выдавать ошибку.

Элемент `<gosub>` аналогичен `gosub` или вызову функции в других процедурных языках, но с некоторым ограничением, таким как неспособность передавать параметры.

Вызывающие теги: `<form>`.

Порожденные теги: <catch>, <help>, <nomatch>, <default>, <noinput>, <prompt>, <filled>.

Пример

```
<vxml>
  <form id="GetPin">
    <field name="pin">
      <grammar>Four_digits</grammar>
      <prompt>
        <audio>Please enter your pin</audio>
      </prompt>
      <noinput>
        <exit expr="login.invalid"/>
      </noinput>
      <filled>
        <if cond="Number(pin) &lt; 9999">
          <exit expr="login.success"/>
        </if> <exit expr="login.invalid"/>
      </filled>
    </field>
  </form>
</vxml>
```

Тег <goto>

Переходит к заданному URL.

Синтаксис

```
<goto next="URL"
      nextitem="string"
      submit="string"
      method="get_post"
      expr="JS_expression"
      expritem="JS_expression"/>
```

Здесь next — URL документа (или ссылка внутри текущего документа). Следует иметь в виду, что ссылка на адрес поля или блока внутри той же самой формы не разрешена.

Замечание

Устанавливает next на внутреннюю переменную:

- `_home`, если нужно вернуться в главное меню Tellme;
- `_lastanchor`, если нужно вернуться к последнему элементу <form>, который имеет собственный атрибут `anchor`, установленный в `true`.

- ❑ `nextitem` — имя элемента поля для перехода внутри текущей формы.
- ❑ `submit` — список переменных, указанных в `http`-запросе, при получении нового документа с использованием метода `POST`. По умолчанию является множеством всех переменных полей (`<field>` и `<transcribe>`), если элемент `<goto>` находится в форме, и является пустым множеством, если элемент `<goto>` расположен где-то в другом месте.

Атрибут `submit` работает только когда `method="post"`. Синтаксис `submit="string"`, который задается как посылка данных по запросу к серверу. Эта строка может включать символьные подстановки:

```
<goto next="http://myserver/myapp.cgi" method="post"
submit="{document.myapp.the_data}"/>
```

- ❑ `method` — метод применяемого запроса: `"GET"` (по умолчанию) или `"POST"`. Если данные, посылаемые к серверу велики по объему, обычно лучше использовать `POST`, так как при использовании `GET` могут быть ограничения по длине URL.

Замечание

При использовании метода `POST` требуется указывать `submit`. Например, `<goto next=http://foolbar.pl method="post" submit="{document.allmypostdata}"/>`.

- ❑ `expr` — выражение JavaScript, которое оценивает URL документа. Заметим, что переход внутри формы документа не разрешен (обязателен, если `next`, `nextitem` или `expritem` не указаны).

Замечание

Устанавливает `expr` на внутренние переменные, когда нужно вернуться в главное меню `Tellme`, `_lastanchor`, если нужно вернуться к последнему элементу `<form>`, который имел атрибут `anchor`, установленный в `true`.

- ❑ `expritem` — выражение JavaScript, которое оценивает имя элемента поля при переходе внутри текущей формы (обязательно, если `next`, `nextitem` или `expr` не указаны).

Элемент `<goto>` передает управление к URL, заданном атрибутом `next`. При переходе к новому документу передача параметров выполняется с использованием указания переменных. Если текущий и следующий документ относятся к одному и тому же приложению (т. е. имеют один корневой документ), переменные корневого документа также передаются, но переменные текущего документа будут потеряны. Для того чтобы сохранить эти локальные переменные, нужно копировать их в переменные уровня приложения.

Вызывающие теги: `<block>`, `<filled>`, `<if><else/><elseif/>`, `<prompt>`, `<catch>`, `<foreach>`, `<noinput>`, `<result>`, `<default>`, `<help>`, `<nomatch>`.

Порожденные теги: нет.

Пример

Пример перехода к другой форме:

```
<goto next="#form_id"/>
```

Пример перехода к другому файлу:

```
<goto next="http://flight/reserve_seat.vxml"/>
```

Пример получения формы из другого файла:

```
<goto next="myapp.vxml#form_id"/>
```

```
<goto next="../special_lunch#wants_vegan"/>
```

Пример передачи управления к CGI-сценарию, который генерирует VoiceXML-документ с использованием:

❑ метода GET `<goto next="http://machine.company.com/cgi-bin/nba.pl?team=lakers&city=nyc"/>`;

❑ метода POST `<goto next="http://machine.company.com/cgi-bin/nba.pl" method="post" submit="team=lakers&city=nyc"/>`.

Замечание

При использовании метода POST нужно использовать submit.

Тег **<grammar>**

Определяет разрешенный словарь при взаимодействии с пользователем.

Синтаксис

```
<grammar name="string"
```

```
src="URL"/>
```

```
<grammar>
```

```
<![CDATA[
```

```
grammar Описание or name of an
```

```
intrinsic grammar
```

```
]]>
```

```
</grammar>
```

где

❑ src — URL файла внешней грамматики (не обязательно).

❑ name — имя грамматики (не обязательно).

Элемент `<grammar>` позволяет определить корректное высказывание, которое может произносить абонент. Грамматика является одним из ключевых строительных блоков телефонного приложения. Телефонное приложение представляет абоненту список возможностей и слушает его ответ. При этом каждое высказывание сравнивается с текущей активной грамматикой.

Когда распознавание становится успешным, соответствующее значение, которое задается директивой `option`, возвращается платформе. Платформа использует это значение для определения следующей последовательности событий приложения.

Типы грамматик

- `inline` — грамматика является списком фраз, а подграмматики включены внутрь файла VoiceXML. Разрешено определять любое количество `inline`-грамматик в приложении с помощью элемента `<grammar>` и тега `CDATA` :

```
<grammar>
  <![CDATA[
    GrammarDescription
  ]]>
</grammar>
```

- `intrinsic`-грамматика — это грамматика, встроенная в Speech Recognition Engine. Чтобы получить доступ к этой грамматике, нужно использовать тот же самый синтаксис, что при `inline`-грамматике, заменяя `GrammarDescription` именем `intrinsic`-грамматики:

```
<grammar>
  <![CDATA[
    GrammarName
  ]]>
</grammar>
```

- `external` — грамматика является текстовым файлом, который существует в качестве отдельного документа. Внутри этого файла вы определяете одну или более фраз и подграмматик. Для того чтобы использовать этот файл, нужно просто сохранить грамматику на Web-сервере и сослаться на данный файл в VoiceXML-приложении:

```
<grammar src="URL_of_grammar_file/GrammarFileName.txt"/>
```

Область действия грамматик

На грамматики можно ссылаться внутри элементов `<vxml>`, `<link>`, `<form>`, `<field>`. Эти элементы определяют область действия грамматики.

- `application` — грамматика, определенная и включенная в корневой документ приложения как порожденный элемент тегов `<vxml>` или `<link>`, имеет область действия всего приложения, что означает, что она активна в любом документе, на который есть ссылка из атрибута `application` элемента `<vxml>`. Рекомендуется использовать универсальный файл Tellme в качестве корневого документа приложения. Этот файл определяет грамматику для ввода универсального пользователя, которая применяется к пользователям услуг Tellme.

- ❑ **document** — грамматика, определенная внутри элемента `<vxml>` или `<link>` (внутри `<vxml>` данного документа), имеет область действия документа, то есть является активной в любой точке документа. Если документ обслуживается в качестве корневого документа приложения, грамматика становится активной в любом загружаемом документе приложения.
- ❑ **form** — грамматика, определенная в элементе `<form>`, имеет область действия формы, то есть активна только в пределах формы, где она определена. Внутри формы все грамматики уровня приложения и уровня документа также остаются активными.
- ❑ **field** — грамматика, заданная в элементе `<field>`, имеет область действия поля, то есть активна только в поле, в котором она задана. В этом поле активны также все грамматики более высокого уровня.

Вызывающие теги: `<field>`, `<form>`, `<link>`, `<transfer>`, `<vxml>`.

Порожденные теги: нет.

Пример

Пример показывает, как писать внутреннюю грамматику, используя элемент `<grammar>` и тег `CDATA`. Так как грамматика определена внутри элемента `<vxml>`, эти высказывания имеют область действия документа:

- ❑ варианты "dogs", "dog" и "canine", любое из которых возвращает значение dogs.
- ❑ варианты "plants", "plant" и "nursery", любое из которых возвращает значение plants.

```
<vxml>
<grammar>
  <![CDATA[
    [
      [dtmf-1 dogs dog canine] {<option "dogs">}
      [dtmf-2 plants plant nursery] {<option "plants">}
    ]
  ]]>
</grammar>
<form>
  <field name="myapp_input">
    <prompt><audio>prompt 1</audio></prompt>
    <nomatch><audio>first nomatch</audio><reprompt/></nomatch>
    <noinput><audio>first noinput</audio><reprompt/></noinput>
    <filled>
      <audio> you said {myapp_input} </audio>
      <goto next="_home"/>
```

```

    </filled>
  </field>
</form>
</vxml>

```

Этот пример показывает, как ссылаться на внутреннюю грамматику YES_NO и внешнюю грамматику с именем dogbreed.txt. Intrinsic-грамматика определяет высказывания, которые пользователь может сказать в любом месте документа. Внешняя грамматика dogbreed.txt задает высказывание, которое абонент говорит при ответе на подсказку поля.

```

<vxml>
  <grammar>
    <![CDATA[ YES_NO]]>
  </grammar>
  <form id="question_1">
    <field name="answer1">
      <prompt>
        <audio>do you know the breed of the dog your are looking for?
      </audio>
    </prompt>
    ...
    <filled>
      <result name="no"><goto next="#breedoptions"/></result>
      <result name="yes"><goto next="#statebreed"/></result>
    </filled>
    </field>
  </form>
  ...
  <form id="statebreed">
    <field name="store_breed">
      <grammar src="http://web_server/app_grammars/dogbreed.txt"/>
      <prompt>
        <audio>say the name of the breed you are looking for</audio>
      </prompt>
      ...
      <filled>
        <result name="chow_chow">
          <goto next="http://web_server/dog_app/husky.vxml"/>
        </result>
      </filled>
    </field>
    ...
  </form>

```

```
</form>
</vxml>
<vxml>
  <form id="question_1">
    <field name="answer1">
      <grammar>
        <![CDATA[ YES_NO ]]>
      </grammar>
      <prompt>
        <audio>Do you know the breed of the dog you are looking for?</audio>
      </prompt>
      <nomatch>
        <audio>Sorry, I didn't get that.</audio>
        <reprompt/>
      </nomatch>
      <noinput>
        <audio>Sorry, I didn't hear you.</audio>
        <reprompt/>
      </noinput>
      <filled>
        <result name="no"><goto next="#breedoptions"/></result>
        <result name="yes"><goto next="#statebreed"/></result>
      </filled>
    </field>
  </form>
  <form id="breedoptions">
    <block>
      <audio>Here are your options.</audio>
      <pause> 500 </pause>
      <audio>Poodle, german sheperd, beagle, golden retriever,
        chow chow.
      </audio>
      <goto next="#statebreed"/>
    </block>
  </form>
  <form id="statebreed">
    <field name="store_breed">
      <grammar src="http://my_server/dog_application/dogbreed.txt"/>
      <prompt>
        <audio>say the name of the breed you are looking for</audio>
      </prompt>
```

```

<nomatch>
  <audio>Sorry, I didn't get that.</audio>
  <reprompt/>
</nomatch>
<noinput>
  <audio>Sorry, I didn't hear you.</audio>
  <reprompt/>
</noinput>
<filled>
  <result name="chow_chow">
    <audio>You chose chow chow.</audio>
  </result>
  <audio>Thank you for stating your breed.</audio>
</filled>
</field>
</form>
</vxml>

```

Файл dogbreed.txt:

```

DOGBREED:str {<option $str>}
DOGBREED [
    poodle {return("poodle")}
    [chow (chow chow)] {return("chow_chow")}
    (german sheperd) {return("german_sheperd")}
    beagle {return("beagle")} (golden retriever)
    {return("golden_r")}
]

```

Тег *<help>*

Обрабатывает случай, когда абонент говорит "help".

Синтаксис

```

<help>
  child elements
</help>

```

Замечание

<help> является коротким написанием выражения *<catch event="help">*.

Обработчик событий `<help>` ловит набор переменных `<field>`, установленных на "help", которые возникают, когда активная грамматика распознает высказывание "help" и возвращает `{<option "help">}`.

Если вы хотите, чтобы обработка включалась каждый раз, когда абонент говорит "help", нужно включить грамматику в соответствующую область действия.

Вызывающие теги: `<field>`, `<menu>`, `<transfer>`, `<form>`, `<subdialog>`, `<vxml>`.

Порожденные теги: `<assign>`, `<goto>`, `<prompt>`, `<throw>`, `<audio>`, `<if><else></elseif>`, `<reprompt>`, `<var>`, `<clear>`, `<listen>`, `<return>`, `<disconnect>`, `<log>`, `<script>`, `<foreach>`, `<pause>`, `<submit>`.

Пример

Пример подсказывает, что абоненту следует что-нибудь сказать. Если абонент говорит "help", проигрывается сообщение, описанное в элементе `<audio>` события `<help>`.

```
<vxml>
<!-- "help" grammar with document scope -->
<grammar>
  <![CDATA[[
    [help (help_me) (assist me please)] {<option "help">} ]
  ]]>
</grammar>
<form id="help_example">
  <grammar src="http://web_server/app_grammars/grammar.txt"/>
  <field name="field_var">
    <prompt>
      <audio>Say something</audio>
    </prompt>
    <!-- begin event handlers -->
    <!-- user provides no input -->
    <noinput>
      <audio>You didn't say anything</audio><reprompt/>
    </noinput>
    <!-- user says something that matches the help grammar -->
    <help>
      <audio>You asked for help...</audio><reprompt/>
    </help>
    <!-- don't care if user says something in an active grammar -->
    <default>
      <audio>Thanks for saying something</audio><reprompt/>
```

```

    </default>
  </field>
</form>
</vxml>

```

Тег **<if><elseif/><else/>**

Выполняет условную логику.

Синтаксис

```

<if cond="JS_expression">
  child elements
  <elseif cond="JS_expression"/> child elements
  <else/>
  child elements
</if>

```

где

cond — условие, которое оценивается (обязательно).

Директива **<if>** требуется для организации условной логики в программе. Она используется вместе с необязательными директивами **<elseif>** и **<else>**.

Вызывающие теги: **<block>**, **<filled>**, **<if><else/><elseif/>**, **<prompt>**, **<catch>**, **<foreach>**, **<noinput>**, **<result>**, **<default>**, **<help>**, **<nomatch>**.

Порожденные теги: **<assign>**, **<goto>**, **<prompt>**, **<throw>**, **<audio>**, **<if><else/><elseif/>**, **<reprompt>**, **<var>**, **<clear>**, **<listen>**, **<return>**, **<disconnect>**, **<log>**, **<script>**, **<foreach>**, **<pause>**, **<submit>**.

Замечание

Элементы **<else/>** и **<elseif/>** не имеют порожденных тегов; они просто являются маркерами внутри списка элементов.

Пример

```

<vxml>
  <form>
    <field name="trivia">
      <grammar>
        <![CDATA[
          [
            [one dtmf-1] {<option "1">}
            [two dtmf-2] {<option "2">}
            [(tell me)] {<option "tellme">}
          ]
        ]>

```

```
}  
]]>  
</grammar>  
<prompt>  
  <audio>Which of these is a national holiday?</audio>  
  <audio>One. The fourth of July.</audio>  
  <audio>Two. December 13.</audio>  
  <audio>Or say Tell me.</audio>  
</prompt>  
<noinput>  
  <audio>Sorry, I did not hear you.</audio>  
  <reprompt/>  
</noinput>  
<nomatch>  
  <audio>Sorry, please repeat your answer.</audio>
```

Тег **<link>**

Задаёт одну или более грамматик и ссылок для перехода, или событие, которое возникает, когда распознаётся высказывание.

Синтаксис

```
<link next="URL"  
      event="event"  
      expr="JS_expression">  
  <grammar>...<grammar/>  
</link>
```

где

- ❑ **next** — URL документа для перехода.
- ❑ **event** — событие, возникающее при успешном распознавании. Возможно задание только одного события.
- ❑ **expr** — оцениваемое выражение JavaScript. Результат этой оценки используется в качестве URL-документа, к которому выполняется переход в случае, если высказывание абонента соответствует грамматике элемента **<link>** (обязателен, если атрибуты **next** или **event** не указаны).

Элемент **<link>** позволяет определить inline-грамматику области действия документа. Когда высказывание соответствует этой грамматике, приложение переходит к URL, описанном в атрибуте **next**.

Примечание

Грамматика `<link>` использует директиву `option` для описания возвращаемого значения. В случае успешного распознавания платформа автоматически переходит к URL, описанному в атрибуте `next`.

Если элемент `<link>` определяется в корневом документе приложения, его грамматики являются активными в любом загруженном документе, входящем в это приложение. Рекомендуется использовать в качестве такого документа универсальный файл платформы Tellme. Этот файл использует элемент `<link>` для определения глобальной грамматики для TELLME MENU и GOODBYE.

Вызывающие теги: `<vxml>`.

Порожденные теги: `<grammar>`.

Пример

В этом примере `<link>` переводит приложение к форме с именем "they_said_it" в новом документе в случае, когда абонент сказал "eggplant" или "my favorite vegetable" в любое время при выполнении текущего документа. Заметим, что эта грамматика не описывает возвращаемое значение, так как платформа переходит к URL, заданному атрибутом `next`, когда абонент произносит верное высказывание.

Файл 1:

```
<vxml>
<link next="link2.vxml#they_said_it">
<grammar>
  <![CDATA[
    [eggplant (my favorite vegetable)]
  ]]>
</grammar>
</link>
<form id="vegetable">
  <field name="vegetable">
    <prompt>
      <audio>Say something</audio>
    </prompt>
    <noinput>
      <reprompt/>
    </noinput>
    <default>
      <audio>you said something</audio>
      <reprompt/>
    </default>
```

```
</field>
</form>
</vxml>
```

Файл 2:

```
<vxml>
  <form id="they_said_it">
    <block>
      <audio>You are inside the next form. Thank you.</audio>
      <goto next="_home"/>
    </block>
  </form>
</vxml>
```

Тег **<listen>**

Заставляет платформу слушать ввод абонента.

Синтаксис

```
<listen/>
```

Директива `<listen>` заставляет платформу выполнить паузу и ждать ввод от абонента. Вам не нужно описывать триггер прослушивания при использовании элемента `<prompt>`. Для того чтобы повторить `<prompt>` и переключить любую логику внутри этого элемента, нужно использовать директиву `<reprompt/>`. Эта директива `<reprompt/>` также итерирует по списку `<prompts>` до тех пор пока не найдет последнюю.

Элемент `<listen>` следует использовать в конце обработчика событий, таких как `<nomatch>`, если нужно слушать следующий ввод без повторной подсказки абоненту.

Будущие версии платформы Tellme не будут требовать использования этой директивы.

Вызывающие теги: `<catch>`, `<help>`, `<prompt>`, `<default>`, `<noinput>`, `<result>`, `<filled>`, `<nomatch>`.

Порожденные теги: нет.

Пример

В примере директива `<listen>` дает абоненту секундную возможность ответа, когда платформа порождает событие `noinput` после первой подсказки `<prompt>`.

```
<vxml>
  <form>
    <field name="response">
      <grammar>YES_NO</grammar>
```

```

<!-- reprompt iterates through these prompts until it hits the
last prompt, which is replayed thereafter -->
<prompt>
  <audio>say something</audio>
</prompt>
<prompt>
  <audio>say something number 2</audio>
</prompt>
<prompt>
  <audio>say something number 3</audio>
</prompt>
<!-- event handlers -->
<noinput>
  <audio>please say YES or NO</audio>
  <listen/>
</noinput>
<default>
  <audio>got to the default</audio>
  <reprompt/>
</default>
<filled>
  <audio>you said {response}</audio>
  <reprompt />
</filled>
</field>
</form>
</vxml>

```

Тег **<log>**

Пишет вывод в Tellme Studio Debug Log.

Синтаксис

```
<log> text to write to the Debug Log </log>
```

Элемент **<log>** позволяет писать вывод, включая переменные, из контейнера исполняемого содержания (**<block>**, **<initial>** и обработчик событий) в лог-файл Tellme Studio Debug Log.

Замечание

Для того чтобы написать что-то в Debug Log изнутри элемента **<script>** лучше использовать `vxmlllog(string)`.

Вызывающие теги: элемент `<log>` можно использовать внутри любого порожденного элемента `<vxml>` за исключением `<form>` и `<script>`.

Порожденные теги: нет.

Пример

Нижеприведенный код порождает следующий вывод в Debug Log:

```
[06/04/2000:17:22:37 DST] DEV : 000190 JAVASCRIPT ***** 409
[06/04/2000:17:22:37 DST] DEV : 000190 JAVASCRIPT ***** 252.1313
[06/04/2000:17:22:37 DST] DEV : 000190 JAVASCRIPT /252/
[06/04/2000:17:22:37 DST] DEV : 000190 JAVASCRIPT -1
```

```
<vxml>
<block>
  <assign name="document.myapp.number" expr="'252.1313'"/>
  <assign name="document.myapp.badareacodes" expr="'409'"/>
</block>
<block>
  <script type="text/javascript">
    <![CDATA[
      var number = vxmldata["document.myapp.number"];
      var badareacodes = vxmldata["document.myapp.badareacodes"];
      vxmlllog("***** " + badareacodes);
      vxmlllog("***** " + number);
      regexp = new RegExp(number.substr(0,3));
      vxmlllog(regexp.toString());
      vxmlllog(badareacodes.search(regexp));
    ]]>
  </script>
  <goto next="_home"/>
</block>
</vxml>
```

Тег *<menu>*

Представляет список выборов для абонента и переходов к выбранной информации.

Синтаксис

```
<menu id="string"
      tmf="true_false">
  child elements
</menu>
```

где

- ❑ `id` — имя меню, которое может обслуживать как адрес для перехода (обязательное).
- ❑ `dtmf` — установка для выбора, может или не может абонент использовать цифры от 1 до 9 для элементов `<choice>` внутри элемента `<menu>`. По умолчанию `false` (необязательно).

Замечание

Если установить `<menu dtmf="true">` и дополнительно описать число для элемента `<choice>` (атрибут `dtmf`), число элемента `<choice>` будет иметь приоритет над числом, заданным в `<menu>`.

Например, если вы имеете пять элементов `<choice>`, А, В, С, D и Е, назначили `<choice dtmf="1">` для С, а `<menu dtmf="true">`, то абонент всегда переходит к С при нажатии `<1>`. В этом случае абонент не может попасть к А, нажав `<1>`, а число "3" не соответствует ни одной из возможностей.

Элемент `<menu>` является удобной короткой версией `<form>`, которая подсказывает абоненту варианты выбора и затем выполняет переход, основываясь на этом выборе. Элемент `<menu>` использует `<prompt>` для представления списка выбора. Каждая опция в списке отражается в элемент `<choice>`, как речь или фрагмент DTMF-грамматики. Если ввод абонента соответствует фрагменту грамматики, платформа переходит к позиции, описанной атрибутом `next`, в этом элементе `<choice>`.

Все общие обработчики событий — `<nomatch>`, `<noinput>`, `<default>` и `<help>` — могут использоваться в `<menu>` совершенно так же, как они используются в элементе `<field>`.

Тем не менее, обработчик события `<filled>` не имеет смысла в `<menu>`, так как элемент `<choice>` автоматически действует, как этот обработчик для активного фрагмента грамматики.

Вызывающие теги: `<vxml>`.

Порожденные теги: `<choice>`, `<help>`, `<nomatch>`, `<script>`, `<default>`, `<noinput>`, `<prompt>`, `<var>`.

Пример

Меню предлагает три выбора. Фрагмент грамматики, описанный для каждого элемента `<choice>` — это один элемент из списка `<prompt>`. Заметим, что пользователь должен сказать "ESPN sports", а не просто "ESPN" или "sports" для того, чтобы достичь правильного распознавания первого пункта меню. Абонент может сказать "Caltech news" или "news" для того, чтобы соответствовать третьему пункту меню.

```
<vxml>
```

```
<menu id="three_choice_menu">
```

```

<prompt>
  <audio>Welcome. Say E S P N sports, weather, or Caltech news.</audio>
</prompt>
<choice next="#sports_report">
  (e s p n sports)
</choice>
<choice next="#weather_report">
  weather
</choice>
<choice next="#news_report">
  (?caltech news)
</choice>
<default><reprompt/></default>
</menu>
<form id = "sports_report">
  <block>
    <audio>The New York Giants will win the Superbowl next year!</audio>
    <goto next= "_home"/>
  </block>
</form>
<form id = "weather_report">
  <block>
    <audio>It will be mostly clear and sunny today</audio>
    <goto next= "_home"/>
  </block>
</form>
<form id = "news_report">
  <block>
    <audio>Earthquake in California causes damages in the billions</audio>
    <goto next= "_home"/>
  </block>
</form>
</vxml>

```

Тег **<meta>**

Описывает общую информацию о документе VoiceXML.

Синтаксис

```

<meta name="string"
      content="string"/>

```

где

- ☐ name — метка, указывающая тип метаданных content. Можно задать любой тип (обязательно).
- ☐ content — значение для name (обязательно).

Замечание

Для того чтобы поддержать действие переменной в старой области, нужно использовать: `<meta name="scoping" content="old"/>`.

Для того чтобы поддержать действие переменной в новой области, нужно писать: `<meta name="scoping" content="new"/>`.

Информация, описанная в элементе `<meta>`, программно доступна как переменная типа "только для чтения" с именем `named session.meta.name`.

Вызывающие теги: `<vxml>`.

Порожденные теги: нет.

Пример

В примере элементы `<meta>` установлены и используются в дальнейшем внутри приложения.

```
<vxml>
<meta name="author" content="John Tellme"/>
<meta name="version" content="2.5.1"/>
<form>
  <block>
    <audio>This application was written by
      <value name="session.meta.author"/>
      version <value name="session.meta.version"/>
    </audio>
    <goto next="_home"/>
  </block>
</form>
</vxml>
```

Тег `<noinput>`

Обрабатывает ситуации, когда пользователь ничего не говорит.

Синтаксис

```
<noinput>
  count="integer"
  child elements
</noinput>
```

где `count` — количество раз, когда это событие должно порождаться в текущей форме прежде чем начнет использоваться обработчик событий. По умолчанию — счетчик предыдущего элемента `<catch>` плюс один (необязательный).

Замечание

`<noinput>` является короткой формой для `<catch event="noinput">`.

Обработчик событий `<noinput>` подхватывает событие `noinput`. Платформа порождает событие `noinput`, если не слышит произнесенной речи или не получает DTMF-последовательности после элемента `<prompt>`.

Разрешено использование нескольких сущностей `<noinput>` в поле. При каждой итерации по полю платформа использует следующую сущность обработчика событий, если событие возникает снова. После итерации по всем обработчикам, платформа заново проигрывает последний. При желании изменений естественный процесс обработки нужно использовать атрибут `count` для изменения ответа на событие `noinput`, которое возникает много раз. Если элемент `<noinput>` имеет счетчик 1, платформа использует обработчик, который обрабатывал событие первый раз.

Если `<noinput>` имеет счетчик 2, платформа использует его, `noinput` событие порождается второй раз. Вообще, платформа находит самый высокий доступный счетчик, который меньше чем или равен числу возникновения события `noinput` в текущей форме.

Вызывающие теги: `<field>`, `<menu>`, `<transfer>`, `<form>`, `<subdialog>`, `<vxml>`.

Порожденные теги: `<assign>`, `<goto>`, `<prompt>`, `<throw>`, `<audio>`, `<if>``<else/>``<elseif/>`, `<reprompt>`, `<var>`, `<clear>`, `<listen>`, `<return>`, `<disconnect>`, `<log>`, `<script>`, `<foreach>`, `<pause>`, `<submit>`.

Пример

```
<vxml>
<form id="MyForm">
  <field name="storename">
    <grammar>
      <![CDATA[
        [
          [dtmf-1 dogs dog canine (pet ?store) (mutt)] {<option "dogs">}
          [dtmf-2 plants plant nursery (hardware ?store) flower flowers]
          {<option "plants">}}
        ]
      ]]>
    </grammar>
    <prompt>
      <audio>Say dogs, plants, or help</audio>
```

```
</prompt>
<nomatch>
  <audio>Sorry, I didn't get that</audio>
  <reprompt/>
</nomatch>
<noinput>
  <audio>Sorry, I didn't hear you</audio>
  <reprompt/>
</noinput>
<help>
  <audio>You are in Buy Me. Please choose a type of store.</audio>
  <reprompt/>
</help>
<default/>
</field>
</form>
</wml>
```

Тег **<nomatch>**

Обрабатывает ввод абонента, который не распознается как часть активной грамматики.

Синтаксис

```
<nomatch>
count="integer"
child elements
</nomatch>
```

где count — количество раз, когда это событие должно породиться прежде чем будет использован обработчик. По умолчанию — счетчик предыдущего элемента <catch> элемента плюс 1 (необязательный).

Замечание

<nomatch> является короткой формой от <catch event="nomatch">.

Обработчик <nomatch> отлавливает событие nomatch. Платформа порождает событие nomatch каждый раз, когда высказывание абонента не соответствует активной грамматике.

Разрешено использование нескольких сущностей <nomatch> в поле. При каждой итерации по полю платформа использует следующую сущность обработчика событий, если событие возникает снова. После итерации по всем обработчикам, платформа заново проигрывает последний. При желании из-

мений естественный процесс обработки нужно использовать атрибут `count` для изменения ответа на событие `nomatch`, которое возникает много раз. Если элемент `<nomatch>` имеет счетчик 1, платформа использует обработчик, который обрабатывал событие первый раз.

Если `<nomatch>` имеет счетчик 2, платформа использует его, `noinput` событие порождается второй раз. Вообще, платформа находит самый высокий доступный счетчик, который меньше чем или равен числу возникновения события `noinput` в текущей форме.

Вызывающие теги: `<field>`, `<menu>`, `<transfer>`, `<form>`, `<subdialog>`, `<vxml>`.

Порожденные теги: `<assign>`, `<goto>`, `<prompt>`, `<throw>`, `<audio>`, `<if><else/><elseif/>`, `<reprompt>`, `<var>`, `<clear>`, `<listen>`, `<return>`, `<disconnect>`, `<log>`, `<script>`, `<foreach>`, `<pause>`, `<submit>`.

Пример

Пример показывает, как использовать событие `<nomatch>` в сочетании с другими обработчиками событий.

```
<vxml>
<form>
  <field name="storename">
    <grammar>
      <![CDATA[
        [
          [dtmf-1 dogs dog canine (pet ?store) (mutt)] {<option "dogs">}
          [dtmf-2 plants plant nursery (hardware ?store) flower flowers]
          {<option "plants">}}
        ]
      ]]>
    </grammar>
    <prompt>
      <audio>Say dogs, plants, or help</audio>
    </prompt>
    <nomatch>
      <audio>Sorry, I didn't get that</audio>
      <reprompt/>
    </nomatch>
    <noinput>
      <audio>Sorry, I didn't get that</audio>
      <reprompt/>
    </noinput>
```

```
<help>
  <audio>You are in Buy Me. Please choose a type of store.</audio>
  <reprompt/>
</help>
<default/>
</field>
</form>
</vxml>
```

Тег **<param>**

Задаёт значение, которое должно быть передано элементу `<subdialog>`.

Примечание

Необходимо использовать `new scoping` поведение для работы элемента `<param>`.

Синтаксис

```
<param name="string"
      expr="JS_expression"
      value="string"/>
```

где

- ☐ `name` — имя переменной, которая должна быть инициирована внутри элемента `<subdialog>` (обязательный).
- ☐ `expr` — выражение JavaScript, которое назначается для `name` (обязательно, если `value` не специфицировано).
- ☐ `value` — строка символов, которая назначается для `name` (требуется, если `expr` не задано).

Элемент `<param>` позволяет инициировать значение, используемое в элементе `<subdialog>` без использования сценария, выполняемого на стороне сервера.

Вызывающие теги: `<subdialog>`.

Порожденные теги: нет.

Пример

Пример использует дату заказа для элемента поиска в специальных предложениях. Элемент `<param>` иницирует дату заказа.

```
<vxml>
  <form>
```

```

<subdialog name="confirm" src="#itemPROMO">
  <param name="orderDATE" expr="'01012000'"/>
  <filled>
    <submit next="http://department/itemList"/>
  </filled>
</subdialog>
</form>
<!-- subdialog -->
<form id="itemPROMO">
  <var name="orderDATE"/>
  <field name="itemSpecial">
    <grammar src="http://myserver/grammars/date.txt"/>
    <prompt> Using your keypdad, enter the date of your order...</prompt>
    <filled>
      <if cond="itemHistory(itemSpecial,orderDATE)">
        <var name="promo" expr="true"/>
      <else/>
        <var name="promo" expr="false"/>
      </if>
      <return namelist="itemSpecial promo"/>
    </filled>
  </field>
</form>
</vxml>

```

Тег *<pause>*

Вводит заданное количество времени неактивности, прежде чем продолжить.

Синтаксис

```
<pause> number (n) of milliseconds </pause>
```

Элемент *<pause>* вводит паузу в миллисекундах. Полезно использовать между отдельными частями звуков так, чтобы они слышались более естественно.

Вызывающие теги: *<block>*, *<filled>*, *<if>**<else/>**<elseif/>*, *<prompt>*, *<catch>*, *<foreach>*, *<noinput>*, *<result>*, *<default>*, *<help>*, *<nomatch>*.

Порожденные теги: нет.

Пример

В примере абонент слышит, "This is prompt one...[1000 msec пауза]....this is prompt two".

```
<vxml>
  <form>
    <block>
      <audio> First sentence </audio>
      <pause>1000</pause>
      <audio> Second sentence </audio>
      <goto next = "_home"/>
    </block>
  </form>
</vxml>
```

Тег **<prompt>**

Проигрывает звук абоненту и слушает ответ.

Синтаксис

```
<prompt item="variable"
      in="comma_delimited_list">
  child elements
</prompt>
```

где

- ❑ `item` — переменная, в которой временно хранится каждый элемент списка (описанного атрибутом `in`) при итерации по нему (необязательно).
- ❑ `in` — разделенный запятыми список элементов, по котоым выполняется итерация. Этот список может быть переменным и вперемежку со строками и другими переменными. Если переменная строится с помощью метода `vxmldata.append()` method, все элементы в этой переменной возвращаются (обязательный, если задан `item`).

Замечание

Платформа Tellme в настоящее время интерпретирует текст прямо внутри элемента `<prompt>` как TTS-текст. Не требуется заключать TTS-текст в теги аудио. Тем не менее, рекомендуется продолжать использовать эти теги для обратной совместимости с предыдущими версиями приложений.

Обработчик событий `<prompt>` подсказывает абоненту, как выполнить ввод с помощью одного или более элементов `<audio>` и затем выполняет директиву `<listen>`, которая указывает платформе на необходимость перейти к прослушиванию сообщений от абонента. Если платформа не слышит ответа, она порождает событие `noinput`. Если она слышит высказывание, которое не соответствует активной грамматике, она порождает событие `nomatch`. Если она слышит высказывание, которое соответствует части активной грамматики, она порождает событие `filled`.

Замечание

При включении элемента `<foreach>` внутрь одиночного элемента `<prompt>` результатом будет один большой элемент `<audio>` — будет слышно весь список каждый раз, когда включается элемент `<prompt>`.

Вызывающие теги: `<field>`, `<menu>`, `<transfer>`, `<form>`, `<subdialog>`, `<vxml>`.

Порожденные теги: `<assign>`, `<foreach>`, `<log>`, `<submit>`, `<audio>`, `<goto>`, `<pause>`, `<throw>`, `<clear>`, `<if><else/><elseif/>`, `<reprompt>`, `<value>`, `<disconnect>`, `<listen>`, `<script>` `<var>`.

Пример

В этом примере абонент слышит "apple...got to the default...pear...got to the default...orange...got to the default...orange...got to the default...orange... etc." Последний элемент заново проигрывается в последующих итерациях повторного ввода в поле, с которого бы начинался первый элемент. Если установить атрибут `order` тега `<reprompt>` на "previous" или "current", первая подсказка проигрывается снова и снова. Заметим, что по умолчанию порядок задается как "next".

```
<vxml>
<form id="form2">
  <field name="fruits">
    <prompt count="1">
      <audio>apple</audio>
    </prompt>
    <prompt count="2">
      <audio>pear</audio>
    </prompt>
    <prompt count="3">
      <audio>orange</audio>
    </prompt>
    <default>
      <audio>got to the default</audio>
      <reprompt/>
    </default>
  </field>
</form>
</vxml>
```

Тег `<property>`

Описывает параметры речи, которые принимаются по умолчанию ко всему приложению, документу или форме.

Синтаксис

```
<property name="string" value="integer"/>
```

где

name — имя свойства, которое устанавливается (обязательно).

Свойства, которые вы будете устанавливать, включают следующее.

- ❑ `confidencelevel` — уровень доверия для поля [0...100]. Если область действия доверия для высказывания находится ниже `rejectthreshold`, высказывание отбрасывается (необязательное). Заметим, что этот атрибут описывает одно значение, с которым сравнивается высказывание, а это означает, что высказывание либо отбрасывается, либо принимается. Атрибуты `rejectthresh` и `acceptthresh` описывают диапазон, в который может попасть доверительный уровень высказывания и работает в сочетании с атрибутом `confirm`. Если результат попадает в этот диапазон, то управление передается элементу `<confirm>` (необязательный).
- ❑ `interdigittimeout` — интервал времени, который платформа ожидает между вводом DTMF-последовательности, прежде чем будет вызван механизм ее распознавания (необязательный).
- ❑ `bargein` — установка управления возможностями абонента прерывать действия системы (по умолчанию `"true"`). Когда установлено в `"false"`, `bargein` отключается и абонент должен слушать все подсказки до конца, пока ему не разрешат что-нибудь сказать. Установка в `"false"` может вызвать неприятие пользователя, но может также быть использована эффективно для коротких подсказок, когда есть потребность того, чтобы их абонент наверняка услышал. Например, "Я получаю много прерываний. Нажмите единицу, если хотите переключиться в режим DTMF" (необязательный).

Замечание

DTMF-input превалирует над `bargein=false`.

- ❑ `timeout` — интервал времени, в течение которого платформа ожидает ввод от пользователя, прежде чем породить событие `noinput`. По умолчанию размер интервала устанавливается платформой (необязательный).
- ❑ `tellme.timeoutondtmf` — установка управления на возможность ожидания в процессе ввода DTMF-последовательности, прежде чем начать действия по распознаванию. В случае, когда параметр `"true"` установлен по умолчанию: если тайм-аут между соседними вводами цифр в коде DTMF истекает до наступления корректной интерпретации кода, диалог пользователя переводят в следующее состояние. Когда установлено

в "false", любая корректная интерпретация DTMF немедленно отсылает абонента к следующему состоянию. Например, если `timeoutondtmf` установлено в "false" и определен ввод с клавиатуры "0" для того, чтобы перевести абонента в главное меню, а абонент пытается ввести "08873" в качестве zip-кода, он немедленно перенаправляется в главное меню. Если установлено "true", абонент ожидает несколько секунд, прежде чем будет переведен в главное меню при нажатии на "0" и этого времени хватит, чтобы нажать следующую цифру "8", если нужно перейти в следующую позицию диалога (необязательный).

- ❑ `tellme.bargeinlevel` — уровень громкости, на котором абонент должен произнести слово для того, чтобы переключить `bargein`. Рекомендуется сказать платформе Tellme прежде, чем модифицировать этот чувствительный параметр (необязательный).
- ❑ `tellme.endseconds` — период времени, в течение которого распознаватель ожидает внутри речи абонента, прежде чем предположит, что он закончил свою речь (по умолчанию 1 секунда). Параметр очень чувствительный. Если значение слишком низко, то система отсекает абонента, прежде чем он завершит высказывание, что обычно приводит к появлению события `nomatch`, пока распознаватель не получил достаточной информации. Если же этот уровень находится слишком высоко, то система выглядит слишком медленной (необязательный).

Замечание

Атрибут `endseconds` предназначен только для речи. Ввод DTMF регулируется параметром `timeoutondtmf`.

- ❑ `tellme.magicword` — установка для переключения режима `magic word` на "on" или "off". "Magic" является способностью системы продолжать проигрывание подсказки в процессе одновременного сравнения высказывания абонента на соответствие активной грамматике. Когда `magicword` установлен на "true" (по умолчанию "false"), `bargein` работает тихо; начало речи не прерывает текущую подсказку и, если высказывание не распознается, система проигрывает скорее подсказку, чем отвечает на ситуацию `nomatch`. Когда фраза распознается, система определяет, какое состояние должно быть следующим, останавливает проигрывание подсказки и затем представляет абоненту новую информацию. Режим "Magic word recognition" является очень надежным средством против шума, но одновременно влияет на скорость работы системы. Заметим, что скорость распознавания и время ответа прямо связаны со сложностью используемой грамматики (необязательный).

Замечание

Любой ввод DTMF является более приоритетным, чем режим `magic word`. Это означает, что когда ввод DTMF не соответствует активной грамматике возника-

ет событие `nomatch`. Грамматики уровня приложения являются активными в процессе всего режима `magic word`.

- ❑ `tellme.pruning` — диапазон поиска [400...5000] Viterbi по акустической модели, который является числом путей для рассмотрения в каждой итерации. Использование этого параметра позволяет выполнить настройку между точностью и скоростью. 5000 — наиболее точный, а 400 наиболее быстрый. Заметим, что платформа Tellme использует установку, принятую по умолчанию для большинства сценариев (необязательный).
- ❑ `tellme.phoneticpruning` — установка `phonetic pruning`. Когда установлено в `"true"` (по умолчанию `"false"`), `phoneticpruning` заставляет сервер распознавания выполнять дополнительные вычисления для исключения невероятных гипотез. При использовании этой возможности для больших грамматик, распознавание может занимать гораздо меньше времени при той же самой точности распознавания. Однако для малых грамматик, производительность по-прежнему сравнима.
- ❑ `value` — значение, назначаемое свойству.

Элемент `<property>` позволяет установить параметры распознавания для всего приложения, документа или отдельной формы. Платформа Tellme сначала будет смотреть на атрибут `<field>`, а затем на значение атрибута `<property>`. Если ни один из этих элементов не имеет значения, платформа использует их значения, заданные по умолчанию.

Вызывающие теги: `<field>`, `<form>`, `<subdialog>`, `<vxml>`.

Порожденные теги: нет.

Пример

В примере поле использует значение `timeout`, описанное на уровне документа, в то время как поле `puff` использует `timeout`, описанное для диалога.

```
<vxml>
<!-- property timeout with document scope -->
<property name="timeout" value="1.0"/>
<form id="favorite_cartoon">
  <field name="cartoon">
    <grammar>
      <![CDATA[
        [
          [yes]{<option "yes">}
          [no]{<option "no">}
        ]
      ]]>
    </grammar>
```

```

<prompt>
  <audio> do you like the power puff girls? </audio>
</prompt>
<filled>
  <audio> all right!! </audio>
  <goto next="#favorite_powerpuff"/>
</filled>
<default>
  <audio> I couldn't understand. Say yes or no. </audio>
  <pause> 750 </pause>
  <reprompt/>
</default>
</field>
</form>
<form id="favorite_powerpuff">
  <field name="puff">
    <!-- property timeout with field scope -->
    <property name="timeout" value="3.0"/>
    <grammar>
      <![CDATA[
        [
          [ buttercup ]{<option "buttercup">}
          [ bubbles ]{<option "bubbles">}
          [ blossom ]{<option "blossom">}
        ]
      ]]>
    </grammar>
    <prompt>
      <audio> who's your favorite power puff girl? </audio>
      <pause> 500 </pause>
      <audio> say bubbles, buttercup, or blossom </audio>
    </prompt>
    <noinput>
      <audio>couldn't hear you</audio>
      <pause> 750 </pause>
      <reprompt/>
    </noinput>
    <nomatch>
      <audio> what was that?</audio>

```

```
<pause>750</pause>
<reprompt/>
</nomatch>
<filled>
  <audio> you said {puff} </audio>
  <goto next="_home"/>
</filled>
</field>
</form>
</vxml>
```

Тег **<record>**

Записывает данные, вводимые абонентом.

Синтаксис

```
<record name="string"
      maxtime="seconds"
      finalsilence="seconds"
      dtmfterm="true_false">
```

child elements

```
</record>
```

где

- ❑ name — имя переменной, в которой хранятся записанные данные (обязательный).
- ❑ maxtime — максимальное время записи в секундах. По умолчанию равно 30.
- ❑ finalsilence — интервал неактивности, указывающий на конец речи. По умолчанию равно 2.

Элемент **<record>** позволяет записывать результаты ввода абонента в процессе звонка. Имеется возможность проигрывать записанный звуковой файл в той же самой форме, используя атрибут *data*. Для того чтобы сохранить звук, нужно послать звуковой файл на Web-сервер с помощью элемента **<submit>** до того момента, как абонент покинет форму.

Замечание

Данные записываются в стандарте Windows RIFF WAV. Скорость — 8 кГц, формат — 8 бит, CCITT, law. Tellme совместима с RFC 2388: <http://www.ietf.org/rfc/rfc2388.txt>.

Вызывающие теги: **<form>**.

Порожденные теги: **<catch>**, **<help>**, **<prompt>**, **<default>**, **<noinput>**, **<filled>**, **<nomatch>**.

Пример

Пример показывает, как проигрывать записанные данные немедленно, с использованием элемента `<audio>` и сохранять их на Web-сервере посредством `<submit>`.

```
<vxml>
  <var name="recordfilename"/>
  <form id="RecordTest">
    <record name="recordfilename" maxtime="60" dtmfterm="true">
      <prompt>
        <audio>What's your favorite holiday?</audio>
      </prompt>
      <filled>
        <audio> You said </audio>
        <audio data="{recordfilename}"/>
        <submit next="my_server/cgi-bin/record.cgi"
          method="post" namelist="recordfilename" />
      </filled>
    </record>
  </form>
</vxml>
```

Этот простой CGI-сценарий демонстрирует, как обращаться со звуковыми данными, переданными с помощью элемента `<submit>`. Этот сценарий только открывает и сохраняет данные в файле независимо от содержания.

record.cgi:

```
#!/usr/local/bin/perl
use strict;
#Need CGI.pm module.
#See http://stein.cshl.org/www/software/CGI/cgi\_docs.html for
```

Использование:

```
use CGI::Carp qw(fatalsToBrowser);
use CGI qw(:standard :html3);
#grab value from parameter submitted, "recordfilename".
my $value = param('recordfilename');
#Save to file named savedaudio.wav
open(OUT, ">/mydirectory/savedaudio.wav") || die "Couldn't
open the output file";
while (<$value>) {
print OUT $_;
}
```

```
close(OUT);  
print header();  
print "<vxml><form id=\"test\"><block><audio>OK, the audio  
has been saved.</audio><goto  
next=\"_home\"/></block></form></vxml>\n";
```

Тег **<reprompt>**

Заново проигрывает предварительно озвученную подсказку.

Синтаксис

```
<reprompt order="previous_current_next"/>
```

где

order — порядок, в котором проигрывается набор элементов `<prompt>` или частей списка `<prompt>` (не обязателен).

Основные опции включают:

- ❑ `previous` — при использовании в сочетании с несколькими элементами `<prompt>`, повторяется сначала в каждой итерации. Когда используется в конструкции списка `<prompt>`, элементы проигрываются последовательно в обратном порядке, проходя по всему списку до тех пор, пока не изменится управление потоком.
- ❑ `current` — при использовании нескольких элементов `<prompt>` или конструкции списка `<prompt>`, первый элемент списка проигрывается в последующих итерациях.
- ❑ `next` — при использовании в сочетании с несколькими элементами `<prompt>`, каждый из них проигрывается до тех пор, пока не будет достигнут последний элемент, который проигрывается. При использовании в конструкции списка `<prompt>`, элементы списка проигрываются последовательно до изменения управления протеканием процесса (параметр умолчания).

Замечание

Все приведенные варианты использования корректны:

```
<reprompt order="prev"/>  
<reprompt order="previous"/>  
<reprompt order="curr"/>  
<reprompt order="current"/>  
<reprompt order="next"/>
```

Элемент `<reprompt>` используется внутри обработчика события для повторного проигрывания подсказки абоненту. Если в поле имеется несколько вари-

антов подсказки, система играет следующую подсказку в списке по всем итерациям, до последней в списке, который проигрывается снова и снова.

Для задания процесса итерации по списку разрешается использовать атрибут `order`.

Вызывающие теги: `<block>`, `<filled>`, `<if><else/><elseif/>`, `<prompt>`, `<catch>`, `<foreach>`, `<noinput>`, `<result>`, `<default>`, `<help>`, `<nomatch>`.

Порожденные теги: нет.

Пример

В примере абонент слышит "apple...got to the default...pear...got to the default...orange...got to the default...orange...got to the default...orange... etc." Последний элемент проигрывается во всех последующих итерациях при повторном вводе в поле.

```
<form id="form2">
  <field name="document.myapp.fruits">
    <prompt count="1">
      <audio>apple</audio>
    </prompt>
    <prompt count="2">
      <audio>pear</audio>
    </prompt>
    <prompt count="3">
      <audio>orange</audio>
    </prompt>
    <default>
      <audio>got to the default</audio>
      <reprompt/>
    </default>
  </field>
</form>
```

Тег ***<result>***

Описывает действие, принятое для конкретного соответствия внутри `<filled>`.

Синтаксис

```
<result name="string">
  child elements
</result>
```

где `name` задает результат распознавания, соответствующий активной грамматике (обязательный).

Элемент `<result>` работает внутри элемента `<filled>` как проверка в условном операторе `if`. Он проверяет, равна ли переменная элемента `<field>` `name`, которая содержит результат распознавания, значению, описанному в ее атрибуте `name`. Если значение `<result>` `name` равно значению `<field>` `name`, директива, описанная в элементе `<result>`, будет исполняться.

Вызывающие теги: `<filled>`.

Порожденные теги: `<assign>`, `<goto>`, `<reprompt>`, `<value>`, `<audio>`, `<if>``<else/>``<elseif/>`, `<return>`, `<var>`, `<clear>`, `<listen>`, `<script>`, `<disconnect>`, `<log>`, `<submit>`, `<foreach>`, `<pause>`, `<throw>`.

Пример

```
<vxml>
  <grammar>
    <![CDATA[
      [ [vote] {<option "vote">}
      [reset] {<option "reset">}
      [results] {<option "results">}
      [exit] {<option "exit">}
    ]
  ]]>
</grammar>
<form>
  <field name="mychoice">
    <prompt>
      <audio>say vote, reset, results, or exit</audio>
    </prompt>
    <nomatch>
      <log>***no match ={mychoice}</log>
      <reprompt/>
    </nomatch>
    <filled>
      <if cond="{mychoice} == 'vote'">
        <goto next="#vote"/>
      <elseif cond="{mychoice} == 'reset'">
        <goto next="#reset"/>
      <elseif cond="{mychoice} == 'results'">
        <goto next="#results"/>
      <elseif cond="{mychoice} == 'exit'">
        <goto next="#exit"/>
      </if>
    </filled>
```

```
</field>
</form>
</vxml>
```

Тег **<return>**

Возвращает управление к последнему документу, добавленному к стеку `<subdialog>`.

Синтаксис

```
<return event="string"
        namelist="string"/>
```

где

- `event` — событие, которое должно порождаться для элемента `<subdialog>` на верху стека `subdialog` (необязательный).
- `namelist` — разделенный пробелом список переменных, который должен быть возвращен в вызывающий поддиалог. В настоящее время не нужно задавать никакой переменной с периодом в `name`. Если это требуется, лучше скопировать значение в локальную переменную, посылаемую в запросе HTTP (необязательная).

Директива `<return>` возвращает управление к последнему приложению, добавленному к стеку `subdialog`. Если атрибут `event` задан, он возвращает управление к другому обработчику событий элемента `<subdialog>`, порождая другое событие. Если атрибут `event` не задан, он возвращает управление к элементу `<filled>`, вызывая поддиалог. В этом процессе любая переменная из списка атрибута `namelist` возвращается со свойствами объекта, заданного атрибутом `name` поддиалога.

Вызывающие теги: `<block>`, `<filled>`, `<nomatch>`, `<catch>`, `<help>`, `<result>`, `<default>`, `<noinput>`.

Порожденные теги: нет.

Пример

gateway.vxml:

```
<vxml>
<form id="intro">
  <subdialog src="login.vxml#GetPin">
    <catch event="login.failed">
      <goto next="failure.vxml"/>
    </catch>
    <catch event="login.success">
      <goto next="success.vxml"/>
    </catch>
```

```
<default>
  <goto next="unexplainedevent.vxml"/>
</default>
</subdialog>
</form>
</vxml>

login.vxml:
<vxml>
  <form id="GetPin">
    <field name="pin">
      <grammar>Four_digits</grammar>
      <prompt>Please enter your pin</prompt>
      <noinput><return event="login.failed"/></noinput>
      <filled>
        <if event="validPin(pin)">
          <return event="login.success"/>
        </if>
        <return event="login.invalid"/>
      </filled>
    </field>
  </form>
</vxml>
```

Тег **<script>**

Включает в приложение блок кода JavaScript.

Синтаксис

```
<script type="text/javascript">
  <![CDATA[ JavaScript ]]>
</script>
<script type="text/javascript" src="URL"/>
```

где

- type — тип сценария. В настоящее время поддерживается только JavaScript (обязательный).
- src — URL внешнего сценария (необязательный).

Примечание

Если описать одновременно атрибут src и иметь CDATA между тегами `<script>` и `</script>`, платформа игнорирует всю информацию в CDATA.

Элемент `<script>` позволяет писать сценарий JavaScript внутри файла VoiceXML или ссылаться на файл JavaScript. Платформа Tellme имеет встроенную функцию JavaScript с именем `vxmllong` и объект с именем `vxmldata`.

Замечание

`vxmldata` является расширением языка.

Вызывающие теги: `<block>`, `<foreach>`, `<noinput>`, `<catch>`, `<form>`, `<nomatch>`, `<vxml>`, `<default>`, `<help>`, `<prompt>`, `<filled>`, `<if>``<else>``</elseif>`, `<result>`.

Порожденные теги: нет.

Пример

```
<script type="text/javascript">
<![CDATA[
  var team_name = "knicks";
  session_data = "return_state=SP_ScoresMainMenu&team_name="+team_name;
  vxmllong("session_data = ",session_data);
]]>
</script>
```

Тег `<subdialog>`

Переходит на описанный URL приложения.

Синтаксис

```
<subdialog src="URL"
           name="string"
           namelist="string"
           method="get_post">
```

где

- ❑ `src` — URL документа или ссылка внутри текущего документа (обязательный).
- ❑ `name` — объект JavaScript, который будет создаваться для хранения возвращаемых результатов.
- ❑ `namelist` — список переменных, которые должны быть указаны в http-запросе (необязательный).
- ❑ `method` — метод используемого HTTP-запроса — GET (по умолчанию) или POST (необязательный).

Элемент `<subdialog>` описывает URL перехода, аналогично директиве `<goto>`. Есть две существенных разницы. Первое, чтобы вернуться из файла,

который вызывается, нужно использовать директиву `<return>`. Второе, `<subdialog>` может появиться только в качестве порожденного элемента от `<form>`, а не в исполняемом контексте, в каком может появляться `<goto>`.

Когда платформа встречает элемент `<subdialog>`, его кладут в стек поддиалога. Если директива `<return>` встречается в другом файле, управление передается к верхнему элементу стека поддиалога, который затем удаляется из стека. Управление можно вернуть тремя путями.

- ❑ Использовать пустой элемент `<return>`, например, `<return/>`. Этот метод возвращает управление к элементу `<filled>` с элементом `<subdialog>`.
- ❑ Описать событие, используя атрибут `event` элемента `<return>`. Этот способ возвращает управление к обработчику событий, описанному в элементе `<subdialog>` и его порожденных элементах.
- ❑ Описать элемент `<return>` со списком `namelist`. Этот метод возвращает управление к элементу `<filled>` внутри `<subdialog>`.

Вызывающие теги: `<form>`.

Порожденные теги: `<catch>`, `<help>`, `<default>`, `<noinput>`, `<filled>`, `<nomatch>`.

Пример

Пример демонстрирует использование `<subdialog>` вместо `<gosub>` при переходе к другому URL приложения. Он также показывает, как передавать переменные с использованием элемента `<param>`. Две переменные передаются конкатенированными, `pass_command` и `another_pass_command`, а конкатенированные данные возвращаются в `got_concat`.

`concat.vxml`:

```
<?xml version="1.0"?>
<!DOCTYPE vxml PUBLIC "-//Tellme Networks//Voice Markup Language 1.0//EN"
"http://resources.tellme.com/toolbox/vxml-tellme.dtd">
<vxml application="init.vxml">
  <meta name="scoping" content="new"/>
  <var name="got_concat"/>
  <form id="choices">
    <field name="choices">
      <grammar>
        <![CDATA[
          [
            [dtmf-1 concat]{<option "concat">}
          ]
        ]]>
```

```

</grammar>
<prompt>
  <audio> Say "concat" to concat two words. </audio>
</prompt>
<default>
  <audio> i'm sorry...i didn't catch that </audio>
  <pause> 750 </pause>
  <goto next="#choices"/>
</default>
<filled>
  <log> {choices} </log>
  <assign name="pass_command" expr="'choices'"/>
  <assign name="another_pass_command" expr="'stuff'"/>
  <goto next="#pass"/>
</filled>
</field>
</form>
<form id="pass">
  <subdialog name="what"
src="http://studio.tellme.com/voicexmlref/concat.cgi">
    <param name="pass_cmd" expr="pass_command"/>
    <param name="a_pass_cmd" expr="another_pass_command"/>
    <catch event="state.happy">
      <audio> you passed some values </audio>
      <pause> 700 </pause>
      <audio> concat result equals {got_concat} </audio>
      <log>{got_concat}</log>
      <goto next="#choices"/>
    </catch>
    <catch event="state.nothing">
      <assign name="pass_state"
expr="'I was not able to get your value'"/>
      <goto next="#pass_failure"/>
    </catch>
    <default>
      <goto next="#pass_failure"/>
    </default>
  </subdialog>
</form>

```

```

<form id="pass_failure">
  <block>
    <audio> I'm sorry...there was a problem </audio>
    <pause> 750 </pause>
    <audio> {pass_state} </audio>
    <pause> 750 </pause>
    <goto next="#choices"/>
  </block>
</form>
</vxml>

```

Корневой документ приложения иницирует все переменные приложения.

init.vxml:

```

<vxml>
  <meta name="scoping" content="new"/>
  <!-- application root document with initialized application variables -->
  <form id="init">
    <block>
      <var name="pass_state" expr="''"/>
      <var name="pass_command" expr="''"/>
      <var name="another_pass_command" expr="''"/>
      <var name="gotconcat" expr="''"/>
    </block>
  </form>
</vxml>

```

Тег **<submit>**

Получает новый документ посредством HTTP GET или HTTP POST.

Синтаксис

```

<submit next="URL"
        namelist="variable_list"
        method="get_post"
        expr="JS_expression"/>

```

где

- **next** — URL, к которому нужно отправить запрос. Можно также включать { } в этот атрибут, чтобы заключить содержание, которое нужно использовать в модуле данных (обязательно, если не используется **expr**).
- **namelist** — передаваемый список параметров. По умолчанию, все именованные переменные из поля **<field>**. Для задания конкретных переменных нужно использовать **namelist** (необязательный).

Замечание

Не разрешается смешивать переменные `<record>` с переменными `<field>`.

❑ `method` — по умолчанию GET.

❑ `expr` — выражение JavaScript, которое оценивает URL (обязательно, если не используется `next`).

Элемент `<submit>` получает новый документ посредством HTTP GET или POST.

Вызывающие теги: `<block>`, `<filled>`, `<if><else/><elseif/>`, `<prompt>`, `<catch>`, `<foreach>`, `<noinput>`, `<result>`, `<default>`, `<help>`, `<nomatch>`.

Порожденные теги: нет.

Пример

```
<vxml>
  <form>
    <field name="document.input">
      <prompt>
        <audio>say yes or no</audio>
      </prompt>
      <filled>
        <var name="answer" expr="36"/>
        <log>** Call filled event **</log>
        <if cond="{document.input} == 'yes'">
          <audio>you said YES</audio>
        <else/>
          <audio>you said NO</audio>
        </if>
        <submit next="my_server/some.cgi" method="post"
          namelist="document.input answer"/>
      </filled>
    </field>
  </form>
</vxml>
```

Тег <throw>

Порождает системные или определенные пользователем события для последующей обработки специальной программой.

Синтаксис

```
<throw event="event"/>
```

где event — событие, которое нужно породить (обязательное).

Элемент <throw> порождает системные или определенные пользователем события, которые затем обрабатываются с помощью элемента <catch>. Когда порождается событие, платформа ищет соответствующий обработчик внутри элемента. Элемент наследует обработчики событий от вышестоящих элементов, так что в случае, если конкретный обработчик не найден, платформа ищет его внутри более высокого элемента. Имеется возможность порождать также события, определенные пользователем или предопределенные системные события. Например, можно порождать определяемые пользователем события для перехода в другую часть приложения. Или возможно возбуждать предопределенное событие, такое как <throw event="nomatch"/>. При этом нужно убедиться, что когда задано событие, соответствующий обработчик также задан.

Вызывающие теги: <catch>, <help>, <prompt>, <default>, <noinput>, <result>, <filled>, <nomatch>.

Порожденные теги: нет.

Пример

Пример показывает, как создать возбуждение события, определенного пользователем, и обработать его.

```
<vxml>
<grammar>
  <![CDATA[
    [
      [vote] {<option "vote">}
      [reset] {<option "reset">}
      [results] {<option "results">}
      [exit] {<option "exit">}
    ]
  ]]>
</grammar>
<form id="result2">
  <field name="document.skunkfood.choice">
```

```
<prompt>
  <audio>say vote, reset, results, or exit</audio>
</prompt>
<nomatch>
  <throw event="myevent"/>
</nomatch>
<filled>
  <result name="vote">
    <goto next="#vote"/>
  </result>
  <result name="reset">
    <goto next="#reset"/>
  </result>
  <result name="results">
    <goto next="#results"/>
  </result>
  <result name="exit">
    <goto next="#exit"/>
  </result>
</filled>
<catch event="myevent">
  <goto next="_home"/>
</catch>
</field>
</form>
</vxml>
```

Тег **<transfer>**

Переключает абонента на другой телефонный номер.

Синтаксис

```
<transfer dest="string"
          connecttimeout="seconds"
          maxlength="seconds">
  child elements
</transfer>
```

где

- ❑ `dest` — строка цифр набора (форматирование не разрешается). Например, "(305)444-1212" или "phone: 2033849" (обязательный).
- ❑ `connecttimeout` — число секунд ожидания при соединении, прежде чем возникнет событие `event.noanswer`. По умолчанию 30 секунд (необязательный).
- ❑ `maxlength` — число секунд или "0", если абонент может вводить данные произвольно долго. По умолчанию "0" (необязательный).

Примечание

Элемент `<transfer>` поддерживается в Tellme Studio, но имеет ограничение длины, равное одной минуте.

Элемент `<transfer>` набирает номер, заданный в атрибуте `dest`, и ожидает завершения вызова или отказа в соединении. В заключение либо порождается событие или выполнение восстанавливается немедленно после элемента `<transfer>`. Элемент `<transfer>` ведет себя как элемент `<field>` при успешном переназначении вызова (вызываемый абонент поднимает трубку); можно описать все атрибуты `<field>` внутри открывающего тега `<transfer>` и описать грамматики внутри, которые активны в течение вызова. Заметим, что платформа поддерживает только `magicword`-распознавание для этих грамматик, то есть событие `<nomatch>` никогда не порождается внутри элемента `<transfer>`.

Элемент `<filled>` используется для обработки успешного завершения процесса распознавания высказывания или DTMF-последовательности в элементе `<transfer>`. Затем имеется возможность переходить к другим элементам, очереди звуков, снова вызывать `<listen/>` или использовать директиву `<disconnect>`. Если выполняется переход к другому элементу (например `<goto>` или `<next>`) при наличии более чем одного активного абонента, система завершает звонки, прежде чем начать обработку.

События, порождаемые внутри `<transfer>`.

- ❑ `event.transfer.failure` — порождается, когда вызов не может быть завершен по некоторым причинам, таким как все исходящие порты заняты или в телефонном наборе имеются неверные символы.
- ❑ `event.busy` — порождается, когда вызываемый номер занят.

- ❑ `event.noanswer` — порождается, если вызываемый номер не отвечает в течение нескольких секунд, заданных в `connecttimeout`.
- ❑ `event.transfer.maxlen.exceeded` — порождается, если вызов вернулся по параметру `maxlength`.

Замечание

Если абонент повесил трубку, платформа завершает все связанные вызовы. Если вызываемая сторона вешает трубку, никаких событий не возникает и система продолжает исполнение приложения немедленно после элемента `<transfer>`. Со временем могут быть добавлены новые события и включены обработчики событий.

Вызывающие теги: `<form>`.

Порожденные теги: `<catch>`, `<dtmf>`, `<prompt>`, `<default>`, `<grammar>`.

Пример

Пример показывает, как обрабатывать события, порожденные элементом `<transfer>`.

```
<vxml>
  <form id="transfer">
    <transfer maxlength="60" dest="8005558355">
      <catch event="event.busy">
        <audio> busy </audio>
        <goto next="_home"/>
      </catch>
      <catch event="event.noanswer">
        <audio>no answer</audio>
        <goto next="_home"/>
      </catch>
      <catch event="event.transfer.maxlen.exceeded">
        <audio>max length exceeded</audio>
        <goto next="_home"/>
      </catch>
      <catch event="event.transfer.failure">
        <audio> failed </audio>
        <goto next="_home"/>
      </catch>
      <default>
        <audio>done</audio>
```

```

    <goto next="_home"/>
  </default>
</transfer>
</form>
</vxml>

```

Тег **<value>**

Вставляет строку содержания или выражение JavaScript.

Синтаксис

```

<value name="string"
      expr="JS_expression"/>

```

где

- ❑ **name** — имя переменной. Используется только если нужно оценить значение строковой переменной (необязательный).
- ❑ **expr** — значение переменной. Если значение — строка, необходимо заключить ее в одиночные кавычки. Если начальное значение не задано, остается текущее значение. Заметим, что в отличие от других вариантов использования **expr**, выражение **JS** не разрешается по имени переменной (обязательное, за исключением случаев использования **name**).

Элемент **<value>** позволяет вставлять содержание переменной в элементы **<audio>**, **<log>** и **<prompt>**. В этих случаях текст преобразуется в речь.

Вызывающие теги: **<audio>**, **<log>**, **<prompt>**.

Порожденные теги: нет.

Пример

Пример показывает процесс преобразования переменной **testvalue** в речь, проигрывая "yes" абоненту, и пишет переменную в **Debug Log**.

```

<vxml>
  <form id="test">
    <block>
      <var name="testvalue" expr="'yes'"/>
      <audio> testvalue is <value expr="testvalue"/> </audio>
      <log>testvalue is: <value expr="testvalue"/> </log>
    </block>
  </form>
</vxml>

```

```
</form>
</vxml>
```

Тег <var>

Объявляет локальную переменную.

Синтаксис

```
<var name="string"
      expr="JS_expression"/>
```

где

□ name — имя переменной (обязательный).

Замечание

Атрибут имени <var> может содержать буквы, цифры, подчеркивание "_" и знак доллара "\$", но первым символом должна быть буква или знак доллара.

□ expr — значение переменной. Если это значение является строкой, необходимо заключить его в кавычки. Если начальное значение не специфицировано, оно сохраняет свое произвольное текущее значение.

Элемент <var> объявляет локальную переменную посредством атрибута name. Локальная переменная хранит данные внутри области действия своего контейнера и доступна для всех порожденных контейнеров. Поэтому они разделяют то же пространство имен, что и переменные внутри элемента <script>. Необходимо следить за тем, чтобы давать переменной уникальное имя. Элемент <var> используется также для формального объявления параметров, которые передаются элементом <subdialog>.

Вызывающие теги: <block>, <filled>, <if><else/><elseif/>, <prompt>, <catch>, <foreach>, <noinput>, <result>, <default>, <help>, <nomatch>.

Порожденные теги: нет.

Пример

Объявленная локальная переменная используется в элементе <if>.

```
<vxml>
<form>
  <block>
    <var name="temp" expr="'eggplant'"/>
    <if cond="temp == 'eggplant'">
      <audio>yay eggplant</audio>
```

```
<else/>
  <audio>some other vegetable</audio>
</if>
<goto next="_home"/>
</block>
</form>
</vxml>
```

Этот пример объявляет переменную области действия документа, называемого "city", и назначает ей значение "Sacramento".

```
<vxml>
  <var name="total" expr="10+20"/>
</vxml>
```

Тег **<vxml>**

Содержит серию диалогов взаимодействия с абонентом.

Синтаксис

```
<vxml application=
    "http://resources.tellme.com/lib/universals.vxml">
version="string"
base="string"
lang="string"
child elements
</vxml>
```

где

- ❑ **application** — URL корневого документа приложения. Все телефонные приложения Tellme ссылаются на универсальный файл Tellme Studio как на корневой документ для создания прозрачной переносимости между сервисами Tellme и вновь создаваемыми услугами (обязателен).
- ❑ **version** — только для интернациональных данных.
- ❑ **base** — только для интернациональных данных.
- ❑ **lang** — только для интернациональных данных.

Все документы VoiceXML начинаются с элемента `<vxml>` и заканчиваются элементом `</vxml>`, формируя контейнер для всех остальных элементов приложения.

Вызывающие теги: нет.

Порожденные теги: <form>, <link>, <meta>, <script>, <grammar>, <menu>, <property>, <var>.

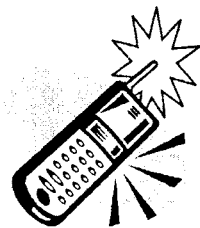
Пример

Пример начинает и завершает простой документ VoiceXML.

```
<?xml version="1.0"?>
<!DOCTYPE vxml PUBLIC "-//Tellme Networks//Voice Markup Language
1.0//EN" "http://resources.tellme.com/toolbox/vxml-tellme.dtd">
<vxml application="http://resources.tellme.com/lib/universals.vxml">
  <form>
    <block>
      <audio>Hello World!</audio>
      <goto next="_home"/>
    </block>
  </form>
</vxml>
```

Глава 8

Справочник по CallXML



Обзор

Язык CallXML является языком разметки, принадлежащим к семейству языков, порожденных спецификацией XML. Этот язык используется для описания телефонного пользовательского интерфейса, телефонного вызова поверх IP или мультимедийного приложения, основанного на CallXML браузере. Браузер CallXML может затем быть использован для последующего управления и взаимодействия с вызовом.

Язык CallXML включает.

- ☐ Элементы среды действия, такие как `<playAudio>` или `<recordAudio>` для описания того, что должно быть представлено пользователю в процессе звонка.
- ☐ Элементы обработки вызова, такие как `<answer>`, `<call>` или `<hangup>`, которые описывают, как управлять и маршрутизировать телефонный вызов.
- ☐ Элементы логических действий, такие как `<assign>`, `<clear>` и `<goto>`, описывают, как модифицировать переменные и взаимодействовать с традиционной Web-логикой, размещенной на http-сервере в виде сценариев на языках perl, PHP или ASP.
- ☐ Элементы обработки событий `<onTermDigit>`, `<onHangup>` для того, чтобы описывать, как реагировать на действия пользователя (такие как нажатие клавиши или вешание трубки) в процессе диалога с ним.
- ☐ Элементы блока, которые логически группируют действия и события вместе, так что один набор обработки событий может быть использован для нескольких последовательных действий.

Сравнение CallXML с HTML

HTML является языком разметки и используется для описания процесса взаимодействия пользователя с Web-приложением. Нижеприведенная таблица сравнивает некоторые элементы обоих языков.

Таблица 8.1. Сравнение элементов языков HTML и CallXML

HTML	CallXML
<code><html></code> начинает HTML-документ	<code><callxml></code> начинает CallXML-документ
<code><table></code> группирует вместе визуальные представления других элементов языка	<code><block></code> группирует вместе другие элементы CallXML, связанные с действиями и событиями
<code></code> показывает графический файл	<code><playAudio></code> проигрывает звуковой файл
<code></code> описывает, куда нужно идти, когда пользователь щелкает на ссылке	<code><onTermDigit></code> элемент описания события описывает, что нужно делать, когда пользователь нажимает на клавишу телефона

Сравнение CallXML и VoiceXML

VoiceXML создавался для того, чтобы упростить для Web-разработчиков процесс создания интерфейсов, основанных на распознавании голоса. Эти приложения могут работать для пользователей телефонов и одновременно для пользователей мультимедийных компьютеров. VoiceXML является прекрасным инструментом для решения задачи голосового обеспечения доступа к Web-содержанию и другой информации. Примерами таких приложений могут быть:

- ❑ приложения, которые позволяют пользователям извлекать Web-содержание с помощью телефона (например: голосовые порталы, Web по телефону и т. д.);
- ❑ приложения, которые позволяют пользователям взаимодействовать с услугами, расположенными в сети, с использованием голосовых команд (например: курсы акций, спортивные результаты и т. д.).

CallXML был создан для того, чтобы облегчить Web-разработчикам создание приложений, взаимодействующих и управляющих любым числом и типом телефонных вызовов, включая:

- ❑ телефонные или VoIP-приложения, которые могут управлять инициацией и маршрутизацией телефонного вызова, поддерживать такие свойства, как исходящий вызов, конференция, взаимодействия нескольких участников (например: мост конференции, ожидание internet-вызова, услуги follow-me/find-me и т. д.);
- ❑ телефонные или VoIP-приложения, которые могут легко взаимодействовать и отвечать на ввод кодов touch-tone или выбор (например: голосовая почта, IVR и т. д.):
 - телефонные приложения, которые включают поддержку дополнительных сред распространения, таких как факсы и видео (например: Unified messaging system — Интегрированная система обработки сообщений, видеоконференции и т. д.).

Из-за естественной сложности, связанной с обработкой голосовых команд, язык VoiceXML использует относительно сложную модель form/field/grammar/filled интерфейса.

В отличие от языка VoiceXML, язык CallXML использует упрощенную модель block/action/event интерфейс, который может быть более прост в изучении и который позволяет визуальным инструментам разработки прямо представлять разметку CallXML как простой график пользовательского интерфейса.

Пример CallXML-документа

Приведенный пример выполняет работу базового приложения автоответчика. Оно проигрывает поздравление, а затем записывает аудиосообщение. Записанное сообщение отправляется в адрес **abc@xyz.com**.

Однако, если вызывающий абонент нажмет клавишу "*" в любое время от начала поздравления или записи звукового сообщения, CallXML-браузер будет завершать режим проигрывания/записи и перейдет к новому URL для последующих инструкций. В свою очередь, если абонент повесит трубку в процессе проигрывания/записи, CallXML-браузер будет переходить к другому новому URL.

```
<?xml version="1.0" encoding="UTF-8">
<callxml>
  <block>
    <answer/>
    <playAudio value="http://xyz.com /greeting.wav/" termDigits="*">
    <recordAudio value="mailto:abc@xyz.com" maxTime="30s"
      termDigits="*" />
    <onTermDigit value="*">
    <goto value="http://xyz.com/cgi-bin/get_pin_code.pl"/>
    </onTermDigit>
    <onHangup>
    <goto value="http://xyz.com/end_call.xml"/>
    </onHangup>
  </block>
</callxml>
```

Если вы знаете HTML...

Если вы знакомы с HTML, а не XML, вы найдете, что второй совершенно аналогичен, несмотря на то, что стандарт XML требует большей точности при кодировании. Важные отличия включают.

- ❑ HTML-файлы часто называют страницами, в то время как XML-файлы обычно называют документами.

- ❑ Каждый элемент диалекта XML должен быть закрыт. Например, в HTML вы часто видите элемент `<p>` без закрывающегося элемента. Это не разрешается в XML. Вы должны закрыть такой элемент, например так: `<p>hi!</p>`. Более непривычно то, что пустой элемент также должен быть закрыт. Например, тег HTML `
` был бы закрыт в XML комбинацией `
`. Для совместимости с существующими интерпретаторами HTML, необходимо включать пробел перед закрывающим слэшем. Кроме того, желательно избегать эквивалентного написания вида: `
</br>`; так как многие HTML-интерпретаторы будут отказываться работать при интерпретации этой конструкции.
- ❑ XML очень ограничивает возможности вложения элементов. Если HTML-браузеры разрешают следующие конструкции: `<p>bold <i>italic </i>.</p>`, то XML требует, чтобы теги не перемешивались: `<p>bold <i>italic</i>.</p>`.

Несмотря на то, что HTML официально также требует выполнения тех же правил, ошибки так часто встречаются, что большинство HTML-браузеров обычно игнорирует их.

Элементы манипулирования переменными

CallXML-браузеры включают переменные, связанные с понятиями "вызов" или "сессия". Логические элементы CallXML используются для того, чтобы назначать и удалять значения этих переменных, а также передавать значения этих переменных в качестве частей http-запросов типа GET и POST для использования в традиционных Web-приложениях, которые написаны на языках C, Perl, Java или ASP.

К переменным CallXML можно обращаться внутри атрибутов элемента языка CallXML.

Тег `<assign>`

Присваивает значение описываемого атрибута `value` переменной, указанной в атрибуте `var`. Например, будет присваивать значение 123 переменной с именем `ttt`.

В дополнение к точному присвоению значений переменным языка CallXML браузеры могут автоматически создавать переменные, которые содержат информацию, связанную с вызовом/сессией.

Синтаксис

```
<assign>
```

```
<assign var="ttt" default: ""  
      value="123"/> default: ""
```

Атрибуты

□ `var` — имя переменной для назначаемого значения.

□ `value` — значение, присваиваемое переменной.

При организации каждой сессии автоматически создается следующий список глобальных переменных (табл.8.2).

Таблица 8.2. Список глобальных переменных

Имя	Описание
<code>Session.Digits</code>	Буфер цифр
<code>Session.CallerID</code>	Номер вызывающего абонента
<code>Session.CalledID</code>	Номер, который был вызван
<code>Session.EventSenderID</code>	ID сессии источника последнего внешнего события
<code>Session.CallerAccountID</code>	Voxeo ID вызывающего (если известно)
<code>Session.CalledAccountID</code>	Voxeo ID вызываемого (если известно)
<code>Session.ID</code>	Идентификатор текущей сессии

Имена переменных должны начинаться с буквы (от A до Z или от a до z) и могут содержать: буквы A—Z, a—z, цифры 0—9. Имена переменных могут иметь длину до 40 символов. Имена переменных, начинающихся с "Session." или "SESSION.", во всех случаях резервируются как внутренние системные переменные и не могут быть созданы или установлены программистом за исключением буфера `Session.Digits`, который может быть очищен использованием элемента `clearDigits` или элементом `clear` с атрибутом `var="Session.Digits"`. Буфер `Session.Digits` может быть установлен с использованием конструкции:

```
<assign var="Session.Digits" value="anything">
```

Возможные события: `<onError>`.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<!-- читать первое и последнее имя вызывающему, используя TTS -->
<callxml>
  <block>
    <assign var="firstname" value="jonathan"/>
    <assign var="lastname" value="smith"/>
  </block>
  <text>
    Thanks for calling, $firstname; $lastname;.
```

```
</text>
</callxml>
```

Тег **<clear>**

Очищает значение переменной атрибута `var`. Работает как `<assign var="ttt" value="" />`.

Синтаксис

```
<clear var="ttt"/>
```

Атрибуты

□ `var` — имя очищаемой переменной.

Возможные события: `<onError>`.

Тег **<clearDigits />**

Очищает буфер `Session.Digits`.

Буфер `Session.Digits` содержит любые цифры, которые пользователь может нажать, прежде чем выполнится какое-нибудь действие CallXML. Этот элемент будет освобождать буфер, в котором находится любое количество цифр.

Кроме освобождения буфера `Session.Digits`, некоторые элементы CallXML имеют атрибут `clearDigits`, который делает ту же самую вещь, чтобы уменьшить количество элементов, требуемых при выполнении более общих задач.

Возможные события: `<onError>`.

Тег **<goto>**

Этот элемент может либо перейти к другому блоку действий CallXML в текущем файле, заданному параметром `value="#blocklabel"`, либо выполнить запрос HTTP GET или POST, чтобы перейти к следующему CallXML-документу, задав `value="url"`.

Синтаксис

```
<goto value="http://w.v.n/next.voxeo#block"   Default: ""
      submit="*|x,y,z"                        Default: "*"
      method="get|post"                       Default: "get" />
```

Атрибуты

□ `value` — либо полный URL (<http://w.v.n/yo.voxeo>), либо локальный URI, указывающий на метку `<block>` в том же самом файле CallXML например, `#main_menu`. Поддерживаемые форматы URL включают:

- `http://` данные;
- `ftp://` данные.

- ❑ `submit` — список переменных для вызываемого URL/URI, который может быть либо `all`, либо `*`, или разделенный запятыми список переменных (например: `submit = "Variable1, Variable2, Variable3, Variable5, Variable9"`).
- ❑ `method` — используемый метод `http`:
 - `get` для HTTP GET;
 - `post` для HTTP POST.

Когда элемент используется для обращения к новому документу CallXML, атрибут `submit` может быть использован для передачи браузеру значений переменных в HTTP GET или POST методе. Атрибут `method` использован для указания метода HTTP GET или POST. Возможные события: `<onError>`.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<!-- назначить две переменные, затем послать их программе на perl
xyz.com как переменную GET URL -- >
<callxml>
  <block>
    <assign var="firstname" value="jonathan"/>
    <assign var="lastname" value="smith"/>
    <goto value="http://xyz.com/cgi-bin/app.pl"
      submit="firstname, lastname"/>
  </block>
</callxml>
```

Замечание относительно элементов сессии

CallXML-браузеры могут поддерживать более одного "вызова" или "сессии". Например, CallXML браузер может быть запущен на системе, связанной с несколькими телефонными линиями, при этом поддерживая уникальные переменные сессии для каждой телефонной линии.

Для такого CallXML-браузера, CallXML-код в одной сессии может запускать новый CallXML-код во второй сессии, используя элемент `<run>`. Этот элемент полезен для приложений, которые могут получать вызов в одной сессии, а затем инициировать новый исходящий вызов для другой сессии — например, при реализации услуги `follow-me/find-me`.

В таких браузерах CallXML некоторые приложения могут требовать способность легкой отправки информации между сеансами, которые могут потенциально прервать действия другого сеанса.

Например, приложение конференции может захотеть иметь способность посылать сообщение из сеанса для каждого вызываемого участника к сеансу пер-

соны, который создавал конференцию, так что "новый вызывающий" тон может быть проигран для слушающего создателя конференции. В результате, CallXML включает базовый механизм отправки сообщения от одного сеанса к другому.

CallXML логические элементы, связанные с сессиями представлены ниже.

Тег <run>

Этот элемент будет запускать новый сеанс и извлекать CallXML-документ для этого сеанса из URL или URI.

Синтаксис

```
<run value="http://w.v.n/next.voxeo|#block" Default: ""
    submit="*|x,y,z" Default: "*"
    method="get|post" Default: "get"
    var="varForReturnedSessionID" Default: "" />
```

Атрибуты

- ☐ **value** — указывает либо полный URL (<http://w.v.n/yo.voxeo>), либо локальный URI, задавая метку <block> в том же самом файле (например, #main_menu). Поддерживаемые форматы URL:
 - http:// данные;
 - ftp:// данные.
- ☐ **submit** — список переменных, передаваемых к вызываемому URL/URI.
- ☐ **method** — используемый метод обращения:
 - get для HTTP GET;
 - post для HTTP POST.
- ☐ **var** — имя переменной, в которой хранится ID сеанса.

Атрибут submit может также использоваться для передачи копий переменных к CallXML-браузеру из родительского сеанса.

Возможные события: <onError>.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<!--запустить новый сеанс -->
<!--передать несколько копий нескольких переменных в код @ xyz.com -->
<callxml>
    <block>
        <assign var="firstname" value="jonathan"/>
```

```

<assign var="lastname" value="smith"/>
<assign var="parentSession" value="$Session.ID;"/>
<run value="http://xyz.com/newsess.asp"
      submit="fistname, lastname, parentSession"
      var="childsessid"/>
      <!-- ID сеанса -->
<!--хранимый в childsessid -->
</block>
</callxml>

```

Тег <sendEvent>

<sendEvent> — это тег, который позволяет новому сеансу посылать сообщение другому сеансу.

Синтаксис

```

<sendEvent value="msg_call_answered"   Default: ""
          Session="sss"                Default: "" />

```

Атрибуты

- ☐ value — строка, содержащая тело сообщения.
- ☐ Session — ID сеанса, которому предназначено событие.

Возможные события: <onExternalEvent>.

Это событие позволяет сеансу, который является получателем <sendEvent>, обрабатывать это сообщение. В процессе обработки один сеанс может стартовать, другой ожидать порожденный им процесс для отправки сообщения, указывая какой подчиненный сеанс должен завершить свою задачу. Например, это упрощенное взаимодействие может быть использовано в качестве основы для следующего приложения "follow me/find me."

Пример

Сеанс 1:

```

<block>

<assign var="parent"
      value="$Session.ID;" />

<run value="Session2.xml"
      submit="*" />

<delay>

<onExternalEvent alue="foo">

```

```
<simline value="caught a foo external event." />
</onExternalEvent>
</block>
```

Сеанс 2:

```
<block>
  <simline value="send an event." />
  <sendEvent value="foo"
              Session="$parent;" />
</block>
```

CallXML — выполнение действий с вызовом

Элементы действий с вызовом языка CallXML описывают действия, которые CallXML-браузер может совершать над вызовом, связанным с текущим сеансом браузера. Эти действия включают элементы `<answer>` для ответа на входящий звонок, `<hangup>` для выполнения разъединения вызова, `<call>` для инициации исходящего вызова и `<conference>` для выполнения связи или конференции различных аудиовызовов или связи с текущим сеансом в единое целое.

Тег `<answer/>`

Этот элемент будет "отвечать на" или "подхватывать" вызов. Каждый раз при поступлении нового вызова к CallXML-браузеру, он использует механизм определения URL, который нужно использовать для обработки звонка. Этот механизм зависит от типа браузера и вызывает соответствующий CallXML-документ. Однако CallXML-браузер не отвечает на вызов до тех пор, пока не закончится исполнение элемента.

Возможные события: `<onError>`.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<!-- ответить на звонок, затем проиграть поздравление -->
<callxml>
  <block>
    <answer/>
    <playAudio value="greeting.wav" />
  </block>
</callxml>
```

Тег <hangup/>

Этот элемент заставляет браузер CallXML разъединить соединение, связанное с сеансом.

Возможные события: <onError>.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<!-- ответить на звонок, затем проиграть поздравление, затем пове-
снуть трубку-- >
<callxml>
  <block>
    <answer/>
    <playAudio value="greeting.wav"/>
    <hangup/>
  </block>
</callxml>
```

Тег <call>

Элемент <call> позволяет помещать исходящий вызов в конкретный адрес. Адрес задается атрибутом value и находится в формате URL. Поддерживаемым форматом является PSTN://, который указывает браузеру, куда нужно звонить по нормальной телефонной линии.

Синтаксис

<call value="pstn:18314395130"	Default: ""
callerID="pstn:1234567890"	Default: "callerID"
maxTime="30s"	Default: "30s" />

Атрибуты

- ☐ value — URL описывает место для инициации вызова.
- ☐ callerID — CallerID представляет, когда начинать вызов.
- ☐ maxTime — максимальное время ожидания для ответа на вызов.

Возможные события: <onError>, <onAnswer type="person|machine|unknown">, <onCallFailure>, <onMaxTime>.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<!-- активировать вызов, после получения ответа проиграть поздрав-
ление --- >
<callxml>
```

```

<block>
  <call value="pstn:14075551212"/>
  <onAnswer>
    <playAudio value="greeting.wav"/>
  </onAnswer>
</block>
</callxml>

```

Тег **<conference>**

Элемент позволяет нескольким линиям в разных сеансах связываться между собой вместе таким образом, что участники каждой линии могут говорить друг с другом одновременно.

Синтаксис

```

<conference targetSessions="SessionID1, SessionID2"      Default: ""
            termDigits="#"                               Default: "" />

```

Атрибуты

- ☐ **targetSessions** — задает список взаимодействующих сеансов (один или более уникальных идентификаторов сеанса, разделенных запятыми).
- ☐ **termDigits** — задает список цифр, которыми разрешено завершать конференцию.

Возможные события: <onError>.

Пример

```

<?xml version="1.0" encoding="UTF-8">
<!-- conference with another Сеанс -->
<!-- предположим, мы предварительно сохранили в переменной id количество сеансов "otherSess" -->
<!-- ждите, пока закончится конференция или наступит другое событие -->
<callxml>
  <block>
    <conference targetSessions="$othersess;"/>
    <delay value="nolimit"/>
  </block>
</callxml>

```

Тег **<waitForConferenceEnd />**

Элемент позволяет сеансу, для которого предназначена конференция, указывать заснуть всем другим обработкам вызовов до тех пор, пока конферен-

ция не завершится любым из участников. Элемент используется как альтернатива более общему элементу `<wait>`.

Возможные события: `<onError>`.

Тег `<wait>`

Элемент используется для указания CallXML-браузеру ожидать в течение заданного интервала времени.

Синтаксис

```
<wait value="10s|nolimit"           Default: ""
      termDigits="*" />           Default: ""
```

Атрибуты

- ☐ `value` — количество времени ожидания.
- ☐ `termDigits` — список цифр, завершающих действие ожидания.

Возможные события: `<onTermDigit>` `<onHangup>` `<onError>`.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<!--проиграть audio файл, затем ждать 30 секунд пока абонент не на-
жет клавишу или повесит трубку -->
<callxml>
  <block>
    <playAudio value="waitingonyou.wav"/>
    <delay value="30s"/>
  </block>
</callxml>
```

CallXML — действия уровня среды распространения

Тег `<getDigits>`

Обычно этот элемент используется для таких задач, как сбор PIN-кодов, номеров пейджеров и других данных, связанных с вводом цифровых последовательностей от абонента. `<getDigits>` требует связанного с ним события `<onTermDigit>`.

Синтаксис

```
<getDigits var="pager_msg"           default: none
      maxDigits="9"                 default: nolimit
```

<code>termDigits="#*"</code>	<code>default: ""</code>
<code>includeTermDigit="TRUE FALSE"</code>	<code>default: FALSE</code>
<code>clearDigits="TRUE FALSE"</code>	<code>default: FALSE</code>
<code>maxTime="30s"</code>	<code>default: 30s</code>
<code>maxSilence="5s"</code>	<code>default: 5s /></code>

Атрибуты

- ❑ `var` — записывает считанные цифры в переменную (в примере `"pager_msg"`).
- ❑ `maxDigits` — максимальное число считываемых цифр (в примере — 9).
- ❑ `termDigits` — цифры, которые завершают или заканчивают ввод цифр; (либо #, либо * в примере). Значение `termDigits` может быть любой комбинацией 1234567890*#ABCD@ или any, или "". any указывает, что любая цифра будет завершать ввод.
- ❑ `includeTermDigit` — TRUE для того, чтобы включить завершающую цифру в вводимую строку `var`, или FALSE, чтобы не включать ее.
- ❑ `clearDigits` — булевское значение, указывающее, должен ли очищаться буфер `Session.Digits` при начале действия. TRUE очищает буфер ввода. FALSE оставляет содержание буфера.
- ❑ `maxTime` — максимальное время ожидания цифры (в примере — 30 секунд). Время может быть задано в ms (миллисекунды), s (секунды) или m (минуты). По умолчанию предполагаются секунды.
- ❑ `maxSilence` — максимальное время ожидания между цифрами. Время может быть задано в ms (миллисекунды), s (секунды) или m (минуты). По умолчанию предполагаются секунды.

Возможные события: `<onTermDigit>`, `<onMaxDigits>`, `<onMaxTime>`, `<onMaxSilence>`.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<!--проиграть звуковой файл, запрашивающий ввод PIN-кода из 4 цифр от
абонента, их получение и сохранение, ждать 30 сек, затем сказать,
какой PIN-код был введен с использованием text to speech -->
<callxml>
  <block>
    <playAudio value="enterpin.wav"/>
    <getDigits var="user_pin"
              maxDigits="4"
              maxTime="30s"/>
    <onMaxDigits>
      <text> you entered $user_pin; </text>
    </onMaxDigits>
```

```
</block>
</callxml>
```

Тег <play... />

Этот тег используется для проигрывания числа, даты, количества денег, цифрового значения или звукового файла.

Синтаксис

<code><playNumber format="digits number"</code>	Default: "digits"
<code>value="12345"</code>	Default: ""
<code>termDigits="*#"</code>	Default: ""
<code>clearDigits="TRUE FALSE"</code>	Default: "FALSE" />
<code><playMoney format="us"</code>	Default: "us"
<code>value="1.25"</code>	Default: ""
<code>termDigits="*#"</code>	Default: ""
<code>clearDigits="TRUE FALSE"</code>	Default: "FALSE" />
<code><playDate format="ddmmyyyyhhss"</code>	Default: as shown
<code>value="1012990732"</code>	Default: ""
<code>termDigits="*#"</code>	Default: ""
<code>clearDigits="TRUE FALSE"</code>	Default: "FALSE" />
<code><playchars value="abcdefgh"</code>	
<code>termdigits="*#"</code>	
<code>cleardigits="TRUE FALSE"/></code>	
<code><playtone value="2000hz+1000hz"\"</code>	
<code>termdigits="*#"</code>	
<code>cleardigits="TRUE FALSE"/></code>	
<code><playAudio format="audio/vox"</code>	Default: "audio/vox"
<code>value="http://www.ttt.com/sample.vox"</code>	Default: ""
<code>termDigits="*#"</code>	Default: ""
<code>clearDigits="TRUE FALSE"</code>	Default: "FALSE" />

Атрибуты

- ☐ `format` — форматирование строки для проигрывания (не специфицировано).
- ☐ `value` — проигрываемое значение (literal, variable или URL).
- ☐ `termDigits` — задает список цифр, которыми разрешено прерывать процесс проигрывания.
- ☐ `clearDigits` — булевское значение, указывающее, должен ли очищаться буфер Session.Digits. TRUE очищает буфер. FALSE оставляет буфер как есть.

Для `<playAudio>` атрибут `format` указывает mime-type звукового файла, в этом случае CallXML-браузер не может автоматически определить mime-type.

Возможные события: `<onError>`.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<!-- ===== -->
<!-- Here is our main menu script -->
<!-- ===== -->
<callxml>
<block label="MainMenu"
  repeat="3"
  clearDigits="TRUE"/>
<playAudio format="audio/vox"
  value="http://www.blahblah.com/mainmenu.vox"
  termDigits="567890*#"
  clearDigits="TRUE"/>
<!-- ===== -->
<!-- Our event handlers -->
<!-- ===== -->
<onTermDigit value="*#">
  Do something here.
</onTermDigit>
<onTermDigit value="567890">
  Do something different here.
</onTermDigit>
</block>
</callxml>
```

Тег `<recordAudio>`

Используется для записи звука в атрибут `value`, заданный в URL.

Синтаксис

<code><recordAudio format="audio/vox"</code>	Default: "audio/vox"
<code>value="ftp://www.v.n/msg.vox"</code>	Default: ""
<code>termDigits="*#"</code>	Default: ""
<code>clearDigits="TRUE FALSE"</code>	Default: "FALSE"
<code>maxTime="30s"</code>	Default: "30s"

```

maxSilence="5s"                Default: "5s"
beep="TRUE|FALSE"             Default: "TRUE"

/>

```

Атрибуты

- ☐ **format** — задает mime-type для использования при записи сообщения. Возможные значения включают audio/vox, audio/ms-gsm и audio/wav.
- ☐ **value** — описывает URL или URI, куда должен быть записан звуковой файл. Поддерживаемые типы URL/URI включают:
 - http:// HTTP-адрес;
 - ftp:// FTP-адрес.
 - mailto: с прицепленным файлом
- ☐ **termDigits** — задает список цифр, которым разрешено прерывать процесс записи звукового файла.
- ☐ **clearDigits** — булевское значение, указывающее, должен ли очищаться буфер Session.Digits. TRUE очищает буфер. FALSE оставляет буфер как есть.
- ☐ **maxTime** — определяет максимальное время записи.
- ☐ **maxSilence** — задает максимальный период паузы, после которого запись будет прервана.

Элемент можно использовать для записи голосового сообщения, поздравления и т. д. В вышеприведенном примере абоненту позволено говорить до 30 секунд с паузой до 5 секунд и допускается завершение записи нажатием ключей * или #. Новый файл будет затем сохранен на **www.v.n/msg.vox** в формате audio.vox. Атрибут clearDigits обеспечивает очищение буфера при записи звукового файла в случае нажатия завершающих клавиш до инициации записи.

Возможные события: <onError>.

Пример

```

<?xml version="1.0" encoding="UTF-8">
<!-- Here is our record greeting script -->
<callxml>
  <block label="RecordGreeting"
    repeat="3"
    clearDigits="TRUE"/>
  <recordAudio format= "audio/vox"
    value="ftp://www.blahblah.com/greeting.vox"
    termDigits="1234567890*#"
    clearDigits="TRUE"
  />
</callxml>

```

```

        maxTime="60s"
        maxSilence="7s"/>
<!-- Our event handlers -->
<onTermDigit value= "#">
    Do something here.
</onTermDigit>
<onTermDigit value= "1234567890*>
    Do something different here.
</onTermDigit>
</block>
</callxml>

```

Тег <text>

Элемент применяется для указания CallXML-браузеру использовать Text-To-Speech — механизм для чтения текста, содержащегося внутри элемента, абоненту.

Синтаксис

```

<text format="?"                               Default: "connected"
    termDigits="#"                             Default: ""
    clearDigits="TRUE|FALSE"                   Default: "FALSE">
    Озвучиваемый текст ...
</text>

```

Атрибуты

- ☐ **format** — задает формат, который должен быть использован для чтения текста **termDigits** — список завершающих клавиш.
- ☐ **clearDigits** — переменная, задает условия очистки буфера **Session.Digits**. **TRUE** очищает буфер. **FALSE** оставляет буфер как есть.

Возможные события: <onError> <onTermDigit>.

Пример

```

<?xml version="1.0" encoding="UTF-8">
<!-- read a paragraph to the caller -->
<callxml>
    <block>
        <text>
            Now is the time for all good folks to
            Answer their telephone.
        </text>
    </block>
</callxml>

```

```
</block>
</callxml>
```

Тег **<addChannel>**

Элемент используется для добавления дополнительных каналов в текущий сеанс, такие как видео, программируемая презентация.

Синтаксис

<code><addChannel value="?"</code>	Default: ""
<code>format="?"</code>	Default: ""
<code>varChannelID="?"></code>	Default: ""

Атрибуты

- ☐ `value` — URL задает тип и адрес добавляемого канала.
- ☐ `format` — разделенный запятыми список `mime-type` звукового канала.
- ☐ `varChannelID` — имя переменной, которая будет получать уникальный идентификатор нового канала.

Возможные события: `<onError>`.

Тег **<removeChannel>**

Элемент используется для удаления предварительно добавленного канала из текущего сеанса.

Синтаксис

<code><removeChannel channelID="?"></code>	Default: ""
--	-------------

Атрибуты

- ☐ `channelID` — уникальный идентификатор удаляемого канала.

Возможные события: `<onError>`.

Элементы уровня блока

Элементы верхнего уровня существуют в языке CallXML для того чтобы упростить процесс создания приложений. В каждом случае они могут быть представлены как последовательность действий одного блока.

Тег **<block>**

Элемент `<block>` используется для логической группировки действий и событий вместе, обеспечивая, таким образом, способность повторять дейст-

вия в блоке заданное количество раз. <block>-элементы могут быть вложенными один в другой.

Синтаксис

```
<block label="anyname"           Default: ""
  repeat="?"                     Default: "1"
  clearDigits="TRUE|FALSE" >    Default: "FALSE"
  action Элемент ...
  event Элемент ...
</block>
```

Атрибуты

- ☐ clearDigits — переменная задает условия очистки буфера Session.Digits. TRUE очищает буфер. FALSE оставляет буфер как есть.
- ☐ label — уникально идентифицирует блок внутри CallXML-документа, и может использоваться элементом <goto> для перехода от одного блока к другому, или перейти в заданный блок другого документа.
- ☐ repeat — описывает количество повторений действий в блоке.

Возможные события:

```
<onAnswer></onAnswer>
<onCallFailure></onCallFailure>
<onError></onError>
<onHangup></onHangup>
<onMaxDigits></onMaxDigits>
<onMaxPages></onMaxPages>
<onMaxTime></onMaxTime>
<onMaxSilence></onMaxSilence>
<onTermDigit></onTermDigit>
<onExternalEvent></onExternalEvent>
```

Тег <menu>

Элемент, который комбинирует атрибуты и функциональность элемента <block> с элементом <playAudio>.

Синтаксис

```
<menu label="main_menu"         Default: ""
  repeat="3"                     Default: "1"
  format="audio/vox"             Default: "audio/vox"
  value="http://w.v.n/msg.vox"   Default: ""
```

```

clearDigits="TRUE|FALSE"           Default: "FALSE"
termDigits="567890*#"             Default: ""
maxTime="15s" >                   Default: "30s"
event Элемент ...
</menu>

```

Атрибуты

- ☐ **label** — метка блока для перехода к нему с использованием элемента `<goto>`.
- ☐ **repeat** — число повторений блока.
- ☐ **clearDigits** — переменная задает условия очистки буфера `Session.Digits`. `TRUE` очищает буфер. `FALSE` оставляет буфер, как есть.
- ☐ **format** — строка форматирования, используемая для проигрывания (полностью не определено).
- ☐ **value** — проигрываемое значение (константа, переменная или URL).
- ☐ **termDigits** — задает список цифр, которыми разрешено прерывать процесс проигрывания.

Целью исполнения элемента является в упрощении исполнения различных меню, где одно нажатие клавиши будет приводить к движению абонента по приложению. В вышеприведенном примере (и в последующем примере) звуковой файл будет проигрываться три раза, прежде чем наступит событие `timing out` и движение по `CallXML`-коду. Сравните нижеприведенный пример с предыдущим.

Возможные события: `<onTermDigit value="n">`, `<onTermDigit value="2">`, `<onMaxTime value="1|2|max">`, `<onHangup>`.

Пример

```

<?xml version="1.0" encoding="UTF-8">
<!-- You have a call submenu -->
<callxml>
  <menu label="YouHaveACall"
        repeat="3"
        format="audio/vox"
        value="http://143.186.161.8/Z/YouHaveACall.vox"
        clearDigits="TRUE"
        termDigits="#"
        maxTime="15s">
    <onTermDigit value="1">
      Do something to answer the call.
    </onTermDigit>
  </menu>
</callxml>

```

```

<onTermDigit value="2">
  Do something to save the call for later user.
</onTermDigit>
<onTermDigit value="#">
  Do something to return to the previous menu.
</onTermDigit>
</menu>
</callxml>

```

Тег **<inputDigits>**

<inputDigits> является элементом, который комбинирует функциональность и атрибуты элементов **<block>**, **<playAudio>** и **<getDigits>**.

Синтаксис

<inputDigits	label="input_pin"	Default: ""
	repeat="3"	Default: "1"
	var="pager_msg"	Default: ""
	format="audio/vox"	Default: "audio/vox"
	value="http://w.v.n.msg.vox"	Default: ""
	termDigits="#*"	Default: ""
	clearDigits="TRUE FALSE"	Default: "FALSE"
	includeTermDigit="TRUE FALSE"	Default: "TRUE"
	maxDigits="4"	Default: no limit
	maxTime="15s"	Default: "30s"
	maxSilence="5s" >	Default: "5s"

event Элемент ...

</inputDigits>

Атрибуты

- ☐ **label** — метка блока, используемая для ссылок.
- ☐ **repeat** — число повторений блока.
- ☐ **clearDigits** — переменная задает условия очистки буфера Session.Digits. TRUE очищает буфер. FALSE оставляет буфер, как есть.
- ☐ **var** — считывает цифры в заданную переменную (пример pager_msg).
- ☐ **maxDigits** — максимальное количество читаемых цифр.
- ☐ **termDigits** — завершающие цифры либо #, либо * в примере. Эти цифры могут быть любыми комбинациями 1234567890*#ABCD@ или any, или "".
- ☐ **includeTermDigit** — TRUE, чтобы включить завершающие цифры в переменную и FALSE, чтобы не включать.

- `maxTime` — максимальное время периода ожидания (30 секунд в примере).
- `maxSilence` — максимальное время паузы между соседними цифрами (5 секунд в примере). Время может быть задано в `ms` (миллисекунды), `s` (секунды) или `m` (минуты).

Элемент предназначен для упрощения создания скриптов, которые запрашивают ввод данных от абонента (ввод DTMF-последовательностей), таких как PIN-коды, сообщения пейджерам и числовые значения.

В вышеприведенном примере абонент имеет 15 секунд на ввод 4-х цифр (возможно PIN-код), с паузой не более 5 секунд между нажатиями. Либо #, либо * будут завершать процесс ввода, а звуковое сообщение — подсказка будет играть 3 раза, прежде чем завершит этот элемент и начнет обработку оставшейся части кода CallXML.

Возможные события: `<onTermDigit value="n">`, `<onMaxDigits>`, `<onMaxTime>`, `<onMaxSilence>`, `<onHangup>`.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<callxml>
  <block>
    <inputDigits value="enterpin.wav"
      var="user_pin"
      maxDigits="4"
      maxTime="30s" />
    <onMaxDigits>
      <text> you entered $user_pin; </text>
    </onMaxDigits>
  </block>
</callxml>
```

Тег `<inputAudio>`

Элемент `<inputAudio>` комбинирует атрибуты и функциональность элементов `<block>`, `<playAudio>` и `<recordAudio>`.

Предназначен для ускорения создания кода CallXML, который запрашивает ввод абонента путем проигрывания звука и записи звука в голосовую почту.

Синтаксис

<code><inputAudio label="leave_message"</code>	Default: ""
<code>repeat="3"</code>	Default: "1"
<code>var="myaudio"</code>	Default: ""
<code>playValue=http://w.v.n/msg.wav</code>	Default: ""

playFormat="audio/vox"	Default: "audio/vox"
recordValue=mailto:bob@xyz.com	Default: ""
recordFormat="audio/vox"	Default: "audio/vox"
termDigits="*#"	Default: ""
clearDigits="TRUE FALSE"	Default: "FALSE"
maxTime="15s"	Default: "30s"
maxSilence="5s"	Default: "5s"
beep="TRUE FALSE" >	Default: "TRUE"
event Элемент ...	

</inputAudio>

Атрибуты

- ❑ label — метка блока для возможности обращения к нему.
- ❑ repeat — число повторений блока <block>.
- ❑ clearDigits — переменная задает условия очистки буфера Session.Digits. TRUE очищает буфер. FALSE оставляет буфер, как есть.
- ❑ playFormat — строка форматирования для проигрывания (mime-type).
- ❑ recordFormat — строка форматирования, используемая для записи (mime-type).
- ❑ playValue — какую подсказку проигрывать (URL-ссылка).
- ❑ recordValue — куда писать звуковой файл (переменная или URL-ссылка).
- ❑ termDigits — цифры, завершающие проигрывание.
- ❑ maxTime — максимальное время периода проигрывания звука (ms, s или m).
- ❑ maxSilence — максимальная длительность периода записи.

Возможные события: <onTermDigit value="#"> <onMaxTime> <onMaxSilence> <onHangup>.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<callxml>
  <block>
    <answer />
    <inputAudio value="nothome.wav"
      value2="mailto:bob@xyz.com"
      format="audio/ms-gsm"
      maxTime="3m" />
    <hangup/>
  </block>
</callxml>
```

Элементы задания событий CallXML

События CallXML не всегда являются полными обработчиками событий как таковых, несмотря на то, что обработка событий может быть сконструирована посредством этих элементов. В большинстве случаев они в основном являются логическими управляющими структурами, которые комбинируют определенные свойства обработки событий и возможности исполняемого программного кода. Если значения и типы определены, структура кода выполняет действия типа `if... then...`, выбирая некоторую последовательность инструкций, основанных на значении события, которое переключает обработчик. При таком использовании полный обработчик события для конкретного события будет требовать так много элементов `on`, сколько возможно вложить. Элементы `on` без атрибутов требуют только один вход.

Тег `<onAnswer>`

Это событие возникает, когда сеанс определяет, что на вызов было что-то отвечено. Используется в сценарии CallXML после выполнения действия `<call>`.

Синтаксис

```
<onAnswer type="possibleAnswerType"/>
```

Атрибуты

- `type` — сервер CallXML будет пытаться определить, что отвечалось на вызов, и создавать отчет о выполненных действиях. Это может быть полезно при определении последовательности выполненных действий. Тип, заданный в `<onAnswer>`, будет сравниваться с действительным типом ответа и в случае совпадения выполнять код из элемента `<onAnswer>`. Правильные значения: `person`, `machine` и `unknown`.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<callxml>
  <block>
    <call value="pstn://14075551212"/>
    <onAnswer>
      <playAudio value="hello.wav"/>
    </onAnswer>
    <onCallFailure>
      ...
    </onCallFailure>
  </block>
</callxml>
```

Тег **<onCallFailure>**

Это событие возникает, когда сеанс определяет вызов и не может быть завершен штатно по причине занятости номера, недоступности линии, обнаружения SIT-тона или другого определяемого режима разрыва соединения. Вызов автоматически не разъединяется.

Синтаксис

```
<onCallFailure type="possibleFailureType" />
```

Атрибуты

- ❑ **type** — CallXML-браузер будет пытаться определить причину обрыва вызова и создать отчет. Это может быть полезно при определении того, как он обрабатывался. Тип, заданный в **<onCallFailure>**, будет сравниваться с действительным типом ошибки и в случае совпадения выполнять код из элемента **<onCallFailure>**.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<callxml>
  <block>
    <call value="pstn://14075551212"/>
    <onAnswer>
      <playAudio value="hello.wav"/>
    </onAnswer>
    <onCallFailure>
      ...
    </onCallFailure>
  </block>
</callxml>
```

Тег **<onError>**

Событие возникает при появлении любой ошибки. Это дает разработчику CallXML способ обработки условий ошибок, которые могут возникнуть при выполнении CallXML.

Синтаксис

```
<onError type="possibleErrorType" />
```

Атрибуты

- ❑ **type** — CallXML-браузер будет пытаться определить причину возникновения ошибки и создать отчет о ее возникновении в приложении. Этот тип, заданный в элементе **<onError>**, будет сравниваться с действитель-

ным типом ошибки и в случае совпадения выполнять код CallXML, содержащийся в элементе `<onError>`.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<callxml>
  <block>
    <recordAudio value="ftp://xyz.com/newmsg.wav"/>
    <onError type="ftp_login_failure">
      ...
    </onError>
  </block>
</callxml>
```

Тег `<onHangup/>`

Это событие возникает, когда сеанс определяет, что одна сторона повесила трубку. Типичным примером использования является выполнение кода обязательной очистки памяти.

Возникает в процессе выполнения: любой вызов или медиа-действие.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<!-- play audio. If the caller hangs up, run some cleanup code on
the web server -->
<callxml>
  <block>
    <playAudio value="info.wav"/>
    <onHangup>
      <goto value="cleanup.asp"/>
    </onHangup>
  </block>
</callxml>
```

Тег `<onMaxTime />`

Это событие возникает при вводе абонентом цифр, если абонент тратит больше времени, чем это позволяет атрибутом `maxTime`.

Возникает в процессе выполнения: `<getDigits>`, `<inputDigits>`.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<!-- record a message for up to one minute. If the caller speaks for
over a minute, let them know we only record one minute of the message -->
```

```
<callxml>
  <block>
    <recordAudio value="mailto:bob@xyz.com"
      maxTime="60s"/>
    <onMaxTime>
      <playAudio value="onlylmin.wav"/>
    </onMaxTime>
  </block>
</callxml>
```

Тег **<onMaxPages />**

Это событие возникает в процессе получения факсимильного сообщения, если номер получаемой страницы превышает атрибут `maxPages`.

Возникает в процессе выполнения: `<receiveFax>`.

Тег **<onMaxSilence />**

Событие возникает, когда приложение ожидает слишком долго между соседними нажатиями клавиш или обнаруживает слишком долгую паузу в конце сеанса записи.

Возникает в процессе выполнения: `<recordAudio>`, `<getDigits>`, `<inputDigits>`, `<inputAudio>`.

Тег **<onTermDigit />**

Синтаксис

```
<onTermDigit value="possibleTerminatingDigit" />
```

Атрибуты

- `value` — значение сравнивается со значением, введенным абонентом, для переключения по событию `<onTermDigit>`. Тип, заданный в элементе `<onTermDigit>`, будет сравниваться с нажатой цифрой, и в случае совпадения выполняется документ, содержащийся в элементе `<onTermDigit>`.

Правильными значениями являются 012356789*#, плюс специальные клавиши тоновых телефонов ABCD.

Возникает в процессе выполнения: `<play...>`, `<recordAudio>`, `<getDigits>`, `<inputDigits>`, `<conference>`.

Пример

```
<?xml version="1.0" encoding="UTF-8">
<!-- Пройграть поздравление для автоответчика. Если абонент нажимает *,
идти в меню администратора. Если решетка, или ничего, записать сообщение.
-- >
<callxml>
  <block>
    <playAudio value="leavemsg.wav"
      termDigits="*#" />
    <onTermDigit value="#">
      <goto value="adminmenu.jsp" />
    </onTermDigit>
    <onTermDigit value="*">
      <recordAudio value="mailto:bob@cyz.com" />
    </onTermDigit>
  </block>
</callxml>
```

Тег <onExternalEvent>

Это событие возникает, когда другой сеанс посылает внешнее событие текущему сеансу.

Синтаксис

```
<onExternalEvent value="possibleExternalEventName" />
```

Атрибуты

- ☐ **value** — это значение должно сравниваться с именем внешнего события, которое было послано, чтобы переключить событие <onExternalEvent>. Этот тип задан в элементе <onExternalEvent> и будет сравниваться с действительным именем внешнего события, и в случае совпадения, любой текст в формате CallXML, содержащийся в элементе <onExternalEvent>, будет выполняться.

Возникает в процессе выполнения: любое действие.

Тестирование и отладка**Тег <simline>**

Элемент позволяет разработчику писать информацию в системный лог-файл. Он может быть полезен при трассировке потока исполнения приложения или документировании процесса разработки приложения.

Синтаксис

```
<simline value="Any text for the system log"/>
```

Атрибуты

□ value — значение строки для печати в log-файл.

Возможные события: <onError>.

Пример приложения

Demo:

```
<code><?xml version="1.0" encoding="UTF-8" ?>
<callxml>
  <block>
    <text>Please enter your four digit badge number followed by the
star key.</text>
    <getDigits var="badge" maxTime="60s" termDigits="*"
clearDigits="TRUE" maxDigits="4" />
    <ontermdigit value="*">
      <goto value="badgeinfo.cfm" submit="*" method="get" />
    </ontermdigit>
    <onMaxDigits>
      <goto value="badgeinfo.cfm" submit="*" method="get" />
    </onMaxDigits>
  </block>
</callxml>
</code>
```

badgeinfo:

```
<code><?xml version="1.0" encoding="UTF-8" ?>
<cfparam name="URL.Badge" default="1001">
<cfquery name="getMsg" datasource="voxeo" dbtype="ODBC">
  SELECT Name, Badge, Position FROM Employees WHERE
Badge=#URL.Badge#
</cfquery>
<callxml>
  <block>
    <text>
      <cfoutput>
```

Hi, #getMsg.Name#. You have badge number #getMsg.Badge# and are employed as a #getMsg.Position#.

</cfoutput>

</text>

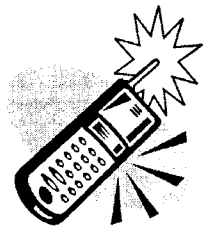
</block>

<hangup/>

</callxml>

</code>

Приложение 1



Особенности платформы Tellme

Отличия от стандарта VoiceXML 1.0

Сегодня VoiceXML является предлагаемой спецификацией рабочей группы W3C Voice Browser. Реализация этого языка на платформе Tellme в целом соответствует этой спецификации, поскольку эта компания является одним из шести авторов стандарта.

Платформа Tellme позволяет использовать расширенное подмножество спецификации VoiceXML 1.0, опубликованной организацией VoiceXML FORUM. Данное приложение показывает подробности этих отличий.

Интерпретация тегов

Этот раздел описывает отличия текущей реализации элементов языка VoiceXML для платформы Tellme. Нужно иметь в виду, что если в текстах приложений используются атрибуты, не поддерживаемые в текущей версии, то платформа Tellme будет их игнорировать.

Таблица П1.1. Отличия платформы Tellme от стандарта VoiceXML 1.0

Название	Коментарий
<assign>	Совместим с VoiceXML 1.0
<audio>	<p>Платформа Tellme интерпретирует текст непосредственно в элементах <prompt> и <block> в качестве текста TTS. В дальнейшем не требуется заключать текст TTS в теги <audio>. Тем не менее, рекомендуется включать текст тега <audio> для согласованности со стандартом. Атрибут <code>expr</code> является расширением стандарта на платформе Tellme</p> <p>Атрибуты, не поддерживаемые в текущей версии:</p> <ul style="list-style-type: none">• <code>caching</code> — игнорируется. Установка по умолчанию для этого атрибута <code>safe</code>. Не поддерживается <code>fast</code>

Таблица П1.1 (продолжение)

Название	Комментарий
<audio>	<ul style="list-style-type: none"> • <code>fetchtimeout</code> — игнорируется. Платформа Tellme исполняет глобальную установку времени тайм-аута HTTP-запроса как для аудиосодержания, так и для текстового содержания • <code>fetchhint</code> — игнорируется. Платформа Tellme автоматически извлекает до начала исполнения все исходные грамматики и JavaScript-файлы, указанные в документе VoiceXML
<block>	<p>Атрибуты, не поддерживаемые в текущей версии:</p> <ul style="list-style-type: none"> • <code>name</code> • <code>expr</code> • <code>cond</code>
<break>	Не поддерживается
<catch>	<p>Атрибут, не поддерживаемый в текущей версии:</p> <ul style="list-style-type: none"> • <code>cond</code>
<choice>	<p>Атрибуты, не поддерживаемые в текущей версии:</p> <ul style="list-style-type: none"> • <code>event</code> • <code>expr</code> • <code>caching</code> — игнорируется. Установка по умолчанию для этого атрибута <code>safe</code>. Не поддерживается <code>fast</code> • <code>fetchaudio</code> — игнорируется. Платформа Tellme проигрывает специфический для данной платформы звуковой файл до момента начала распознавания • <code>fetchhint</code> — игнорируется. Платформа Tellme автоматически извлекает до начала исполнения все исходные грамматики и JavaScript-файлы, указанные в документе VoiceXML • <code>fetchtimeout</code> — игнорируется. Платформа Tellme исполняет глобальную установку времени тайм-аута для HTTP-запроса как для звукового, так и для текстового содержания
<clear>	В настоящей версии элемент <prompt> с атрибутом <code>count</code> не поддерживается, так что <clear> неприменим к <prompt>
<confirm>	Расширение платформы Tellme

Таблица П1.1 (продолжение)

Название	Коментарий
<debug>	<p>Расширение платформы Tellme</p> <p>Элемент <debug> идентичен исполнению для <log> спецификации VoiceXML FORUM Version 1.0</p> <p>В настоящее время поддерживается также <log></p> <p>Рекомендуется использовать <log> вместо <debug></p>
<default>	Расширение платформы Tellme
<disconnect>	Совместим с VoiceXML 1.0
<div>	В текущей версии не поддерживается
<dtmf>	Поддерживается
<emp>	В текущей версии не поддерживается
<enumerate>	В текущей версии не поддерживается
<error>	Совместим с VoiceXML 1.0
<exit>	<p>Платформа Tellme исполняет <exit> как элемент возврата из <gosub>.</p> <p>Рекомендуется использовать <subdialog> в сочетании с <return>.</p> <p>Для того чтобы вернуть управление в главный диалог Tellme меню, нужно использовать <goto next="_home"/></p> <p>Для завершения телефонного вызова нужно использовать элемент <disconnect/>. В настоящее время неподдерживаемым является атрибут:</p> <ul style="list-style-type: none"> • namelist
<field>	<p>Атрибуты, не поддерживаемые в текущей версии:</p> <ul style="list-style-type: none"> • cond • type — можно ссылаться на внутреннюю грамматику внутри элемента <grammar> • slot
<filled>	Совместим с VoiceXML 1.0
<foreach>	Расширение платформы Tellme

Таблица П1.1 (продолжение)

Название	Комментарий
<form>	Интерпретация Tellme для алгоритма <form> не использует атрибуты <code>expr</code> и <code>cond</code> . Кроме того, форма не описывает "пункт назначения" передаваемой информации, то есть она будет передавать управление в следующую форму
<gosub>	Расширение платформы Tellme Элемент <gosub> аналогичен элементу <subdialog> спецификации VoiceXML FORUM Version 1.0. Рекомендуется использовать <subdialog> в сочетании с <return>
<goto>	Атрибуты, не поддерживаемые в текущей версии: <ul style="list-style-type: none"> • <code>entype</code> • <code>caching</code> — игнорируется. По умолчанию <code>safe</code>. Значение <code>fast</code> не поддерживается • <code>fetchaudio</code> — игнорируется. Платформа Tellme проигрывает специфический для данной платформы звуковой файл до момента начала распознавания • <code>fetchhint</code> — игнорируется. Платформа Tellme автоматически извлекает до начала исполнения все исходные грамматики и JavaScript-файлы, указанные в документе VoiceXML • <code>fetchtimeout</code> — игнорируется. Платформа Tellme исполняет глобальную установку времени тайм-аута HTTP-запроса как для аудио содержания, так и для текстового содержания <p>Атрибуты <code>method</code> и <code>submit</code> обеспечивают функциональность, аналогичную элементу <submit> стандартной спецификации VoiceXML.</p>
<grammar>	Атрибуты, не поддерживаемые в текущей версии: <ul style="list-style-type: none"> • <code>scope</code> — область действия грамматики зависит от ее родительского элемента • <code>caching</code> — игнорируется. Значение по умолчанию <code>safe</code>. Значение <code>fast</code> не поддерживается
<grammar>	<ul style="list-style-type: none"> • <code>fetchhint</code> — игнорируется. Платформа Tellme автоматически извлекает до начала исполнения все исходные грамматики и JavaScript-файлы, указанные в документе VoiceXML • <code>fetchtimeout</code> — игнорируется. Платформа Tellme исполняет глобальную установку времени тайм-аута HTTP-запроса как для аудио содержания, так и для текстового содержания

Таблица П1.1 (продолжение)

Название	Коментарий
<help>	Совместим с VoiceXML 1.0
<if><elseif/> <else/>	Совместим с VoiceXML 1.0
<initial>	<p>В текущей версии элемент <initial> ведет себя идентично элементу <block>.</p> <p>Атрибуты, не поддерживаемые в текущей версии:</p> <ul style="list-style-type: none"> • name • expr • cond
<link>	<p>Атрибуты, не поддерживаемые в текущей версии:</p> <ul style="list-style-type: none"> • fetchaudio — игнорируется. Платформа Tellme проигрывает специфический для данной платформы звуковой файл до момента начала распознавания • fetchhint • fetchtimeout — игнорируется. Платформа Tellme исполняет глобальную установку времени тайм-аута HTTP-запроса как для аудио содержания, так и для текстового содержания
<listen>	Расширение платформы Tellme
<menu>	<p>Грамматики для меню имеют область действия в рамках конкретного меню. Это означает, что они доступны только в текущем меню.</p> <p>Атрибут, не поддерживаемый в текущей версии:</p> <ul style="list-style-type: none"> • scope
<meta>	<p>Атрибут, не поддерживаемый в текущей версии:</p> <ul style="list-style-type: none"> • http-equiv
<noinput>	Совместим с VoiceXML 1.0
<nomatch>	Совместим с VoiceXML 1.0
<object>	В текущей версии не поддерживается
<option>	В текущей версии не поддерживается
<param>	<p>Атрибуты, не поддерживаемые в текущей версии:</p> <ul style="list-style-type: none"> • value • valuetype

Таблица П1.1 (продолжение)

Название	Комментарий
<pause>	Расширение Tellme
<prompt>	Атрибуты, не поддерживаемые в текущей версии: <ul style="list-style-type: none"> • baregin • cond • count • timeout
<property>	Атрибуты, не поддерживаемые в текущей версии: <ul style="list-style-type: none"> • sensitivity • speedvsaccuracy • completetimeout • incompletetimeout • termtimeout • termchar • caching • audiofetchhint • documentfetchhint • objectfetchhint • scriptfetchhint • fetchaudio • fetchtimeout • inputmodes <p>Необходимо отметить, что многие ключевые свойства процесса распознавания речи доступны посредством использования атрибутов <field>, а не параметров элемента <property></p>
<pros>	В текущей версии не поддерживается
<record>	Атрибуты, не поддерживаемые в текущей версии: <ul style="list-style-type: none"> • beep • cond • expr • modal • type

Таблица П1.1 (продолжение)

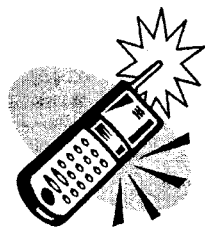
Название	Коментарий
<reprompt>	Совместим с VoiceXML 1.0 Атрибут order является расширением Tellme
<result>	Расширение платформы Tellme
<return>	Элементы <subdialog> и <return> аналогичны исполнению <gosub> и <exit>. В текущей версии поддерживается сочетание элементов <subdialog> и <return>, которые рекомендуется использовать
<sayas>	В текущей версии не поддерживается
<script>	Атрибуты, не поддерживаемые в текущей версии: <ul style="list-style-type: none"> • charset • caching — игнорируется. По умолчанию установлен в safe. Значение fast не поддерживается • fetchhint — игнорируется. Платформа Tellme автоматически извлекает до начала исполнения все исходные грамматики и JavaScript-файлы, указанные в документе VoiceXML • fetchtimeout — игнорируется. Платформа Tellme исполняет глобальные установки тайм-аута как для извлечения звука, так и извлечения текстового содержания
<subdialog>	Элементы <subdialog> и <return> аналогичны исполнению <gosub> и <exit> в платформе Tellme. Атрибуты, не поддерживаемые в текущей версии: <ul style="list-style-type: none"> • expr • cond
<subdialog>	<ul style="list-style-type: none"> • enctype • caching • fetchaudio • fetchtimeout • fetchhint
<submit>	Атрибуты, не поддерживаемые в текущей версии: <ul style="list-style-type: none"> • enctype • caching • fetchaudio • fetchhin • fetchtimeout

Таблица П1.1 (окончание)

Название	Комментарий
<throw>	Совместим с VoiceXML 1.0
<transfer>	Атрибуты, не поддерживаемые в текущей версии: <ul style="list-style-type: none">• bridge — значение атрибута, устанавливаемое по умолчанию (true). Значение false не поддерживается• name• expr• cond• destexpr• connecttimeout — эта функциональность поддерживается посредством атрибута timeout• maxtime — эта функциональность поддерживается с помощью атрибута maxlength• timeout
<value>	Совместим с VoiceXML 1.0
<var>	Совместим с VoiceXML 1.0
<vxml>	Совместим с VoiceXML 1.0

Приложение 2

Особенности платформы Nuance



В табл. П2.1—П2.3 описываются поддерживаемые и неподдерживаемые элементы стандартного языка VoiceXML версии 1.0.

Таблица П2.1. Поддерживаемые элементы языка VoiceXML версии 1.0

Элемент	Поддерживаемые атрибуты	Неподдерживаемые атрибуты	Расширения
fssign	name, expr		
fudio	src, expr	fetchtimeout, fetchhint, expr caching	
block	cond, expr, name		
break	msecs, size		
catch	event, cond, count		
clear	namelist		
else			
elseif	cond		
error	count, cond, type		type
exit		expr, namelist	
form	id	scope	
field	name, cond, expr, slot	type, modal	
filled	mode, namelist		
goto	next, expr, nextitem, expritem	caching, fetchaudio, fetchhint, fetchtimeout	
grammar	type (GSL Only), src	scope, caching fetchint, fetchtimeout	
help	count, cond		
if	cond		
link	expr, next, event	caching, fetchaudio, fetchhint, fetchtimeout, enctype	

Таблица П2.1 (окончание)

Элемент	Поддерживаемые атрибуты	Неподдерживаемые атрибуты	Расширения
meta	name, content	http-equiv	
noinput	count, cond		
nomatch	count, cond		
object	name, cond, expr, classid, codebase, data, archive	caching, fetchaudio, fetchhint, fetchtimeout, type, codetype	
param	name, type, expr	value, valuetype	index
prompt	count, bargein	cond, timeout	
property	name, value		
reprompt			
script		src, caching, fetchhint, fetchtimeout	
submit	next, expr, namelist, method, enctype, caching	fetchaudio, fetchhint, fetchtimeout	
throw	event		
value	expr	class, mode, recsrc	
var	name, expr		
vxml	application, version	base, lang	

Таблица П2.2. Неподдерживаемые элементы языка VoiceXML версии 1.0

Элемент	Замечание
choice	
disconnect	Вместо этого можно использовать <exit/>
div	
dtmf	Можно определять коды DTMF созданием грамматики для них
enumerate	
emp	
initial	
menu	
option	
pros	

Таблица П2.2 (окончание)

Элемент	Замечание
record	
return	
sayas	
subdialog	
transfer	Можно делать <goto> с указанием номера телефона 18315551212

Таблица П2.3. Дополнения в спецификацию VoiceXML

Элемент	Атрибуты	Замечания
debug	expr, level	Пишет отладочный log-файл

Приложение 3



Преобразование из WML 1.3 в WML 2.0

В этом приложении будет представлена таблица стилей XSLT 1.1 [XSLT], которая определяет все правила преобразования спецификации WML 2.0.

Таблица стилей организована следующим образом. Существует один темплет для каждого элемента WML 1.3 из исходного документа. Этот темплет создает соответствующий элемент WML 2.0 результирующего документа и включает в него дополнительные атрибуты. Такой подход применим ко всем типам элементов языка WML 1.3.

Поскольку для того чтобы преобразовать элемент `<do>` языка WML 1.3, должен быть разработан соответствующий алгоритм, преобразование этого элемента является наиболее сложной задачей, которая выполняется в двух темплетах.

Так как результирующий документ является корректным документом WML 2.0 с пространствами имен из XHTML и WML и также может быть частью более крупного WAP сервиса, то атрибут `use-xml-fragment` (WML 2.0) установлен в `false`. Это позволяет WML 2.0 объявлять тип документа до корневого элемента.

Таблица преобразования использует произвольный Java class (`org.wapforum.wml.WMLExpression`) для того, чтобы извлечь WML-переменные из текста.

Таблица преобразования XSLT

Приведенная таблица преобразования доступна по адресу:
[http:// www.wapfourm.org/xslt/wap-244-wmltr.xsl](http://www.wapfourm.org/xslt/wap-244-wmltr.xsl).

```
<!--Приведенная таблица XSLT преобразует корректные WML 1.3 и WTA-  
WML 1.2 документы в правильный документ WML 2.0. Для исполнения  
необходима поддержка XSLT версии 1.1.
```

```
http://www.w3.org/TR/xslt11/-->
```

```
<xsl:transform
```

```
    version="1.1"
```

```
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
xmlns:wml="http://www.wapforum.org/2001/wml"
xmlns:wml:expr="http://www.wapforum.org/2001/wml-expression"
exclude-result-prefixes="wml:expr"
xmlns="http://www.w3.org/1999/xhtml">

<xsl:script
  implements-prefix="wml:expr"
  language="java"
  src="java:org.wapforum.wml.WMLExpression"/>
<xsl:template match="select">
  <select>
    <xsl:copy-of select="@id | @class | @xml:lang | @title | @name |
@tabindex" />
    <xsl:if test="boolean(@multiple='true')">
      <xsl:attribute name="multiple">multiple</xsl:attribute>
    </xsl:if>
    <xsl:if test="boolean(@iname)" >
      <xsl:attribute name="wml:iname">
        <xsl:value-of select="@iname" />
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="boolean(@ivalue)" >
      <xsl:attribute name="wml:ivalue">
        <xsl:value-of select="@ivalue" />
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="boolean(@value)" >
      <xsl:attribute name="wml:value">
        <xsl:value-of select="@value" />
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="boolean(@name)" >
      <xsl:attribute name="wml:name">
        <xsl:value-of select="@name" />
      </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates select="option|optgroup" />
  </select>
```

```

</xsl:template>
<xsl:template match="optgroup">
  <optgroup>
    <xsl:copy-of select="@id | @class | @title | @xml:lang" />
    <xsl:attribute name="label"> </xsl:attribute>

<!--NOTE: Nested optgroup elements are ignored-->

    <xsl:apply-templates select="./option" />
  </optgroup>
</xsl:template>
<xsl:template match="option" >
  <option>
    <xsl:copy-of select="@id | @class | @xml:lang | @title | @value"
/>
    <xsl:if test="boolean(@onpick)" >
      <xsl:attribute name="wml:onpick">
        <xsl:value-of select="@onpick" />
      </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates select="text() | onevent" />
  </option>
</xsl:template>
<xsl:template match="input" >
  <input>
    <xsl:copy-of select="@id | @class | @xml:lang | @title | @type |
@size | @maxlength |
@tabindex | @value | @accesskey" />
    <xsl:if test="boolean(@name)" >
      <xsl:attribute name="wml:name">
        <xsl:value-of select="@name" />
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="boolean(@format)" >
      <xsl:attribute name="wml:format">
        <xsl:value-of select="@format" />
      </xsl:attribute>

```

```

</xsl:if>
<xsl:if test="boolean(@emptyok) " >
  <xsl:attribute name="wml:emptyok">
    <xsl:value-of select="@emptyok" />
  </xsl:attribute>
</xsl:if>
</input>
</xsl:template>
<xsl:template match="fieldset" >
  <fieldset>
    <xsl:copy-of select="@id | @class | @xml:lang | @title" />
    <xsl:apply-templates
select="do|input|select|fieldset|text()|em|strong|b|i|u|big|small|br
|img|anchor|a|table" />
  </fieldset>
</xsl:template>
<xsl:template match="img">
  <img>
    <xsl:copy-of select="@id | @class | @xml:lang | @height | @width |
@align | @hspace |
@vspace | @alt | @src" />
    <xsl:if test="boolean(@localsrc)" >
      <xsl:attribute name="wml:localsrc">
        <xsl:value-of select="@localsrc" />
      </xsl:attribute>
    </xsl:if>
  </img>
</xsl:template>
<xsl:template match="a">
  <a>
    <xsl:copy-of select="@*" />
    <xsl:apply-templates select="text() | br | img" />
  </a>
</xsl:template>
<xsl:template match="anchor">
  <wml:anchor>
    <xsl:copy-of select="@id | @class | @xml:lang | @accesskey |
@title" />

```

```

<xsl:apply-templates select="text() | br | img | go | prev | re-
fresh" />
</wml:anchor>
</xsl:template>
<xsl:template match="table">
  <table>
    <xsl:copy-of select="@id | @class | @xml:lang | @title" />
    <xsl:if test="boolean(@columns)" >
      <xsl:attribute name="wml:columns" >
        <xsl:value-of select="@columns" />
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="boolean(@align)" >
      <xsl:attribute name="wml:align">
        <xsl:value-of select="@align" />
      </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates select="tr" />
  </table>
</xsl:template>
<xsl:template match="tr">
  <tr>
    <xsl:copy-of select="@id | @class" />
    <xsl:apply-templates select="td" />
  </tr>
</xsl:template>
<xsl:template match="td" >
  <td>
    <xsl:copy-of select="@*" />
    <xsl:apply-templates se-
lect="text() |br|em|strong|b|i|u|big|small|img|anchor|a " />
  </td>
</xsl:template>
<xsl:template match="b">
  <b>
    <xsl:copy-of select="@*" />
    <xsl:apply-templates select="text() | em | strong |b |i |u |big
|small | br | img | anchor |a

```

```

|table" />
</b>
</xsl:template>
<xsl:template match="u">
  <u>
    <xsl:copy-of select="@*" />
    <xsl:apply-templates select="text() | em | strong |b |i |u |big
|small | br | img | anchor |a
|table" />
  </u>
</xsl:template>
<xsl:template match="i">
  <i>
    <xsl:copy-of select="@*" />
    <xsl:apply-templates select="text() | em | strong |b |i |u |big
|small | br | img | anchor |a
|table" />
  </i>
</xsl:template>
<xsl:template match="big">
  <big>
    <xsl:copy-of select="@*" />
    <xsl:apply-templates select="text() | em | strong |b |i |u |big
|small | br | img | anchor |a
|table" />
  </big>
</xsl:template>
<xsl:template match="small">
  <small>
    <xsl:copy-of select="@*" />
    <xsl:apply-templates select="text() | em | strong |b |i |u |big
|small | br | img | anchor |a
|table" />
  </small>
</xsl:template>
<xsl:template match="em">
  <em>
    <xsl:copy-of select="@*" />

```

```

    <xsl:apply-templates select="text() | em | strong |b |i |u |big
|small | br | img | anchor |a
|table" />
  </em>
</xsl:template>
<xsl:template match="strong">
  <strong>
    <xsl:copy-of select="@*" />
    <xsl:apply-templates select="text() | em | strong |b |i |u |big
|small | br | img | anchor |a
|table" />
  </strong>
</xsl:template>
<xsl:template match="br">
  <br>
  <xsl:copy-of select="@*" />
</br>
</xsl:template>
<xsl:template match="pre">
  <pre>
    <xsl:copy-of select="@id | @class"/>
    <xsl:apply-templates select="text() | a | anchor | do | u | br | i
| b | em | strong | input |
select" />
  </pre>
</xsl:template>
<xsl:template match="p">
  <p>
    <xsl:copy-of select="@id | @class | @xml:lang " />
    <xsl:if test="boolean(@mode)" >
      <xsl:attribute name="wml:mode">
        <xsl:value-of select="@mode" />
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="boolean(@align)" >
      <xsl:attribute name="align">
        <xsl:value-of select="@align" />
      </xsl:attribute>

```

```

</xsl:if>
<xsl:apply-templates
select="do|input|select|fieldset|text()|em|strong|b|i|u|big|small|br
|img|anchor|a|table" />
</p>
</xsl:template>
<xsl:template match="onevent" >
  <wml:onevent>
    <xsl:choose>
      <xsl:when test="starts-with(@type, 'on')">
        <xsl:attribute name="type"><xsl:value-of select="substring-
after(@type, 'on')"/>
        </xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
        <xsl:attribute name="type"><xsl:value-of select="@type" />
        </xsl:attribute>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:copy-of select="@id.| @class" />
    <xsl:apply-templates select="prev | noop | refresh | go" />
  </wml:onevent>
</xsl:template>
<xsl:template name="DO">
  <wml:do>
    <xsl:choose>
      <xsl:when test="@type='accept'">
        <xsl:attribute name="role">positive
      </xsl:attribute>
      </xsl:when>
      <xsl:when test="@type='prev' ">
        <xsl:attribute name="role">back
      </xsl:attribute>
      </xsl:when>
      <xsl:when test="@type='help'">
        <xsl:attribute name="role">help
      </xsl:attribute>
      </xsl:when>

```

```

<xsl:when test="@type='options'">
  <xsl:attribute name="role">options
</xsl:attribute>
</xsl:when>
<xsl:when test="@type='reset' or @type='delete'">
  <xsl:attribute name="role">negative
</xsl:attribute>
</xsl:when>
<xsl:otherwise>
  <xsl:attribute name="role"><xsl:value-of select="@type"/>
</xsl:attribute>
</xsl:otherwise>
</xsl:choose>
<xsl:copy-of select="@id | @class | @xml:lang" />
<xsl:if test="@label">
  <xsl:element name="wml:widget">
    <xsl:choose>
      <xsl:when test="function-available('wml:expr:WMLExpression')">
        <xsl:value-of select="wml:expr:WMLExpression(string(@label))"
disable-output-escaping="yes" />
      </xsl:when>
      <xsl:otherwise>
        <xsl:copy-of select="." />
      </xsl:otherwise>
    </xsl:choose>
  </xsl:element>
</xsl:if>
<xsl:apply-templates select="prev|noop|go|refresh" />
</wml:do>
</xsl:template>
<xsl:template match="do" >
  <xsl:param name="shadow" select="' '"/>
  <xsl:choose>
    <xsl:when test="@name">
      <xsl:if test="not(contains(string($shadow), @name) )">
        <xsl:call-template name="DO" />
      </xsl:if>

```

```

</xsl:when>
<xsl:otherwise>
  <xsl:if test="not(contains(string($shadow), @type) )">
    <xsl:call-template name="DO" />
  </xsl:if>
</xsl:otherwise>
</xsl:choose>
</xsl:template>
<xsl:template match="timer">
  <wml:timer>
    <xsl:copy-of select="@*" />
  </wml:timer>
</xsl:template>
<xsl:template match="meta">
  <meta>
    <xsl:copy-of select="@http-equiv | @name | @content | @scheme" />
    <xsl:if test="boolean(@forua)" >
      <xsl:attribute name="wml:forua">
        <xsl:value-of select="@forua" />
      </xsl:attribute>
    </xsl:if>
  </meta>
</xsl:template>
<xsl:template match="access">
  <wml:access>
    <xsl:copy-of select="@*" />
  </wml:access>
</xsl:template>
<xsl:template match="wml | wta-wml">
  <html wml:use-xml-fragments="false">
    <xsl:call-template name="Template.events" />

<!-- NOTE: Игнорировать id и class в элементах html, head и tem-
plate -->

  <head>

<!-- NOTE: Использовать заголовок первой карты в качестве названия
документа -->

```

```

<title>
  <xsl:value-of select="card[position()=1]/@title" />
</title>
<xsl:apply-templates select="head/access | head/meta" />
</head>
<xsl:apply-templates select="template/onevent | card" />
</html>
</xsl:template>
<xsl:template match="card">
  <wml:card>
    <xsl:copy-of select="@id | @class | @xml:lang | @title |
@onenterforward |
@onenterbackward | @ontimer | @newcontext"/>
    <xsl:apply-templates select="onevent | timer | p | pre | do" />
    <xsl:if test="not(p | pre)">
      <p/>
    </xsl:if>
<!-- Установить не отработанные DO элементы
создать простансвто имен (либо name либо тип атрибута)
-->

    <xsl:apply-templates select="//template/do" >
    <xsl:with-param name="shadow">
    <xsl:for-each select="do">
    <xsl:choose>
      <xsl:when test="@name">
        <xsl:value-of select="@name" />
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="@type" />
      </xsl:otherwise>
    </xsl:choose>
    <xsl:text>

</xsl:text>
</xsl:for-each>
</xsl:with-param>

```

```

    </xsl:apply-templates>
  </wml:card>
</xsl:template>
<xsl:template name="Template.events">
  <xsl:if test="template/@ontimer">
    <xsl:attribute name="wml:ontimer">
      <xsl:value-of select="template/@ontimer"/>
    </xsl:attribute>
  </xsl:if>
  <xsl:if test="template/@onenterforward">
    <xsl:attribute name="wml:onenterforward">
      <xsl:value-of select="template/@onenterforward"/>
    </xsl:attribute>
  </xsl:if>
  <xsl:if test="template/@onenterbackward">
    <xsl:attribute name="wml:onenterbackward">
      <xsl:value-of select="template/@onenterbackward"/>
    </xsl:attribute>
  </xsl:if>
</xsl:template>
<xsl:template match="go">
  <wml:go>
    <xsl:copy-of select="@id | @class | @href | @sendreferer | @method
| @cache-control
| @accept-charset" />
    <xsl:choose>
      <xsl:when test="@enctype='multipart/form-data'">
        <xsl:attribute
name="enctype">application/vnd.wap.wml.form.data</xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
        <xsl:attribute
name="enctype">application/vnd.wap.wml.form.urlencoded</xsl:attribute>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:apply-templates select="postfield | setvar" />
  </wml:go>
</xsl:template>

```

```
<xsl:template match="postfield">
  <wml:postfield>
    <xsl:copy-of select="@*" />
  </wml:postfield>
</xsl:template>
<xsl:template match="prev">
  <wml:prev>
    <xsl:copy-of select="@*" />
    <xsl:apply-templates select="setvar" />
  </wml:prev>
</xsl:template>
<xsl:template match="noop">
  <wml:noop>
    <xsl:copy-of select="@*" />
  </wml:noop>
</xsl:template>
<xsl:template match="refresh">
  <wml:refresh>
    <xsl:copy-of select="@*" />
    <xsl:apply-templates select="setvar" />
  </wml:refresh>
</xsl:template>
<xsl:template match="setvar">
  <wml:setvar>
    <xsl:copy-of select="@*" />
  </wml:setvar>
</xsl:template>
<xsl:template match="text()">
  <xsl:choose>
    <xsl:when test="function-available('wml:expr:WMLExpression')">
      <xsl:value-of select="wml:expr:WMLExpression(string())" disable-
output-escaping="yes" />
    </xsl:when>
    <xsl:otherwise>
      <xsl:copy-of select="." />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

```
<xsl:output
  method="xml"
  indent="yes"
  doctype-system="wml20.dtd"
  doctype-public="-//WAPFORUM//DTD WML 2.0//EN"
  media-type="application/wml+xml"/>
</xsl:transform>
```

Использованные ресурсы и литература

1. Bray Tim and Paoli Jean and C.M.Sperberg-McQueen. Extensible Markup Language (XML). W3C Recommendation February 10, 1998, REC-xml-19980210 February 10, 1998. URL: <http://www.w3.org/TR/1998/REC-xml-19980210>
2. CallXML description. URL: <http://community.voxeo.com>.
3. Faynberg Igor, Gaurgde Lawrence, Hui-Lan Lu. Convergent Networks and Servers: Internet marking and PSTN. Wiley Computer Publishing, 2000.
4. King P. et al. Handheld Device Markup Language Specification. April 11, 1997.
5. Nuance Voice Web Server, Version 1.0, Voice Site Developer's Guide. URL: <http://www.nuance.com>.
6. Tellme VoiceXML Reference, Programming for the Tellme Platform, URL: <http://www.tellme.com>.
7. The Unicode Consortium, The Unicode Standard: Version 2.0, Addison-Wesley Developers Press, 1996. URL: <http://www.unicode.org/>.
8. Voice eXtensible Markup Language, VoiceXML Version: 1.00, March 7, 2000. URL: <http://www.voicexml.org>.
9. WAP FORUM. Wireless Application Environment Specification. February 19, 2000. URL: <http://www.wapforum.org/>.
10. WAP FORUM. Wireless Application Protocol Architecture Specification. April 30, 1998. URL <http://www.wapforum.org/>.
11. WAP FORUM. Wireless Markup Language Specification. February 19, 2000. URL: <http://www.wapforum.org/>.
12. WAP FORUM. Wireless Telephony Application Interface Specification. November 8, 1999. URL: <http://www.wapforum.org/>.
13. WAP FORUM. WMLScript Language Specification. March 24, 2000. URL: <http://www.wapforum.org/>.
14. WAP FORUM. WMLScript Standart Libraries Specification. March 24, 2000. URL: <http://www.wapforum.org/>.
15. Русеев С.П. WAP: технология и приложения. — СПб: БХВ-Петербург, 2001.