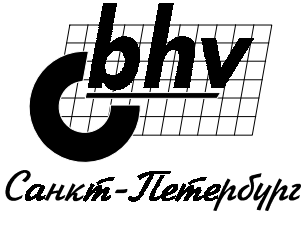
Александр Матросов
Александр Сергеев
Михаил Чаунин

HTML 4.0



Дюссельдорф ◆ Киев ◆ Москва ◆ Санкт-Петербург

УДК 681.3.06

Представлен весь спектр технологий создания Web-документов (начиная от простейших — статических — и до документов на основе динамического HTML), включая форматирование текста, создание списков, таблиц, форм, применение графики, каскадных таблиц стилей, встраивание различных объектов, использование средств интерактивного общения с пользователем, баз данных, мультимедиа-объектов и пр. Рассматриваются объектно-ориентированные технологии и программирование на языке Perl, а также создание CGI-программ и написание сценариев на языках JavaScript и VBScript.

Приводятся сведения о браузерах Netscape Communicator и Microsoft Internet Explorer и таблице HTML-тэгов.

Для Web-дизайнеров

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зав. редакцией	*Наталья Таркова*
Ответственный редактор	*Борис Желваков*
Компьютерная верстка	*Ольга Сергиенко*
Корректура	*Зинаида Дмитриева*
Дизайн обложки	*Наталья Смирнова*
Производство	*Николай Тверских*

Матросов А. В., Сергеев А. О., Чаунин М. П.
HTML 4.0. — СПб.: БХВ — Санкт-Петербург, 1999. — 672 с.: ил.

ISBN 5-8206-0072-X

© Матросов А. В., Сергеев А. О., Чаунин М. П., 1999
© Оформление, издательство "БХВ — Санкт-Петербург", 1999

Лицензия ЛР № 065953 от 15.06.98. Подписано в печать 12.11.99.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 54,18.
Тираж 3000 экз. Заказ
"БХВ — Санкт-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

ЧАСТЬ I. ОСНОВЫ HTML 11

Глава 1. Правила построения HTML-документов 13

Что такое HTML 13
Спецификации HTML 16
Структура документа 18
Раздел документа HEAD 19
Название документа 19
Связь с другими документами 20
Тэг *<META>* 22
Другие элементы заголовка 24
Раздел документа BODY 24
Форматирование текста 25
Тэги уровня блока и последовательные тэги 25
Логическое и физическое форматирование 26
Тэги логического форматирования текста 27
Тэги физического форматирования текста 31
Форматирование HTML-документа 39
Разделение на абзацы 39
Перевод строки 40
Тэги *<NOBR>* и *<WBR>* 41
Заголовки внутри HTML-документа 42
Горизонтальные линии 43
Использование предварительно отформатированного текста 44
Тэг *<DIV>* 45
Тэг *<CENTER>* 46
Включение комментариев в документ 46
Тэг *<BLOCKQUOTE>* 46
Тэг *<ADDRESS>* 48
Специальные символы 48
Ссылки на другие документы и файлы 49
Организация ссылок 49
Правила записи ссылок 50
Прочие тэги 54

Глава 2. Списки ... 56

Маркированный список ... 56
Тэги <*UL*> и <*LI*> ... 57
Графические маркеры списка ... 59
Нумерованный список ... 61
Тэги <*OL*> и <*LI*> ... 61
Список определений ... 65
Списки типа <*DIR*> и <*MENU*> ... 67
Вложенные списки ... 68

Глава 3. Графика ... 71

Общие соображения ... 72
Способы хранения изображений ... 73
Фоновые изображения ... 75
Встраивание изображений в HTML-документы ... 78
Выравнивание изображений ... 78
Задание размеров выводимого изображения ... 83
Отделение изображения от текста ... 84
Рамки вокруг изображений ... 85
Альтернативный текст ... 86
Использование изображения в качестве ссылки ... 87
Параметр *LOWSRC* ... 88
Использование миниатюрных версий изображений ... 88
Формат GIF ... 90
Формат JPG ... 96
Какой формат предпочесть — GIF или JPG ... 97
Некоторые проблемы использования цвета ... 99
Создание анимации на основе GIF-файлов ... 100
Программа GIF Construction Set ... 102
Блок HEADER ... 103
Блок LOOP ... 104
Блок CONTROL ... 104
Блок IMAGE ... 107
Использование мастера анимации ... 108
Дополнительные возможности пакета GIF Construction Set ... 110

Глава 4. Таблицы в HTML ... 114

Создание простейших HTML-таблиц ... 116
Представление таблиц на странице ... 118
Заголовок таблицы <*CAPTION*> ... 118
Параметры тэга <*TABLE*> ... 119
Параметр *BORDER* ... 120
Параметр *CELLSPACING* ... 121
Параметр *CELLPADDING* ... 122
Параметры *WIDTH* и *HEIGHT* ... 123
Параметр *ALIGN* ... 124
Форматирование данных внутри таблицы ... 128

Вложенные таблицы ... 132
Особенности построения таблиц ... 136
Отображение пустых ячеек в таблицах ... 136
Выравнивание данных в столбцах таблицы ... 137
Задание цвета рамок таблицы ... 139
Задание фонового рисунка для таблицы ... 140
Тэги структурирования таблицы *<THEAD>*, *<TBODY>* и *<TFOOT>* ... 140
Задание числа столбцов таблицы ... 143
Вертикальное выравнивание таблиц ... 143
Альтернатива табличному представлению ... 144
Подготовка таблиц ... 144

Глава 5. Фреймы ... 148

Сферы применения фреймов ... 148
Правила описания фреймов ... 155
Тэг *<FRAMESET>* ... 157
Тэг *<FRAME>* ... 159
Тэг *<NOFRAMES>* ... 162
Особенности описания фреймовых структур ... 162
Примеры фреймов ... 164
Особенности навигации при использовании фреймов ... 167
Взаимодействие между фреймами ... 168
Примеры более сложного взаимодействия между фреймами ... 174
Различие между фреймами и окнами браузера ... 179
Дополнительные возможности браузеров ... 181
Возможности браузера Netscape ... 181
Возможности браузера Microsoft Internet Explorer ... 183
Плавающие фреймы ... 185
Средства создания документов, содержащих фреймы ... 186
Редактор фреймов FrameGang ... 187
Редактор фреймов Frame-It ... 190
Информация об использовании фреймов на WWW ... 193

Глава 6. Карты-изображения ... 194

Основы использования карт-изображений ... 196
Терминология ... 197
Графическое представление карты-изображения ... 197
Описание конфигурации карты-изображения ... 197
Варианты реализации карт-изображений ... 198
Преимущества и недостатки карт-изображений ... 198
Серверный вариант реализации карт-изображений ... 200
Формат CERN ... 202
Формат NCSA ... 203
Клиентский вариант карты-изображения ... 204
Тэг *<MAP>* ... 205
Тэг *<AREA>* ... 205
Параметр *SHAPE* ... 205

Параметр *COORDS* ... 206
Параметры *HREF* и *NOHREF* ... 206
Параметр *TARGET* ... 207
Параметр *ALT* ... 207
Комбинация клиентского и серверного вариантов ... 208
Особенности использования карт-изображений ... 209
Альтернативные средства навигации ... 209
Средства создания карт-изображений ... 211
Программа MapEdit ... 212
Программа Map THIS! ... 217
Программа CrossEye ... 220
Заключительный пример ... 221

Глава 7. Звук ... 223

Средства воспроизведения звука ... 223
Как компьютер работает со звуком ... 224
Модуль LiveAudio ... 226
Управление модулем LiveAudio ... 226
Встраивание звуковых файлов в Web-страницу ... 227
Другие звуковые модули ... 232
Технология RealAudio ... 235
Программа-плейер RealPlayer Plus G2 ... 238
Встраивание в страницу звуковых файлов формата RealAudio ... 239
Ресурсы RealAudio в Интернете ... 246
Звуковые файлы формата MP3 ... 249
Воспроизведение файлов формата MP3 ... 251
Установка программы Winamp ... 252
Управление программой Winamp ... 256
Подключаемые модули программы Winamp ... 257
Изменение внешнего вида программы Winamp ... 259
Скрытые возможности программы Winamp ... 261
Декодирование файлов формата MP3 ... 262
Где найти файлы MP3? ... 263
Создание звуковых файлов MP3 ... 265
Программы вычленения файлов с аудио CD ... 266
Программы кодирования ... 266
Выбор параметров кодирования ... 270
Переносные плейеры звуковых файлов MP3 ... 271
Потоковое воспроизведение ... 274
Звуковые файлы формата VQF ... 274
Звуковые файлы формата AAC ... 277
HomeboyAAC ... 278
AT&T a2bAAC ... 278
Liquifier Pro AAC ... 279
Astrid/Quartex AAC ... 279
Звуковые файлы формата PAC ... 280
Новый формат хранения звука MP4 ... 281
Формат WMA ... 282

Глава 8. Разработка HTML-страниц при помощи текстового процессора Microsoft Word 283

Создание Web-страниц 285
Создание маркированных и нумерованных списков на Web-страницах 285
Вставка горизонтальной линии в Web-страницу 288
Выбор фона создаваемого документа 289
Изменение цвета и форматирование текста Web-страниц 291
Работа со стилями на Web-страницах 294
Непосредственное редактирование HTML-кода 295
Предварительный просмотр Web-страницы в процессе редактирования 296
Таблицы на Web-страницах 296
Работа с рисунками на Web-страницах 299
Создание ссылок в документе 301
Создание форм на Web-страницах 302
Сохранение существующего документа Word в формате HTML 303
Проблемы преобразования полей 308
Использование примечаний в документе 314
Версии и реализации Microsoft Word 315

ЧАСТЬ II. ИНТЕРАКТИВНЫЕ WEB-ДОКУМЕНТЫ 319

Глава 9. Выполняемые сценарии 321

Основы объектно-ориентированных технологий 321
- Что такое программный объект 321
- Событийные приложения 323
- Объектные модели языков сценариев 324

Язык создания сценариев JavaScript 325
- Общий обзор языка 326
  - Синтаксис языка 327
  - Размещение операторов языка на странице 327
  - Использование тэга *<SCRIPT>* 327
  - Задание файла с кодом JavaScript 328
  - Элементы JavaScript в параметрах тэгов HTML 329
  - Обработчики событий 329
- Язык ядра JavaScript 333
  - Переменные и литералы 333
  - Выражения и операторы 335
  - Стандартные объекты и функции 339
  - Операторы управления 345
- Объекты клиента и обработка событий 350
  - Иерархия объектов 351
  - Свойства и методы ключевых объектов 352
  - Обработчики событий 359
- Практические примеры 364
  - Часы JavaScript 364
  - Простое меню 365
  - Документ с фреймами 368

Язык VBScript ... 369
Основные понятия ... 370
Типы данных ... 370
Переменные, массивы и константы ... 373
Операторы ... 375
Операторы условия и цикла ... 377
Процедуры ... 383
Объектная модель и взаимодействие с элементами документа ... 385
Функции и объекты ядра VBScript ... 385
Объекты MS Internet Explorer ... 391
Процедуры обработки событий ... 393
Практические примеры ... 396
Плавающий фрейм ... 396
Баннер ... 398
CGI-сценарии и язык Perl ... 401
Основные понятия ... 401
HTML-формы ... 402
Тэг *<FORM>* ... 403
Тэг *<INPUT>* ... 404
Тэг *<SELECT>* ... 407
Тэг *<TEXTAREA>* ... 408
Пример формы ... 409
Передача информации CGI-программе ... 411
Кодирование и пересылка данных формы ... 411
CGI-сценарии ... 414
Общие сведения ... 414
Переменные в языке Perl ... 418
Переменные среды CGI ... 426
Поиск и замена текста. Регулярные выражения ... 429
Обработка данных формы ... 432
Подпрограммы, библиотеки, модули ... 438

Глава 10. Динамический HTML ... 449

Каскадные таблицы стилей ... 450
Общие положения ... 451
Встраивание таблиц стилей в документ ... 452
Группирование и наследование ... 454
Селекторы ... 455
Селектор *CLASS* ... 456
Селектор *ID* ... 457
Контекстные селекторы ... 457
Псевдоклассы ... 458
Псевдоклассы связей ... 459
Применение таблиц стилей ... 459
Модель форматирования ... 462
Блоковые элементы ... 463
Встроенные элементы ... 465

Свойства форматирования элементов ... 466
Шрифты ... 468
Свойство *font-family* ... 468
Свойство *font-style* ... 469
Свойство *font-variant* ... 469
Свойство *font-weight* ... 470
Свойство *font-size* ... 470
Свойство *font* ... 471
Свойство *@font-face* ... 471
Цвет и фон ... 471
Форматирование текста ... 474
Блоки ... 476
Визуальное форматирование ... 478
Абсолютное позиционирование ... 479
Относительное позиционирование ... 481
Статическое позиционирование ... 482
Визуальные эффекты ... 482
Отображение списков ... 486
Объектная модель документа ... 488
Структура документа ... 489
Объектная модель DHTML в MS Internet Explorer 4.0 ... 490
Иерархия объектов ... 491
Свойства и методы объектов ... 494
Событийная модель ... 497
Цикл жизни события ... 497
Объект *event* ... 500
Объектная модель документа в MS Internet Explorer 5.0 ... 503
Динамический HTML в Internet Explorer ... 509
Динамическое изменение документа ... 510
Раскрывающийся список ... 510
Движущийся элемент ... 514
Поиск в документе ... 516
Фильтры и переходы ... 519
Общие свойства некоторых фильтров ... 523
Описание фильтров ... 525
Описание переходов ... 530
Связывание данных с документом ... 533
Архитектура привязки данных ... 533
Объекты-источники данных ... 536
Динамический HTML в Netscape Navigator ... 543
Применение каскадных таблиц стилей ... 543
Позиционирование и объектная модель сценария ... 547
Динамическое позиционирование ... 551
Загружаемые шрифты ... 553

Глава 11. Встраиваемые компоненты ... 555

Элементы управления ActiveX ... 555
Встраивание в HTML-страницу ... 557

Элементы управления ActiveX и сценарии 561
Редактор FrontPage 98 565
Безопасность и элементы управления ActiveX 573
Цифровая подпись 574
Безопасное использование элементов управления ActiveX 576
Лицензирование элементов управления ActiveX 579
Элементы управления на HTML-страницах 581
Элемент управления *TabStrip* 582
Элемент управления *TreeView* 588
Java-апплеты 597
Приложения и апплеты 599
Структура приложения 600
Интерфейсы 602
Апплеты 603
Встраивание апплета в HTML-документ 607
Жизненный цикл апплета 608
Создание графического интерфейса пользователя 612
Обработка событий 615
Рисунки в апплетах 620
Архивы 621
Java и JavaScript 622
Заключение 630

Приложение 1. Поддержка тэгов и параметров HTML-браузерами 633

Приложение 2. Названия и коды цветов для HTML 650

Приложение 3. Совместимость приводов CD-ROM и программ для вычленения звуковых файлов 656

Предметный указатель 664

ЧАСТЬ I

1. Правила построения HTML-документов
2. Списки
3. Графика
4. Таблицы в HTML
5. Фреймы
6. Карты-изображения
7. Звук
8. Разработка HTML-страниц при помощи текстового процессора Microsoft Word

Основы HTML

Данная книга посвящена вопросам создания документов, предназначенных для использования на Web-страницах. В ней представлен весь спектр технологий создания различных Web-документов, начиная от простейших статических документов, использующих "чистый" HTML-код, до сложных документов, использующих динамическую генерацию содержимого, средства интерактивного общения с пользователем, базы данных, мультимедиа-объекты и др.

Книга состоит из двух частей. Первая часть посвящена типовым вопросам создания документов: форматирование текста, применение графики, создание списков, таблиц и др. В этой части приводятся основные тэги языка HTML и способы их применения. Для чтения первой части книги не потребуется знаний из области программирования, достаточно лишь владеть основами работы с компьютером.

Вторая часть книги, хотя и не требует изначальных знаний в области программирования, однако предназначена читателям, обладающим некоторым "программистским" мышлением.

Книга построена таким образом, чтобы первая часть была доступна практически любому читателю. Поэтому в ней рассматриваются вопросы непосредственного форматирования документов, оставляя за рамками методы современного построения документов, основанных на стилевом оформлении. Опыт обучения слушателей методам составления разнообразных документов показывает, что на начальном этапе не удается сразу пояснить необходимость применения современных методов. Действительно, человеку, который составляет свой первый документ в текстовом редакторе, невозможно объяснить, зачем, например, нужно применять стили или шаблоны вместо конкретного указания размеров шрифта и его названия.

Аналогично учащимся, постигающим азы программирования и написавшим свою первую программу, невозможно объяснить преимущества объектно-ориентированного подхода. Маленькая программа, написанная традиционным способом, будучи переписанная с использованием объектов, их методов и свойств, покажется начинающему программисту громоздкой и непонятной.

Понимание необходимости использования современных методов разработки документов приходит со временем. Поэтому в первой части книги используются только традиционные формы разработки документов, на которых можно научиться основам HTML-разметки документов. Приобретя необходимый опыт, можно приступать к изучению второй части.

В данной книге главы 1—8 и приложения написаны Сергеевым А. О., а главы 9—11 — Матросовым А. В. и Чауниным М. П.

Замечание и пожелания по книге просим направлять авторам по электронной почте **sergeev@mail.ifmo.ru**.

ГЛАВА 1

Правила построения HTML-документов

Что такое HTML

Всемирная паутина World Wide Web (WWW) соткана из Web-страниц, которые создаются с помощью так называемого языка разметки гипертекста HTML (HyperText Markup Language). Хотя многие говорят о программировании на этом языке, HTML вовсе не является языком программирования в традиционном понимании. HTML — *язык разметки* документа. При разработке HTML-документа выполняется разметка текстового документа точно так же, как это делает редактор при помощи красного карандаша. Эти пометки служат для указания формы представления информации, содержащейся в документе.

Специальные программы просмотра HTML-документов, которые часто называют *браузерами*, служат для интерпретации файлов, размеченных по правилам языка HTML, форматирования их в виде Web-страниц и отображении их содержимого на экране компьютера пользователя. Существует большое количество программ-браузеров, разработанных различными компаниями, однако, на сегодняшний день из всего разнообразия программ явно выделяются две программы-лидера — Netscape Communicator и Microsoft Internet Explorer.

Программа Netscape Navigator разработана компанией Netscape Communications Corporation. Как и у многих программных продуктов, существует ряд версий этой программы. Последней версией программы Netscape Communicator на момент написания книги являлась версия 4.7. Программа Internet Explorer разработана компанией Microsoft. Последняя версия этой программы — 5.0.

Другие браузеры значительно отстают по популярности. Несколько лет назад браузер компании Netscape занимал ведущее место среди браузеров, более двух третей пользователей применяли именно эту программу просмотра. Выпустив свой браузер, компания Microsoft приложила огромные усилия для завоевания этой части рынка. В средствах массовой информации часто

встречались сообщения о войне между браузерами за пользователей. Сейчас эти два браузера сравнимы по популярности. Росту популярности браузера Microsoft способствует включение браузера в состав операционной системы Windows 98, однако, в конечном счете, выбор браузера остается за пользователем.

Современные браузеры обладают широкими возможностями, но основным для них является интерпретация документов, размеченных по правилам HTML. Описанию этих правил, в основном, и посвящена данная книга. В первой части мы рассмотрим лишь основополагающие принципы построения HTML-документов.

Чтобы понять, что собой представляет язык разметки, вспомним старые добрые времена, когда многие работали с текстовыми редакторами типа WordStar. В них для выделения какой-либо фразы, например, полужирным шрифтом, в ее начале и в конце ставились специальные отметки (`/B` и `/b`):

```
/BЭтот текст будет выведен полужирным шрифтом/b
```

При выводе такого текста на печатающее устройство (о дисплеях еще речь не идет, в те далекие времена их еще или не было вообще или существовали алфавитно-цифровые дисплеи, не позволяющие изменять шрифты) символы `/B` заставляли использовать полужирный шрифт до тех пор, пока не встретятся символы `/b`.

HTML работает точно так же. Если есть необходимость выделить текст на экране полужирным шрифтом, то это можно сделать аналогично:

```
<B>Этот текст будет выведен полужирным шрифтом</B>
```

Символы `<B>` включают полужирное начертание, а символы `</B>` выключают его. Такие символы, которые управляют отображением текста и при этом сами не отображаются на экране, в языке HTML принято называть *тэгами* (от английского слова tag — ярлык, признак).

Все тэги языка HTML выделяются символами-ограничителями (< и >), между которыми записывается идентификатор (имя) тэга (в нашем примере это `B`), и, возможно, его *параметры*. Единственным исключением из этого правила являются тэги комментария с более сложными ограничителями (<!-- и -->). Названия тэгов, а также их параметров можно записывать на любом регистре. Для единообразия в данной книге большинство тэгов записывается прописными буквами.

Большинство тэгов HTML используется попарно, т. е. для определенного тэга, назовем его *открывающим*, в документе имеется соответствующий *закрывающий* тэг. По правилам HTML закрывающий тэг записывается так же, как и открывающий, но с символом / (прямой слэш) перед именем тэга. Единственным принципиальным различием парных тэгов является то, что закрывающие тэги не используют параметры.

Тэги, которые нуждаются в соответствующих завершающих тэгах, будем называть *тэгами-контейнерами*. Все, что записано между соответствующим открывающим и закрывающим тэгом, будем называть *содержимым* тэга-контейнера. Иногда завершающий тэг можно опускать. Например, для тэга, описывающего данные для ячейки таблицы `<TD>`, соответствующий закрывающий тэг `</TD>` можно всегда опускать. Окончание данных для ячейки таблицы будет распознано по появлению очередного тэга `<TD>` или тэга окончания строки таблицы `</TR>`.

Есть ряд тэгов, для которых завершающие тэги опускаются большинством авторов документов. Примером может служить тэг элемента списка `<LI>` или тэг абзаца `<P>`. Современные браузеры во многих случаях правильно форматируют документы, если опущены некоторые завершающие тэги, однако такая практика не может быть рекомендована.

Ряд тэгов в принципе не нуждается в завершающих тэгах. Примерами могут служить тэг вставки изображений `<IMG>`, принудительного перевода строки `<BR>`, указания базового шрифта `<BASEFONT>` и др. Часто из самого предназначения тэга можно догадаться, нуждается ли он в завершающем.

Существуют общие правила интерпретации тэгов браузерами. В отличие от языков программирования, в которых ошибочные операторы приводят к выдаче соответствующих сообщений на этапе компиляции программы и требуют правки, в HTML не принято реагировать на неверную запись тэгов. Неверно записанный тэг или его параметр должен просто игнорироваться браузером. Это общее правило для всех браузеров, под действие которого подпадают не только ошибочно записанные тэги, но и тэги, не распознаваемые данной версией браузера. Примером могут служить тэги, предложенные и реализованные для отдельного браузера и неизвестные для другого. Например, тэг-контейнер `<NOFRAMES>`, который служит для предоставления альтернативной информации браузерам, не обеспечивающим поддержку фреймовых структур, такими браузерами не будет распознан. Браузер же, поддерживающий фреймы, встретив тэг `<NOFRAMES>`, пропустит всю заключенную в нем информацию. А браузер, не знакомый с фреймами, естественно, не поймет и тэг `<NOFRAMES>`. Однако, согласно приведенному правилу, этот тэг будет просто пропущен, зато вся последующая информация будет отображена.

Тэги могут записываться с *параметрами* или *атрибутами* (от англ. attribute). В этой книге будем чаще всего использовать термин *параметр*. Наборы допустимых параметров индивидуальны для каждого тэга. Общие правила записи параметров заключаются в следующем. После имени тэга могут следовать параметры, которые отделяются друг от друга пробелами. Порядок следования параметров тэга произволен. Многие параметры требуют указания их *значений*, однако некоторые параметры не имеют значений или могут записываться без них, принимая *значения по умолчанию*. Если параметр требует значения, то оно указывается после названия параметра через знак

равенства. Значение параметра может записываться в кавычках, так и без них. Единственным случаем, в котором без кавычек не обойтись, является случай, когда в значении параметра имеются пробелы. В значениях параметров (в отличие от названий тэгов и самих параметров) иногда важен регистр записи. Приведем пример записи тэга с параметрами:

```
<TABLE BORDER ALIGN="left">
```

Здесь для тэга `<TABLE>` задано два параметра. Первый параметр `BORDER` указан без значения. Второй параметр `ALIGN` имеет значение `left`.

В последующих главах первой части книги будет описано назначение тэгов языка HTML и их параметров. В общем, тэги могут иметь различные параметры, однако существует ряд параметров, единых практически для всех тэгов. Упомянем здесь общие параметры тэгов, чтобы более не говорить о них при описании каждого тэга.

Все тэги, которые допустимо использовать в разделе `<BODY>` документа HTML, могут иметь параметры `CLASS`, `ID`, `LANG`, `LANGUAGE`, `STYLE` и `TITLE`. Использование этих параметров полезно, прежде всего, при стилевом оформлении документов, речь о котором пойдет во второй части книги.

Параметры `CLASS`, `ID`, `STYLE` поддерживаются Internet Explorer, начиная с версии 3.0, и Netscape, начиная с версии 4.0. Эти параметры нужны при использовании стилей.

Параметры `LANG`, `LANGUAGE`, `TITLE` — поддерживаются только Internet Explorer, начиная с версии 4.0. Эти параметры указывают, соответственно, используемый язык (например, для России: `LANG=ru`), язык записи скриптов (например, `LANGUAGE=JavaScript`), а также текст подсказки, выдаваемой при наведении указателя мыши на данный элемент (`TITLE`).

В современном HTML, помимо тэгов языка и их содержимого, в исходном HTML-коде также записываются *коды сценариев* (`JavaScript` или `VBScript`). В первой части книги об этом практически нигде не упоминается, зато часть вторая целиком посвящена вопросам использования сценариев.

Завершая общий обзор HTML, отметим, что простейшие HTML-документы представляют собой обычные текстовые файлы, для просмотра и редактирования которых можно воспользоваться любым текстовым редактором. Эти файлы обычно имеют расширение HTM или HTML.

Спецификации HTML

Язык HTML приобрел популярность в середине 90-х годов, благодаря экспоненциальному росту сети Интернет. К этому времени назрела необходимость стандартизации языка, поскольку различные компании, разрабатывавшие программное обеспечение для доступа в Интернет, предлагали свои

варианты инструкций HTML, число которых все возрастало и возрастало. Настала пора прийти к какому-то единому соглашению в части применения тэгов языка HTML.

Работу по созданию спецификации HTML взяла на себя организация, называемая World Wide Web Consortium (сокращенно — W3C). В ее задачу входило составление спецификации, отражающей современный уровень развития возможностей языка с учетом разнообразных предложений компаний-разработчиков браузеров. Так, в ноябре 1995 г. появилась спецификация HTML 2.0, призванная формализовать сложившуюся к концу 1994 г. практику использования HTML.

Схема утверждения спецификаций состоит в следующем. Консорциум W3C выпускает проект спецификации, после обсуждения которого выпускается так называемый черновой, рабочий (draft) вариант спецификации и предлагает его к обсуждению на определенный период. После периода обсуждения рабочий вариант спецификации может стать рекомендацией, т. е. официально признанным вариантом спецификации HTML.

Вскоре после спецификации 2.0 была выпущена рабочая версия спецификации 3.0, срок окончания периода обсуждения которой истек в сентябре 1995 г. Эта спецификация так и не была принята в качестве официальной рекомендации. В нее планировалось включить большое разнообразие тэгов и возможностей, специфичных для отдельных браузеров, однако Консорциум W3C не нашел возможности разработать хорошую спецификацию для такого большого числа инструкций.

После долгих размышлений в мае 1996 г. был выпущен проект HTML 3.2. Проект основывался на части тэгов, имеющихся в версии 3.0, которые показывали стабильность в работе. В сентябре 1996 г. после нескольких месяцев обсуждения версия 3.2 стала предлагаемой спецификацией, а в январе 1997 г. — официальной рекомендацией.

Июль 1997 года ознаменовался выходом предлагаемой спецификации HTML 4.0, которая в декабре 1997 г. стала официальной рекомендацией. На сегодняшний день это последняя из принятых спецификаций.

В приводимом здесь кратком обзоре истории развития языка HTML вряд ли стоит детально описывать особенности различных спецификаций, тем более, что в реальной жизни разработчики далеко не всегда следуют рекомендациям Консорциума. Отметим лишь некоторые идеи, заложенные в основу последней спецификации.

В спецификации HTML 4.0 ключевой идеей стало отделение описания *структуры* документа от описания его *представления* на экране монитора. Опыт показывает, что разделение структуры и представления документа уменьшает затраты на поддержку широкого спектра платформ, сред и т. п., а также облегчает внесение исправлений в документы. В соответствии с этой идеей следует шире пользоваться методами описания представления

документа с помощью *таблиц стилей*, вместо того, чтобы задавать конкретные данные о форме представления вперемешку с содержанием документа. Для реализации этой идеи в спецификации HTML 4.0 ряд тэгов, используемых для непосредственного задания формы представления HTML-элементов, отменены. К отмененным по этой причине тэгам относятся <CENTER>, <FONT>, <BASEFONT>, <S>, <STRIKE>, <U>. Среди других отмененных тэгов отметим <ISINDEX>, <APPLET>, <DIR>, <MENU>. Вместо отмененных тэгов предлагаются альтернативные варианты реализации соответствующих возможностей, на что мы обращаем особое внимание в этой книге.

Понятие *отмененного* (deprecate) тэга состоит в следующем. Если в данной спецификации языка тэг назван отмененным, то это означает, что браузеры должны пока продолжать поддержку таких тэгов, но их использование не рекомендуется. В следующих спецификациях эти тэги, возможно, будут переведены в разряд устаревших (obsolete). Устаревшие тэги могут более не поддерживаться браузерами. В спецификации HTML 4.0 устаревшими названы всего три тэга: <XMP>, <PLAINTEXT> и <LISTING>. Информацию о том, какие из тэгов включены в спецификацию, можно получить из таблицы, приводимой в приложении П1.

Официальные сведения о спецификации HTML всегда можно получить с Web-сайта Консорциума W3C по адресу **http://www.w3.org/TR/**. Спецификация 4.0 находится по адресу **http://www.w3.org/TR/REC-html40-971218**.

Заметим, что по логике вещей официальная спецификация должна играть роль руководящей и направляющей силы, обеспечивая одинаковую форму представления информации различными браузерами. Это идеальный вариант, к которому следует стремиться. На деле все обстоит не так хорошо. Постоянно появляются новые идеи, реализуемые компаниями-разработчиками в своих браузерах и пропагандируемые ими. Удачные идеи приживаются, а затем подхватываются другими разработчиками. Часть возможностей так и остается специфическими особенностями отдельного браузера. Удачные разработки в итоге попадают в спецификацию и становятся общепринятыми. Таким образом, процесс усовершенствования возможностей браузеров и уточнения спецификации идет непрерывно, оказывая взаимное влияние друг на друга.

Структура документа

Первым тэгом, с которого следует начинать описание документов HTML, является тэг <HTML>. Он должен всегда начинать описание документа, а завершать описание документа должен тэг </HTML>. Эти тэги обозначают, что находящиеся между ними строки представляют единый HTML-документ. Сам по себе документ является обыкновенным текстовым ASCII-файлом. Без этих тэгов браузер или другая программа просмотра, возможно, будет не

в состоянии идентифицировать формат документа и правильно его интерпретировать.

Чаще всего тэг <HTML> используется без параметров. В предыдущих версиях использовался параметр VERSION, отмененный спецификацией HTML 4.0. На смену этому параметру пришел тэг <!DOCTYPE>.

Большинство современных браузеров могут опознать документ и не содержащий тэгов <HTML> и </HTML>, все же их употребление крайне желательно.

Между парой тэгов <HTML> и </HTML> располагается сам документ. Документ может состоять из двух разделов — раздела заголовка (начинающийся тэгом <HEAD>) и раздела содержательной части документа (начинающийся тэгом <BODY>). Для документов, описывающих фреймовые структуры, вместо раздела BODY используется раздел FRAMESET (с тэгом <FRAMESET>). Далее будут рассмотрены правила составления разделов документа HEAD и BODY. Построение документов, содержащих фреймы, рассматривается в главе 5.

Раздел документа HEAD

Раздел документа HEAD определяет его заголовок и не является обязательным тэгом, однако хорошо составленный заголовок может быть весьма полезен. Задачей заголовка является представление необходимой информации для программы, интерпретирующей документ. Тэги, находящиеся внутри раздела HEAD (кроме названия документа, описываемого с помощью тэга <TITLE>), не отображаются на экране.

Раздел заголовка открывается тэгом <HEAD>. Обычно этот тэг следует сразу же за тэгом <HTML>. Закрывающий тэг </HEAD> показывает конец этого раздела. Между упомянутыми тэгами располагаются остальные тэги раздела заголовка.

Название документа

Тэг-контейнер <TITLE> является единственным обязательным тэгом заголовка и служит для того, чтобы дать документу название. Оно обычно показывается в заголовке окна браузера. Тэг <TITLE> нельзя путать с названием файла документа; напротив, он представляет собой текстовую строку, совершенно независимую от имени и местоположения файла, что делает его весьма полезным. Имя же файла жестко определяется операционной системой компьютера, на котором он хранится. Также следует отличать название документа (с тэгом <TITLE>) от заголовков внутри документа, обычно размечаемых тэгами <Hx>.

Примечание

Обязательность названия документа, вообще говоря, носит характер настоятельной рекомендации. Документ без тэга <TITLE> также будет отображаться

браузерами. При этом различные браузеры в качестве заголовка окна будут выдавать различную информацию. Так ранние версии браузера Netscape выдавали строчку "No title". Другие браузеры либо не показывают ничего, либо отображают адрес загруженного файла, повторяя информацию панели **Location** браузера.

Название документа записывается между тэгами `<TITLE>` и `</TITLE>` и представляет собой строку текста. В принципе, название может иметь неограниченную длину и содержать любые символы, кроме некоторых зарезервированных. На практике следует ограничиться одной строкой, имея в виду, что название появляется в заголовке окна браузера. Также следует помнить о том, что останется от названия документа при минимизации окна браузера. Можно рекомендовать ограничивать длину названия документа 60 символами. Увидеть, как отображается название в окне браузера, можно на любом рисунке в данной книге, где приводится пример отображения какого-либо документа.

По умолчанию текст, содержащийся в названии документа, используется при создании закладки (bookmark) для документа. Поэтому, для большей информативности, избегайте безликих названий (Home Page, Index и т. д.). Подобные слова, используемые в качестве названия закладки, обычно совершенно бесполезны. Название документа должно кратко характеризовать его содержание. Заметим, что при отображении на экране документов с фреймовой структурой, когда в каждый из фреймов загружается отдельный документ, имеющий свое название, на экране будет видно только название главного документа. Тем не менее, задавать название отдельных документов, предназначенных для загрузки во фреймы, также настоятельно рекомендуется. Более подробно этот вопрос рассматривается в главе 5.

Важность названия документа определяет следующий факт. Поскольку тэг `<TITLE>` располагается практически в самом начале HTML-файла, то после начала загрузки документа первым делом отображается именно оно. Далее выполняется загрузка основного содержания документа, при этом браузер начинает форматирование документа в окне. Этот процесс, вообще говоря, в зависимости от содержания и структуры документа, а также скорости соединения, может затянуться. В течение достаточно продолжительного времени пользователь будет созерцать пустой экран, единственной информативной строчкой которого будет являться название документа. Весьма часто (при обрыве соединения или, если пользователь не желает дожидаться окончания загрузки документа) вся информация о документе на этом и заканчивается.

Связь с другими документами

Часто HTML-документы связаны между собой, то есть имеют ссылки друг на друга. Ссылки могут быть как абсолютные, так и относительные. И те и другие имеют недостатки. Абсолютные ссылки могут быть слишком гро-

моздкими и переставать работать, если перемещен младший по иерархии документ. Относительные ссылки легче вводить и обновлять, но и эта связь обрывается, если перемещен старший по иерархии документ. Оба вида связей могут нарушиться при переносе документа с одного компьютера на другой.

Часто случается, что пользователь загрузил на свою машину большой документ и отключился от сети для его подробного изучения. Все ссылки в локальной копии документа перестанут работать. Для их "реанимации" придется вновь обратиться к оригиналу документа, находящемуся на удаленном компьютере.

К счастью, разработчики HTML предусмотрели эту проблему и добавили два тэга, <BASE> и <LINK>, которые включаются в заголовок для того, чтобы связь между документами не нарушалась.

Тэг *<BASE>*

Тэг <BASE> служит для указания полного базового URL-адреса документа. С его помощью относительная ссылка продолжает работать, если документ переносится в другой каталог или даже на другой компьютер. Тэг <BASE> работает аналогично команде **path** MS-DOS, что позволяет программе просмотра определить ссылку на искомый документ, даже если она находится в старшем по иерархии документе, расположенном на другом компьютере.

Тэг <BASE> имеет один обязательный параметр HREF, после которого указывается полный URL-адрес документа. Ниже показан пример использования тэга <BASE>.

```
<HTML>
<HEAD>
<TITLE>Указание базового адреса</TITLE>
<BASE HREF="//www.my_host.ru/~sergeev">
</HEAD>
<BODY>
<IMG SRC=/gifs/news.gif ALT="News">
</BODY>
</HTML>
```

Тэг <BASE> указывает браузеру, где искать файл. В случае, если пользователь работает с локальной копией файла и его машина не отключена от сети, изображение пиктограммы **News** будет найдено и показано в окне браузера.

Тэг *<LINK>*

Даже если тэг <BASE> позволяет найти файл, остается открытым вопрос о взаимоотношениях документов. Важность этих отношений возрастает пропорционально росту сложности ваших документов. Для того чтобы поддерживать логическую связь между ними, в HTML введен тэг <LINK>.

Тэг <LINK> указывает на связь документа, содержащего данный тэг и другого документа или объекта. Он состоит из URL-адреса и параметров, конкретизирующих отношения документов. Заголовок документа может содержать любое количество тэгов <LINK>. Табл. 1.1 описывает параметры тэга <LINK> и их функции.

Таблица 1.1. *Параметры тэга <LINK>*

Параметр	Назначение
HREF	Указывает на URL-адрес другого документа
REL	Определяет отношение между текущим и другим документом
REV	Определяет отношение между другим документом и текущим (отношение, обратное REL)
TYPE	Указывает тип и параметры присоединенной таблицы стилей

Приведем примеры тэга <LINK> с параметрами:

```
<LINK REL="contents" HREF =".../toc.html">
<LINK HREF="mailto:sergeev@mail.ifmo.ru" REV="made">
```

Первая строка указывает на связь с файлом оглавления документа (toc.html — table of contents) с прямым отношением contents. Вторая строка описывает связь с URL-адресом автора документа (с обратным отношением made).

Между документами может существовать множество различных отношений. Примеры других значений параметра REL: bookmark, copyright, glossary, help, home, index, toc, next, previous. Параметр REV может также принимать значения: author, editor, publisher, owner.

Тэг *<META>*

Разработка новых спецификаций языка разметки гипертекста занимает немалый срок, и за это время компании, производящие браузеры, успевают выпустить несколько версий своих продуктов. Поэтому в раздел заголовка может быть добавлен еще один тэг <META>, позволяющий авторам документа определять информацию, не имеющую отношения к HTML.

Эта информация используется браузером для действий, которые не предусмотрены текущей спецификацией HTML. Тэг <META> не потребуется вам для создания первых HTML-документов, но он вам наверняка понадобится, когда ваши страницы станут более сложными.

Пример:

```
<META HTTP-EQUIV=refresh" CONTENT="60" RL="www.my_host.ru/homepage.html">
```

Браузеры Netscape Navigator и Internet Explorer поймут эту запись как инструкцию ожидать 60 секунд, а затем загрузить новый документ. Такая инструкция часто используется при изменении местоположения документов. Небольшой документ с приведенной строкой может быть оставлен на старом месторасположении документа для автоматической ссылки на его новое месторасположение.

Следующая строка:

```
<META HTTP-EQUIV=refresh" CONTENT="60">
```

инструктирует браузер перезагружать страницу каждые 60 секунд. Это может быть полезно, если данные на странице часто обновляются, например, в случае отслеживания котировок акций.

Стало весьма популярным применение элемента `<META>` для решения некоторых типичных задач. В качестве примера можно привести указание ключевых слов, используемых поисковыми системами. Этот способ позволяет включать в индекс документа дополнительные слова, которые могут явно не входить в его содержание. Для этого в тэге `<META>` в качестве значения параметра `NAME` указывается имя некоторого свойства. А при помощи параметра `CONTENT` указывается значение данного свойства, например:

```
<META NAME="author" CONTENT="Александр Сергеев">
```

Спецификация HTML не определяет каких-либо конкретных имен свойств, записываемых в тэге `<META>`. Однако есть несколько часто применяемых свойств, например, `description`, `keywords`, `author`, `robots` и др.:

```
<META NAME="description" CONTENT="Описание возможностей языка HTML 4.0">
<META NAME="keywords" CONTENT="тэг, гипертекст, HTML, браузер">
```

Приведенные тэги `<META>` вполне могли бы быть указаны, например, для электронного варианта данной книги.

Тэг `<META>` может иметь параметры, указанные в табл. 1.2.

Таблица 1.2. *Параметры тэга* `<META>`

Параметр	Назначение
`HTTP-EQUIV`	Определяет свойство для тэга
`NAME`	Обеспечивает дополнительное описание тэга. Если этот параметр опущен, он считается эквивалентным параметру `HTTP-EQUIV`
`URL`	Определяет адрес документа для свойства
`CONTENT`	Определяет возвращаемое значение для свойства

Еще одно важное предназначение тэга <META> — это указание кодировки текста. Так, для текста на русском языке в кодировке Windows нужно записать следующую строчку:

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=Windows-1251">
```

Другие элементы заголовка

В разделе заголовка документа могут присутствовать еще два тэга — <STYLE> и <SCRIPT>. Их назначение связано с использованием таблиц стилей в документе и записью скриптов. Эти вопросы подробно рассматриваются во второй части книги.

Раздел документа BODY

В этом разделе документа располагается его содержательная часть. Большинство тэгов, рассматриваемых далее в этой главе и последующих, должно располагаться в данном разделе документа. Здесь мы рассмотрим лишь некоторые общие вопросы.

Раздел документа BODY должен начинаться тэгом <BODY> и завершаться тэгом </BODY>, между которыми располагается все содержимое данного раздела. Строго говоря, наличие этих тэгов не является обязательным, поскольку браузеры могут определить начало содержательной части документа по контексту. Однако их употребление рекомендуется.

Тэг <BODY> имеет ряд параметров, ни один из которых не является обязательным. Перечень параметров приведен в табл. 1.3.

Таблица 1.3. *Перечень параметров тэга <BODY>*

Параметр	Назначение
ALINK	Определяет цвет активной ссылки
BACKGROUND	Указывает на URL-адрес изображения, которое используется в качестве фонового
BOTTOMMARGIN	Устанавливает границу нижнего поля документа в пикселах
BGCOLOR	Определяет цвет фона документа
BGPROPERTIES	Если установлено значение FIXED, фоновое изображение не прокручивается
LEFTMARGIN	Устанавливает границу левого поля документа в пикселах
LINK	Определяет цвет еще не просмотренной ссылки
RIGHTMARGIN	Устанавливает границу правого поля документа в пикселах

Таблица 1.3 *(окончание)*

Параметр	Назначение
SCROLL	Устанавливает наличие или отсутствие полос прокрутки окна браузера
TEXT	Определяет цвет текста
TOPMARGIN	Устанавливает границу верхнего поля документа в пикселах
VLINK	Определяет цвет уже просмотренной ссылки

Использование параметров BACKGROUND и BGCOLOR, определяющих фон документа, подробно рассмотрено в главе 3.

Параметр BGPROPERTIES, принимающий единственное значение FIXED, поддерживается только браузером Microsoft Internet Explorer.

Параметры BOTTOMMARGIN, LEFTMARGIN, RIGHTMARGIN и TOPMARGIN, задающие расстояния в пикселах между краями текста и соответствующими краями окна, а также параметр SCROLL, распознаются только браузером Microsoft Internet Explorer, начиная с версии 4.0.

Параметры ALINK, LINK, TEXT и VLINK определяют цвета ссылок и текста документа.

В языке HTML цвета определяются цифрами в шестнадцатеричном коде. Цветовая система базируется на трех основных цветах — красном, зеленом и синем — и обозначается RGB. Для каждого цвета задается шестнадцатеричное значение в пределах от 00 до FF, что соответствует диапазону 0—255 в десятичном исчислении. Затем эти значения объединяются в одно число, перед которым ставится символ #. Например, число #800080 обозначает фиолетовый цвет. Чтобы не запоминать совокупности цифр, вместо них можно пользоваться названиями цветов, которые приводятся в приложении П2. Ранние версии браузеров распознавали только названия 16 стандартных цветов, отмеченных в приложении звездочками. Современные версии браузеров распознают все 140 названий цветов.

Форматирование текста

В данном разделе будут рассмотрены возможности форматирования отдельных символов текста документа.

Тэги уровня блока и последовательные тэги

Некоторые HTML-тэги, которые могут появляться в разделе BODY, называют *тэгами уровня блока* (block level), в то время как другие *последователь-*

ными (inline) тэгами или, называя по-другому, *тэгами уровня текста* (text level), хотя такое разделение тэгов по уровням в известной степени условно.

Различие уровней HTML-тэгов заключается в следующем: тэги уровня блока могут содержать последовательные тэги и другие тэги уровня блока, тогда как последовательные тэги содержат только данные и другие последовательные тэги. Блоковые тэги описывают более крупные структуры документов, по сравнению с последовательными тэгами.

По умолчанию тэги этих видов размещаются в тексте описания различным образом: тэги уровня блока начинаются с новой строки, в то время как последовательные — нет.

Логическое и физическое форматирование

Для форматирования текста HTML-документов предусмотрена целая группа тэгов, которую можно условно разделить на тэги *логического* и *физического* форматирования.

Тэги логического форматирования обозначают (своими именами) структурные типы своих текстовых фрагментов, такие, например, как *программный код* (тэг `<CODE>`), *цитата* (тэг `<CITE>`), *аббревиатура* (тэг `<ABBR>`) и т. д. (см. описания этих и других подобных тэгов в следующем разделе). С помощью тэгов `<EM>` и `<STRONG>` можно, например, отметить отдельные фрагменты как *выделенные*, или *сильно выделенные*. Заметим, что речь идет о *структурной разметке*, которая не влияет на конкретное экранное представление фрагмента браузером. Поэтому такая разметка и называется *логической*. Фрагменты с логическим форматированием браузеры отображают на экране определенным образом, заданным по умолчанию. Вид отображения никак не связан со структурным типом фрагмента (т. е. именем тэга логического форматирования), но может быть легко переопределен.

Тэги физического форматирования определяют *формат отображения* указанного в них фрагмента текста в окне браузера (согласно предпочтениям автора документа). Например, для отображения фрагмента *курсивом* можно использовать тэг курсива `<I>`. Этот и другие, часто используемые тэги физического форматирования описаны ниже в разделе *"Тэги физического форматирования текста"* этой главы.

Между разработчиками HTML-документов долгое время шли споры о преимуществах и недостатках того или иного подхода. С выходом спецификации HTML 4.0 эти споры завершились в пользу применения логического форматирования, поскольку был провозглашен принцип отделения структуры документа от его представления. Действительно, только на базе логического форматирования можно гибко управлять представлением документа, используя современные методы (основанные на таблицах стилей, динамически изменяющихся документах и т. д.).

Тем не менее, на настоящий момент может свободно использоваться и физическое форматирование. В спецификации HTML 4.0 некоторые тэги физического форматирования не рекомендуются для применения, однако, пока они все еще поддерживаются всеми браузерами. Заметим, что некоторые тэги логического форматирования, призванные заменить отдельные тэги физического форматирования, распознаются не всеми браузерами, что делает их применение крайне неудобным. Примером может служить логический тэг `<DEL>`, который рекомендуется использовать вместо физического тэга `<STRIKE>`.

Рассматриваемые ниже тэги относятся к тэгам уровня текста, т. е. призваны, в основном, размечать небольшие группы символов. Некоторые тэги могут задавать разметку и на уровне блока.

Тэги логического форматирования текста

Тэг *<ABBR>*

Тэг `<ABBR>` отмечает свой текст как *аббревиатуру* (ABBReviation). Несмотря на то, что этот тэг включен в спецификацию HTML 4.0, он до настоящего времени не поддерживается ни одним браузером.

Тэг *<ACRONYM>*

Тэг `<ACRONYM>`. Так же, как и тэг `<ABBR>`, используется для отметки аббревиатур. Этим тэгом рекомендуется отмечать так называемые *акронимы*, т. е. произносимые слова, состоящие из аббревиатур. Тэг `<ACRONYM>` возможно в будущем станет использоваться для невизуального отображения элементов, например при речевом синтезе.

Данный тэг удобно использовать в сочетании с параметром `TITLE`, в качестве значения которого можно указать полную форму записи аббревиатуры. Тогда визуальные браузеры при наведении курсора на текст, размеченный тэгом `<ACRONYM>`, будут выдавать полное наименование в виде появляющейся подсказки.

Заметим, что тэг `<ACRONYM>` распознается только браузером Microsoft Internet Explorer. Пример:

```
<ACRONYM TITLE="Санкт-Петербургский государственный институт
  точной механики и оптики">
СПбГИТМО</ACRONYM> — один из ведущих технических вузов Санкт-Петербурга
```

Тэг *<CITE>*

Тэг `<CITE>` используется для отметки цитат или названий книг и статей, ссылок на другие источники и т. д. Браузерами такой текст обычно выводится курсивом. Пример:

```
<CITE>Невское время</CITE> является одной из наиболее популярных
  городских газет Санкт-Петербурга
```

Тэг *<CODE>*

Тэг `<CODE>` отмечает свой текст как *небольшой фрагмент программного кода*. Как правило, отображается моноширинным шрифтом. Этот тэг не следует путать с тэгом `<PRE>`, являющимся элементом уровня блока, который следует использовать для отметки *больших фрагментов (листингов) кода*.

Например:

```
Пример простейшего оператора языка программирования C:<BR>
<CODE>puts("Hello, World!");</CODE>
```

Есть еще одно различие в использовании тэгов `<CODE>` и `<PRE>`. В коде программ часто бывает важно наличие нескольких идущих подряд пробелов. Их отображение будет сохранено только при использовании тэга `<PRE>`.

Тэг *<DEL>*

Тэг `<DEL>` отмечает свой текст как *удаленный*. Этот элемент полезно использовать для отметки изменений, вносимых в документ от версии к версии. Тэг `<DEL>` может использоваться как элемент уровня текста и как элемент уровня блока.

Тэг имеет два необязательных параметра: `CITE` и `DATETIME`. Значение параметра `CITE` должно представлять собой URL-адрес документа, поясняющего причины удаления данного фрагмента.

Параметр `DATETIME` указывает дату удаления в формате: YYYY-MM-DDThh:mm:ssTZD, определяющем год, месяц, число, часы, минуты и секунды удаления, а также часовой пояс (Time Zone). Например:

```
Последней принятой спецификацией языка разметки HTML является версия
<DEL DATETIME=1999-10-29T16:12:53+0.00>3.2</DEL> 4.0
```

Текст, помеченный тэгом `<DEL>` обычно отображается перечеркнутым текстом. В спецификации HTML 4.0 этому тэгу отдается предпочтение перед тэгом физического форматирования `<STRIKE>` или `<S>`, обозначающих перечеркнутый текст. Однако тэг `<DEL>` в настоящее время распознается только браузером Microsoft Internet Explorer.

Тэг *<DFN>*

Тэг `<DFN>` отмечает свой текстовый фрагмент как *определение* (DeFinitioN). Например, этим тэгом можно отметить какой-либо термин, когда он встречается в тексте в первый раз. Пример:

```
<DFN>Internet Explorer</DFN> — это популярный Web-браузер
```

Тэг `<DFN>` поддерживается только браузером Microsoft Internet Explorer. Отображается по умолчанию курсивом.

Тэг *<INS>*

Тэг <INS> отмечает свой текст как *вставку* (INSertion). Этот элемент полезно использовать для отметки изменений, вносимых в документ от версии к версии. Тэг <INS> может использоваться как элемент уровня текста и как элемент уровня блока.

Тэг имеет два необязательных параметра: CITE и DATETIME. Значение параметра CITE должно представлять собой URL-адрес документа, поясняющего подробности внесенных дополнений.

Параметр DATETIME указывает дату вставки в формате: YYYY-MM-DDThh:mm:ssTZD, определяющем год, месяц, число, часы, минуты и секунды вставки, а также часовой пояс (Time Zone).

Текст, помеченный тэгом <INS>, обычно отображается подчеркнутым текстом. Тэг <INS> в настоящее время распознается только браузером Microsoft Internet Explorer.

Тэг *<EM>*

Тэг <EM> (EMphasis — выделение, подчеркивание) используется для выделения важных фрагментов текста. Браузеры обычно отображают такой текст курсивом. Пример:

```
Пример выделения <EM>отдельных слов</EM> текста
```

Применение данного тэга предпочтительнее применения тэга физического форматирования <I>.

Тэг *<KBD>*

Тэг <KBD> отмечает текст как вводимый пользователем с клавиатуры. Обычно отображается моноширинным шрифтом, например:

```
Чтобы запустить текстовый редактор, напечатайте: <KBD>notepad</KBD>
```

Применение данного тэга предпочтительнее применения тэга физического форматирования <TT>.

Тэг *<Q>*

Тэг <Q> отмечает *короткие цитаты* в строке текста. В отличие от тэга уровня блока <BLOCKQUOTE> при отображении не выполняется отделение размеченного текста пустыми строками. Обычно отображается курсивом. Тэг <Q> (в отличие от <BLOCKQUOTE>) в настоящее время распознается только браузером Microsoft Internet Explorer.

Тэг имеет параметр CITE, в качестве значения которого можно указать источник цитаты.

Тэг *<SAMP>*

Тэг <SAMP> отмечает текст как *образец* (SAMPle). Обычное использование этого тэга — отметка текста, выдаваемого программами (sample output). Ис-

пользуется также для выделения нескольких символов моноширинным шрифтом.

Применение данного тэга предпочтительнее применения тэга физического форматирования <TT>. Например:

```
В результате работы программы будет напечатано: <SAMP>Hello, World!</SAMP>.
```

Тэг *<STRONG>*

Тэг <STRONG>, как правило, используется для *выделения* важных фрагментов текста. Браузеры обычно отображают такой текст полужирным шрифтом. Пример:

```
Санкт-Петербург расположен в самой восточной оконечности <STRONG>Финского
  залива</STRONG> в устье реки <STRONG>Невы</STRONG>
```

Применение данного тэга предпочтительнее применения тэга физического форматирования <B>. Тэгом <STRONG> обычно размечают более важные фрагменты текста, чем те, что размечены тэгом <EM>.

Тэг *<VAR>*

Тэг <VAR> отмечает *имена переменных* программ. Обычно такой текст отображается курсивом. Пример:

```
Задайте значение переменной <VAR>N</VAR>
```

Отображение некоторых из примеров, приведенных при описании тэгов логического форматирования текста, показано на рис. 1.1. На рисунке видно, как при наведении указателя мыши на текст, отмеченный тэгом <ACRONYM>, выдается подсказка.

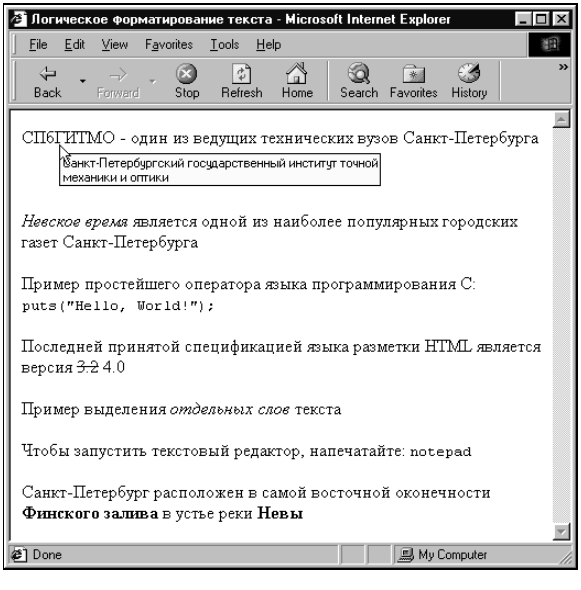


Рис. 1.1. Примеры форматирования текста

Вы, наверное, обратили внимание на то, что некоторые элементы дают одинаковый результат. Более того, часть элементов может никак не изменять представление фрагмента текста на экране. Может возникнуть законный вопрос: для чего создано такое разнообразие элементов форматирования?

Ответ — в названии этой группы элементов. Они предназначены для расстановки логических ударений, выделения логических частей и подчеркивания сути высказываний. Их использование весьма актуально, поскольку, вероятно, в ближайшем будущем возможности браузеров возрастут, например, станет возможен поиск цитат на Web-пространстве, а может быть следующее поколение браузеров научится читать документы вслух. Кроме того, авторам документов ничто не мешает уже сегодня, применяя таблицы стилей, задать желаемое отображение для любого из тэгов, переопределив значения по умолчанию.

Тэги физического форматирования текста

Приведем описание тэгов физического форматирования. Часть из них не рекомендуется к использованию спецификацией HTML 4.0 по приведенным выше причинам. Некоторые тэги отменены (deprecate) спецификацией HTML 4.0, однако они продолжают поддерживаться браузерами.

Тэг *<B>*

Тэг `<B>` отображает текст полужирным шрифтом. В большинстве случаев рекомендуется вместо тэга `<B>` использовать тэг логического форматирования `<STRONG>`. Например:

```
Это <B>полужирный</B> шрифт.
```

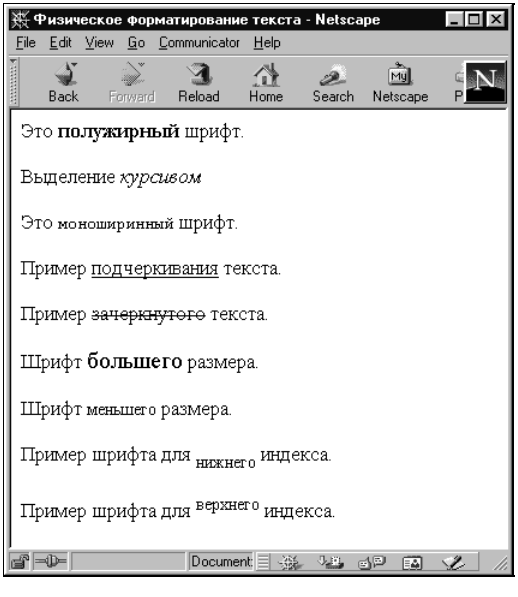


Рис. 1.2. Примеры физического форматирования текста (браузером Netscape)

Тэг *<I>*

Тэг <I> отображает текст курсивом. Для большинства случаев вместо этого тэга рекомендуется использовать тэги <EM>, <DFN>, <VAR> или <CITE>, поскольку последние лучше отражают назначение выделяемого текста. Например:

```
Выделение <I>курсивом</I>
```

Тэг *<TT>*

Тэг <TT> отображает текст моноширинным шрифтом. Для большинства случаев вместо этого тэга лучше использовать тэги <CODE>, <SAMP> или <KBD>. Пример:

```
Это <TT>моноширинный</TT> шрифт.
```

Тэг *<U>*

Тэг <U> отображает текст подчеркнутым. Отмененный тэг. Вместо него рекомендуется использовать тэги <STRONG> или <CITE>. Например:

```
Пример <U>подчеркивания</U> текста.
```

Тэги *<STRIKE>* и *<S>*

Тэги <STRIKE> и <S> отображают текст, перечеркнутый горизонтальной линией. Отмененный тэг. Вместо него следует использовать тэг <DEL>. Например:

```
Пример <STRIKE>зачеркнутого</STRIKE> текста.
```

В настоящее время тэг <DEL> поддерживается не всеми браузерами, поэтому пока рекомендуется использовать <DEL> в сочетании с тэгом <STRIKE>. А именно, внутрь тэга-контейнера <DEL> можно вложить пару тэгов <STRIKE>...</STRIKE>.

Тэг *<BIG>*

Тэг <BIG> выводит текст шрифтом большего (чем непомеченная часть текста) размера. Вместо данного элемента лучше использовать <STRONG> или тэги заголовков, например, <H3>. Большинство браузеров поддерживают вложенные тэги <BIG>, однако использовать такой подход не рекомендуется. Например:

```
Шрифт <BIG>большего</BIG> размера.
```

Тэг *<SMALL>*

Тэг <SMALL> выводит текст шрифтом меньшего размера. Поскольку в HTML нет тэга, противоположного по действию тэгу <EM>, то для этих целей можно применять тэг <SMALL>. Большинство браузеров поддерживают вложенные тэги <SMALL>, однако использовать такой подход не рекомендуется. Например:

```
Шрифт <SMALL>меньшего</SMALL> размера.
```

Тэг *<SUB>*

Тэг `<SUB>` сдвигает текст ниже уровня строки и выводит его (если возможно) шрифтом меньшего размера. Удобно использовать для математических индексов. Например:

```
Пример шрифта для <SUB>нижнего</SUB> индекса.
```

Тэг *<SUP>*

Тэг `<SUP>` сдвигает текст выше уровня строки и выводит его (если возможно) шрифтом меньшего размера. Удобно использовать для задания степеней чисел в математике. Например:

```
Пример шрифта для <SUP>верхнего</SUP> индекса.
```

Тэг *<BLINK>*

Тэг `<BLINK>` отображает мигающий текст. Этот тэг не входит в спецификацию HTML и поддерживается только браузером Netscape. Опытные разработчики крайне редко прибегают к использованию этого тэга, поскольку наличие на странице мигающих символов раздражает многих пользователей.

Тэг *<SPAN>*

Тэг-контейнер `<SPAN>` является аналогом тэга уровня блока `<DIV>`. Может использоваться в тех случаях, когда требуется отметить фрагмент текста для задания его свойств, и при этом не удается использовать никакой другой структурный тэг форматирования.

Браузер Microsoft Internet Explorer дополнительно разрешает использование следующих параметров тэга: `DIR`, `DATAFLD`, `DATAFORMATAS`, `DATASRC`. Описание параметров можно найти во второй части книги.

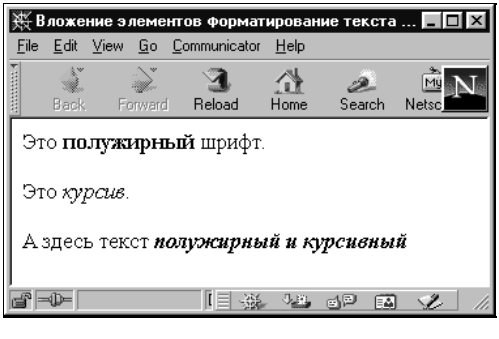


Рис. 1.3. Использование вложенных тэгов форматирования текста

Тэги форматирования могут быть вложенными друг в друга. При этом нужно внимательно следить, чтобы один контейнер находился целиком в другом контейнере. На рис. 1.3 показан пример использования вложения элемента курсива в элемент полужирного шрифта. Использован следующий фрагмент HTML-кода:

```
Это <B>полужирный</B> шрифт.
<P>
Это <I>курсив</I>.
<P>
А здесь текст <B><I>полужирный и курсивный</I></B>
```

Тэг *<FONT>*

Тэг <FONT> указывает параметры шрифта. Он относится к тэгам физического форматирования уровня текста.

Назначение параметров шрифта непосредственно в тексте документа нарушает основную идею разделения содержательной части документа и описания формы представления документа. Поэтому в спецификации HTML 4.0 данный тэг, а также тэг <BASEFONT> отнесены к отмененным. Их дальнейшее применение не рекомендуется.

Несмотря на эти грозные предупреждения, видимо, для самых простых документов физическое форматирование можно считать допустимым. Кроме того, начинать обучение основам форматирования проще всего именно с правил непосредственного указания форматов элементов. До стилевого оформления начинающий разработчик должен еще дорасти.

Тэг <FONT> относится к последовательным элементам, поэтому не может включать в себя элементы уровня блока, например, <P> или <TABLE>.

Для тэга могут задаваться следующие параметры: FACE, SIZE и COLOR. Заметим, что браузер Netscape допускает также использование двух дополнительных параметров: POINT-SIZE и WEIGHT, описание которых опускаем.

- FACE

  Параметр FACE служит для указания типа шрифта, которым программа просмотра пользователя будет выводить текст (если такой шрифт имеется на компьютере). Значением данного параметра служит название шрифта, которое должно в точности совпадать с названием шрифта, имеющего у пользователя. Если такого шрифта не будет найдено, то данное указание будет проигнорировано и будет использован шрифт, установленный по умолчанию.

  Можно указать как один, так и несколько названий шрифтов, разделяя их запятыми. Это весьма полезное свойство, так как в разных системах могут быть почти идентичные шрифты, называющиеся по-разному. Другим важным качеством является задание предпочтения использования шрифтов. Список шрифтов просматривается слева направо. Если на компьютере пользователя нет шрифта, указанного в списке первым, то делается попытка найти следующий шрифт и т. д.

  Приведем пример использования параметра FACE:

```
<HTML>
<HEAD>
```

```
<TITLE>Назначение шрифтов</TITLE>
</HEAD>
<BODY>
Текст, записанный шрифтом по умолчанию.
<BR>
<FONT FACE="Verdana", "Arial", "Helvetica">
Пример задания названия шрифта.
</FONT>
</BODY>
</HTML>
```

На рис. 1.4 показано отображение примера браузером Netscape. В примере в качестве предпочитаемого указывается шрифт `Verdana`, при его отсутствии будет использован шрифт `Arial` и т. д.

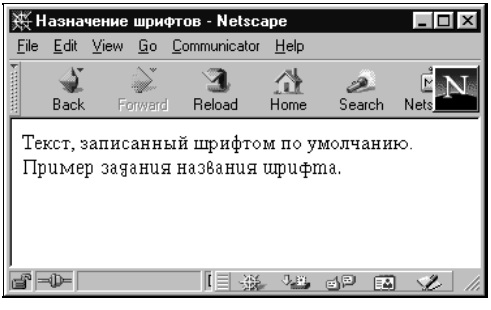


Рис. 1.4. Отображение примера браузером Netscape

- `SIZE`

Этот параметр служит для указания размеров шрифта в условных единицах от 1 до 7. Конкретный размер шрифта зависит от используемой программы просмотра. Принято считать, что размер "нормального" шрифта соответствует значению 3.

Настройки размеров шрифта, используемых по умолчанию, а также величины абсолютного изменения размеров шрифта, зависят от браузеров. На рис. 1.5 показано окно настройки браузера Netscape, в котором задаются шрифты, используемые по умолчанию.

Размер шрифта указывается как абсолютной величиной (`SIZE=2`), так и относительной (`SIZE=+1`). Последний способ часто используется в сочетании с заданием базового размера шрифта с помощью тэга `<BASEFONT>`.

Примечание

При указании размеров шрифтов записи типа "2" и "+2" (в отличие от большинства языков программирования, в которых унарный знак "+" можно опускать) дают принципиально разный результат.

Приведем пример, в котором использованы различные способы назначения размеров шрифтов. Отображение примера показано на рис. 1.6.

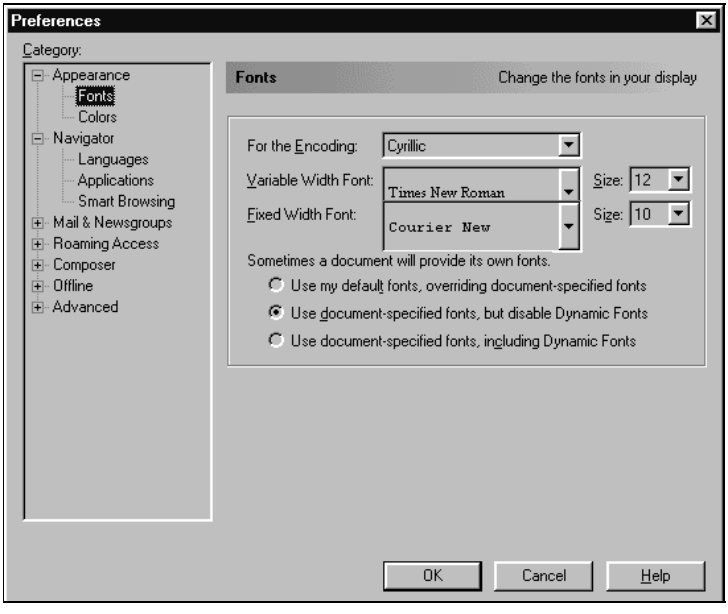


Рис. 1.5. Окно настройки параметров шрифтов браузера Netscape

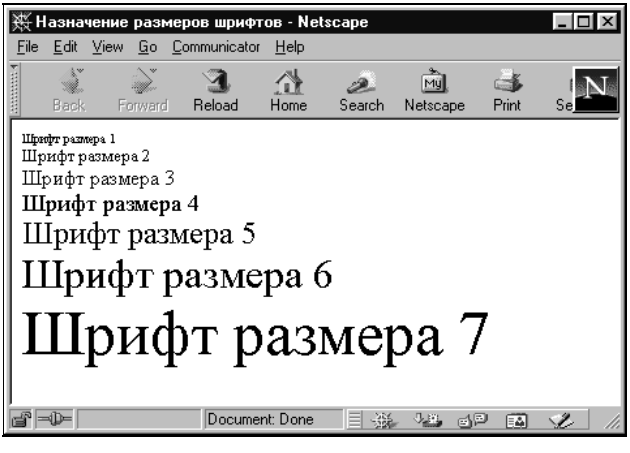


Рис. 1.6. Назначение размеров шрифтов

```
<HTML>
<HEAD>
<TITLE>Назначение размеров шрифтов</TITLE>
</HEAD>
<BODY>
<FONT SIZE=1>Шрифт размера 1</FONT><BR>
<FONT SIZE=-1>Шрифт размера 2</FONT><BR>
<FONT SIZE=3>Шрифт размера 3</FONT><BR>
<FONT SIZE=4>Шрифт размера 4</FONT><BR>
```

```
<FONT SIZE=5>Шрифт размера 5</FONT><BR>
<FONT SIZE=+3>Шрифт размера 6</FONT><BR>
<FONT SIZE=7>Шрифт размера 7</FONT><BR>
</BODY>
</HTML>
```

- COLOR

  Этот параметр устанавливает цвет шрифта, который может задаваться с помощью стандартных имен или в формате #RRGGBB. Приведем пример документа с разноцветным текстом.

```
<HTML>
<HEAD>
<TITLE>Выбор цвета шрифта</TITLE>
</HEAD>
<BODY>
<FONT COLOR=green>Текст зеленого цвета</FONT><BR>
<FONT COLOR=#FF0000>Текст красного цвета</FONT><BR>
</BODY>
</HTML>
```

Тэг *<BASEFONT>*

Тэг <BASEFONT> используется для указания размера, типа и цвета шрифта, используемого в документе по умолчанию. Эти значения обязательны для всего документа, однако могут в нужных местах переопределяться с помощью тэга <FONT>. После закрывающего тэга </FONT> действие тэга <BASEFONT> восстанавливается. Значения параметров шрифтов, используемых по умолчанию, могут неоднократно переопределяться в документе, т. е. тэг <BASEFONT> может появляться в документе любое количество раз.

Примечание

Тэг <BASEFONT> может появляться также и в разделе <HEAD> документа.

Заметим, что для тэга <BASEFONT> не существует закрывающего тэга.

В качестве параметров могут использоваться точно такие же параметры, что и для тэга <FONT>, а именно: FACE, SIZE и COLOR. Назначение и правила записи параметров аналогичны.

Примечание

Браузер Netscape не допускает применение параметра FACE тэга <BASEFONT>.

Приведем пример использования тэга <BASEFONT>.

```
<HTML>
<HEAD>
```

```
<TITLE>Назначение размеров шрифтов</TITLE>
</HEAD>
<BODY>
Текст, записанный шрифтом по умолчанию.

<BASEFONT SIZE=2>
<P>
Шрифт размера 2.

<BASEFONT SIZE=4>
<P>
Шрифт размера 4.
<P>
   <TABLE BORDER=1>
   <TR>
   <TD>Текст внутри ячейки таблицы</TD>
   </TR>
   </TABLE>
<P>
Текст после таблицы

</BODY>
</HTML>
```

В приведенном примере дважды переопределяется размер шрифта, используемого по умолчанию. Изначально он равен 3 (по умолчанию). Затем устанавливается равным 2, далее — 4. Обратите внимание на отображение данного примера (рис. 1.7). Видно, что для таблиц назначение тэга `<BASEFONT>` не действует. Это характерно для многих браузеров, хотя формально нарушает идею применения тэга.

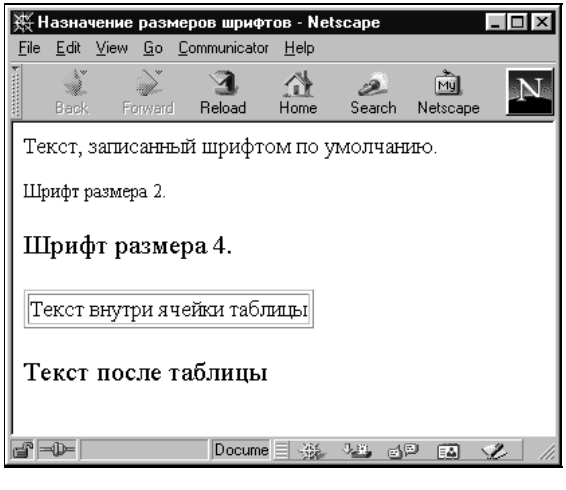


Рис. 1.7. Отображение примера с тэгом `<BASEFONT>` (браузером Netscape)

Форматирование HTML-документа

Разделение на абзацы

Любые тексты, будь то школьное сочинение, заметка в газете или техническое описание устройства имеют определенную структуру. Элементами такой структуры являются заголовки, подзаголовки, таблицы, абзацы и др.

Одним из первых правил составления практически любых документов является разбиение его текста на отдельные абзацы, выражающие законченную мысль. HTML-документы не являются исключением из этого правила. При создании документов с помощью текстовых редакторов разбиение на абзацы выполняется вводом символа перевода строки. Большинство редакторов реализует это при нажатии клавиши <Enter>. В HTML-документах символы перевода строки не приводят к образованию нового абзаца.

Язык HTML предполагает, что автор документа ничего не знает о компьютере своего читателя. Читатель вправе установить любой размер окна и пользоваться любым из имеющихся у него шрифтов. Это означает, что место переноса в строке определяется только программой просмотра и установками конечного пользователя. Поскольку символы перевода строки оригинального документа игнорируются, то текст, отлично смотревшийся в окне редактора автора документа, может превратиться в сплошной неудобочитаемый текст в окне программы просмотра.

Избежать этой неприятности позволяет применение специального тэга разделения на абзацы `<P>`. Перед началом каждого абзаца текста следует поместить тэг `<P>`. Закрывающий тэг `</P>` не обязателен. Браузеры обычно отделяют абзацы друг от друга пустой строкой.

Примечание

Браузеры обычно интерпретируют несколько стоящих подряд тэгов абзаца `<P>` как один. То же самое относится и к тэгу перевода строки `<BR>`. Поэтому создать несколько пустых строк при помощи этих тэгов не удается.

Тэг `<P>` может задаваться с параметром горизонтального выравнивания `ALIGN`. Возможные значения параметра приведены в табл. 1.4. По умолчанию выполняется выравнивание по левому краю.

Таблица 1.4. *Значения параметра* `ALIGN`

Значение параметра `ALIGN`	Действие
`LEFT`	Выравнивание текста по левой границе окна браузера
`CENTER`	Выравнивание по центру окна браузера

Таблица 1.4 *(окончание)*

Значение параметра ALIGN	Действие
RIGHT	Выравнивание по правой границе окна браузера
JUSTIFY	Выравнивание по ширине (по двум сторонам)

Заметим, что выравнивание по ширине (ALIGN = JUSTIFY) долгое время не поддерживалось браузерами. Во многих описаниях языка HTML для значений параметра выравнивания указывается только три варианта (LEFT, CENTER и RIGHT). В настоящее время все популярные браузеры умеют выполнять выравнивание по ширине. Некоторые проблемы создания документов с выравниванием по ширине рассмотрены также в главе 8.

Примечание

Отсутствие выравнивания по ширине в настоящее время кажется удивительным. Этот режим применяется очень часто для печатных изданий. Достаточно взять в руки любую газету или внимательно присмотреться к абзацам данной книги — все они выровнены по ширине. Однако для HTML-документов до последнего времени выравнивание по ширине не допускалось. Всего лишь пару лет назад ни один браузер не имел такого режима. Так, читатели, до сих пор использующие браузер Netscape версии 3.x, не смогут увидеть ровные строчки в документе. Можно попытаться предугадать, какие же еще возможности появятся в будущем у браузеров, сравнив возможности мощных текстовых редакторов и современных браузеров. Примером не реализованных пока возможностей может являться автоматическая расстановка переносов и пр.

Перевод строки

При отображении текстовых документов в браузере место переноса строки в пределах абзаца определяется автоматически в зависимости от размера шрифтов и размера окна просмотра. Перенос строки может осуществляться только по символам-разделителям слов (например, пробелам). Иногда в документах требуется задать принудительный перевод строки, реализующийся независимо от параметров настроек браузера. Для этого служит тэг принудительного перевода строки <BR>, который не имеет соответствующего закрывающего тэга. Включение тэга
 в текст документа обеспечит размещение последующего текста с начала новой строки. Например, такой подход может использоваться для создания структур типа списков без использования специальных тэгов разметки списка. Или, например, без данного тэга не обойтись для отображения стихотворений и т. п.

Приведем пример использования принудительного перевода строки (рис. 1.8):

```
<HTML>
<HEAD>
<TITLE>Использование принудительного перевода строки</TITLE>
```

```
</HEAD>
<BODY>
Над омраченным Петроградом<BR>
Дышал ноябрь осенним хладом.<BR>
Плеская шумною волной<BR>
В края своей ограды стройной,<BR>
Нева металась, как больной<BR>
В своей постеле беспокойной.
<P>
<CITE>А.С.Пушкин. Медный всадник</CITE>
</BODY>
</HTML>
```

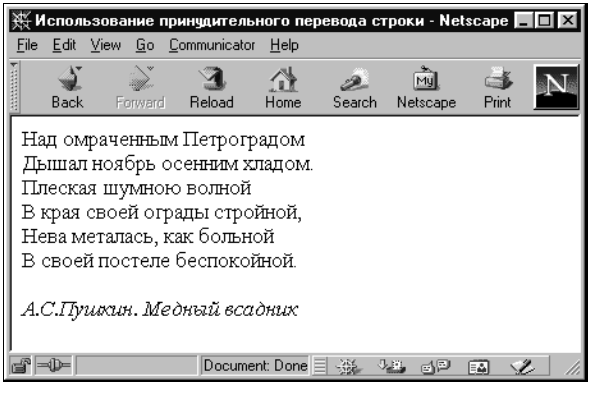


Рис. 1.8. Тэг
 можно использовать для принудительного перевода строки

В отличие от тэга абзаца <P> при использовании тэга
 не будет образована пустая строка.

Использование тэга
 требует осторожности — возможна ситуация, когда браузер уже сделал перевод строки за одно-два слова до того, как встретил ваш тэг <BR>. Это бывает в случае, если ширина окна программы просмотра читателя меньше, чем тот же параметр программы, с помощью которой вы тестировали ваш документ. При этом может получиться, что в строке посреди абзаца останется только одно слово, нарушая тем самым красоту компоновки документа.

Примечание

При использовании тэга
 для разбивки текста, обтекающего изображения или таблицы, можно задавать параметр CLEAR, прекращающий обтекание текста. Об этом можно прочитать в главах 3 и 4.

Тэги *<NOBR>* и *<WBR>*

Бывают ситуации, когда требуется выполнить операцию противоположного назначения — запретить перевод строки. Для этого существует тэг-контей-

нер <NOBR>. Текст, размеченный этим тэгом, будет гарантированно располагаться в одной строке, независимо от ее длины. Если при этом получающаяся строка будет выходить за пределы окна просмотра браузера, то появится горизонтальная полоса прокрутки.

Примечание

Для обеспечения неразрывности текста, располагаемого в ячейках таблиц, существует специальный параметр NOWRAP тэга <TD>. Об этом можно узнать в главе 4.

Размечая текст с помощью тэга неразрывной строки <NOBR> можно получить очень длинные строки. Чтобы этого избежать, можно указать браузеру читателя место возможного перевода строки, что будет выполнено только при необходимости (так называемый "мягкий" перевод строки). Это можно осуществить, поставив в нужном месте текста тэг <WBR> (Word BReak), который так же, как и тэг <BR>, не нуждается в закрывающем тэге.

Примечание

Тэг <WBR> вообще не поддерживается браузером Netscape. Браузер Microsoft Internet Explorer распознает этот тэг только в тексте, размеченном тэгами <NOBR>.

Заголовки внутри HTML-документа

Наряду с названием всего документа, на Web-странице могут использоваться заголовки для отдельных частей документа. Эти заголовки могут иметь шесть различных уровней (размеров) и представляют собой фрагменты текста, которые выделяются на экране при отображении страницы браузером.

Для разметки заголовков используются тэги <H1>, <H2>, <H3>, <H4>, <H5> и <H6>. Эти тэги требуют соответствующего закрывающего тэга. Заголовок с номером 1 является самым крупным (заголовок верхнего уровня), а с номером 6 — самым мелким. Тэги заголовка являются элементами уровня блока, поэтому с помощью них нельзя размечать отдельные слова текста для увеличения их размера. При использовании тэгов заголовков осуществляется вставка пустой строки до и после заголовка, поэтому тэгов абзаца или перевода строки здесь не требуется.

Тэги заголовков могут задаваться с параметром горизонтального выравнивания ALIGN. Возможные значения параметра совпадают с параметрами выравнивания тэга абзаца <P> (см. табл. 1.4).

Пример использования заголовков разного уровня с различным выравниванием (рис. 1.9):

```
<HTML>
<HEAD>
```

```
<TITLE>Примеры заголовков</TITLE>
</HEAD>
<BODY>
<H1>Заголовок размера 1</H1>
<H2>Заголовок размера 2</H2>
<H3 ALIGN=CENTER>Заголовок размера 3</H3>
<H4 ALIGN=RIGHT>Заголовок размера 4</H4>
<H5>Заголовок размера 5</H5>
<H6>Заголовок размера 6</H6>
Основной текст документа
</BODY>
</HTML>
```

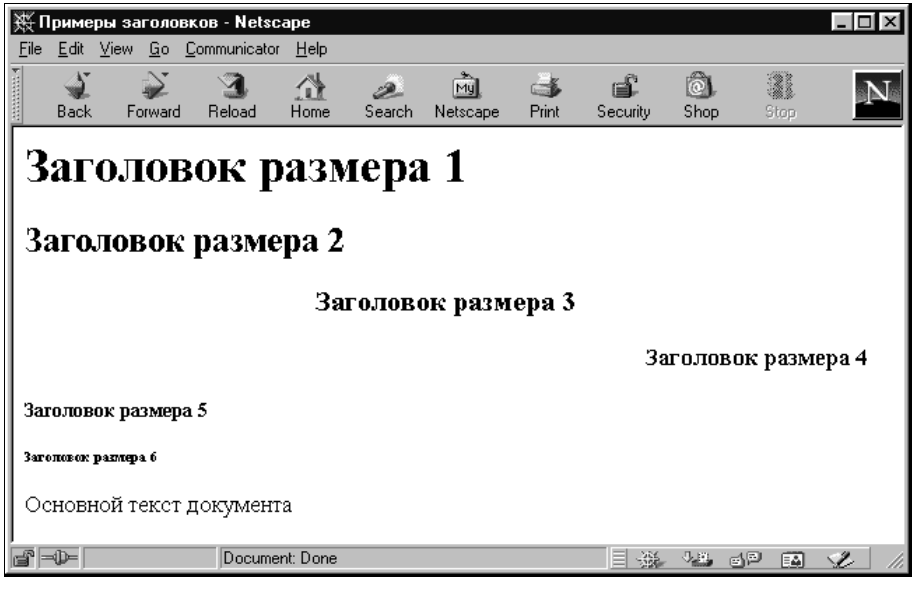


Рис. 1.9. Отображение заголовков различного размера

Горизонтальные линии

Другим методом разделения документа на части является проведение горизонтальных линий. Они визуально подчеркивают законченность той или иной области страницы. Сейчас часто используют рельефную, вдавленную линию, чтобы обозначить "объемность" документа.

Тэг `<HR>` позволяет провести рельефную горизонтальную линию в окне большинства программ просмотра. Этот тэг не является контейнером, поэтому не требует закрывающего тэга. До и после линии автоматически вставляется пустая строка. Параметры тэга `<HR>` представлены в табл. 1.5.

Таблица 1.5. *Параметры тэга* `<HR>`

Параметр тэга `<HR>`	Назначение
`ALIGN`	Выравнивает по краю или центру; имеет значения `LEFT`, `CENTER`, `RIGHT`
`WIDTH`	Устанавливает длину линии в пикселах или процентах от ширины окна браузера
`SIZE`	Устанавливает толщину линии в пикселах
`NOSHADE`	Отменяет рельефность линии
`COLOR`	Указывает цвет линии. Используется формат RGB или стандартное имя

Пример:

```
<HR ALIGN=CENTER WIDTH=50% NOSHADE>
```

В этом примере задается горизонтальная линия, которая занимает половину ширину окна просмотра и располагается посередине окна. Заметим, что параметры выравнивания имеют смысл только тогда, когда линия занимает не всю ширину окна.

Примечание

Браузер Netscape не позволяет использовать параметр `COLOR` тэга `<HR>`.

Использование предварительно отформатированного текста

Как видно в приведенных выше разделах, для разбивки текста по абзацам и обеспечения принудительного перевода строки следует пользоваться специальными тэгами. Однако бывают случаи, когда в HTML-документ необходимо включить текст, уже имеющий форматирование, выполненное традиционным способом при помощи символов перевода строки, необходимого количества пробелов, символов табуляции и т. д. Для решения таких задач предусмотрен специальный тэг-контейнер `<PRE>`, определяющий предварительно форматированный (преформатированный) текст.

Текст, размеченный тэгом `<PRE>`, будет отображаться в таком виде, как он выглядит в обычном текстовом редакторе. Для отображения всегда будет использоваться моноширинный шрифт. При этом вы сможете в большей степени контролировать вывод документа программой просмотра, правда, за счет некоторой потери в гибкости.

Одним из вариантов использования этого тэга являются таблицы, построенные без применения специальных тэгов разметки таблиц. Другим важным

применением является вывод на экран больших блоков программного кода (Java, C++ и т. п.), не позволяющий браузеру переформатировать их.

Текст внутри контейнера <PRE> может содержать элементы форматирования уровня текста, кроме следующих: <IMG>, <OBJECT>, <APPLET>, <BIG>, <SMALL>, <SUB>, <SUP>, <FONT>, <BASEFONT>. Недопустимо внутри преформатированного текста задавать элементы форматирования уровня блока, например, тэги заголовков. Тэг абзаца по логике вещей также не должен встречаться внутри преформатированного текста, однако если встречается, то будет реализовывать переход на новую строку (без образования пустой).

Примечание

Тэг <PRE> имеет необязательный параметр WIDTH, назначение которого — указывать браузеру максимальную длину строки преформатированного текста. В зависимости от этого значения браузер мог бы подобрать нужный шрифт и/или отступ для оптимального отображения преформатированного текста. В качестве значения по умолчанию предлагалось использовать 80. Другими рекомендованными значениями являются 40 и 132. Заметим, что читателям, имеющим значительный опыт работы со средствами вычислительной техники, эти числа говорят о многом. На деле же современные браузеры игнорируют значение параметра WIDTH.

Существуют еще несколько тэгов, решающих близкую по смыслу задачу. К ним относятся тэги <XMP>, <PLAINTEXT> и <LISTING>. Все три упомянутых тэга в спецификации HTML 4.0 отмечены как устаревшие. Это означает, что в будущих версиях браузеры прекратят их поддержку. Вместо этих тэгов рекомендуется использовать тэг <PRE>.

Тэг *<DIV>*

Тэг-контейнер <DIV> является элементом уровня блока, служащим для выделения фрагмента документа. Целью этого выделения является управление параметрами данного фрагмента, которое обычно выполняется с помощью назначения стилей. Приведем пример:

```
<DIV STYLE="color: green">
(Фрагмент документа)
</DIV>
```

В этом примере фрагмент HTML-документа обрамляется тэгами <DIV> и </DIV> для задания некоторых его свойств. В данном случае все текстовые элементы выделенного фрагмента будут отображаться зеленым (green) цветом. Аналогом тэга <DIV> уровня текста является элемент <SPAN>.

Заметим, что непосредственное назначение стилевых свойств отдельного фрагмента так, как это сделано в примере, использовать не желательно в соответствии с концепцией разделения структуры документа и его представ-

ления. Следует использовать таблицы стилей, речь о которых пойдет во второй части книги.

Тэг *<CENTER>*

Тэг-контейнер <CENTER> предназначен для горизонтального выравнивания всех элементов посередине окна просмотра браузера. Он имеет уровень блока и его полезно использовать для центрирования таких элементов, как, например, таблиц, так как они не могут быть центрированы назначением ALIGN=CENTER тэга <TABLE>.

По существу тэг <CENTER> является краткой формой следующей записи: <DIV ALIGN=CENTER>. Дальнейшее использование тэга <CENTER> по причинам, отмеченным в предыдущем разделе, также нежелательно.

Включение комментариев в документ

В HTML-документ можно включать комментарии, которые не будут видны читателю. Они могут состоять из произвольного числа строк и должны начинаться тэгом <!-- и заканчиваться тэгом -->. Все, что заключено внутри этих тэгов, при просмотре страницы не будет отображаться на экране.

Комментарии обычно используются авторами документа для заметок, предназначенных только для собственного использования. Заметим, что текст комментариев не отображается на экране браузера, однако передается вместе с документом и вполне может быть просмотрен читателями. Большинство браузеров предоставляют возможность просмотра исходного кода документа. Поэтому не следует включать в комментарии информацию, не предназначенную для чужих глаз. Это замечание может оказаться важным для разработчиков-программистов, привыкших писать комментарии в своих программах. Дело в том, что комментарии в программах, написанных на большинстве языков программирования, не попадают в результирующий код программы, получаемый в результате ее компиляции. Исходные же коды программ обычно хранятся только у авторов. Для языка же HTML нет понятия компиляции.

Комментарии в HTML применяются также для того, чтобы "спрятать" от браузера скрипты в случае, если он не в состоянии распознать их. Этот вопрос освещается во второй части книги.

Существует еще один тэг-контейнер для записи комментариев — <COMMENT>. Этот тэг используется редко, поскольку поддерживается только браузером Microsoft Internet Explorer. Некоторые проблемы использования этого тэга рассматриваются в главе 8.

Тэг *<BLOCKQUOTE>*

Бывают случаи, когда в текст HTML-документа необходимо включить какую-либо длинную цитату. Для выделения цитат из основного текста суще-

ствует тэг <BLOCKQUOTE>. Он является контейнером и может содержать любые тэги форматирования.

В отличие от тэга <Q>, предназначенного для выделения коротких цитат в строке текста, <BLOCKQUOTE> является тэгом уровня блока. Текст, размеченный данным тэгом, при отображении отделяется от основного текста пустыми строчками и, как правило, выводится с небольшим отступом вправо.

Пример отображения длинной цитаты приведен на рис. 1.10.

Примечание

Иногда полезно знать не только назначение тэга, но и принципы его реализации браузерами. Так, по существу единственное действие, которое выполняют браузеры, встретив тэг <BLOCKQUOTE>, заключается в отделении текста пустыми строками и сдвиге его вправо. Эти сведения позволяют использовать данный элемент не только по прямому назначению, но и просто для тех случаев, когда требуется отобразить фрагмент текста с отступом. Конечно, это нарушает логику структурирования документа, однако на деле такой подход часто применяется. В частности, редактор HTML-документов Netscape Composer, входящий в состав пакета Netscape Communicator, имеет в панели инструментов кнопки "увеличить отступ" и "уменьшить отступ". Каждое нажатие кнопки увеличения отступа реализуется путем обрамления выделенного текста парой тэгов <BLOCKQUOTE> и </BLOCKQUOTE>. Просматривая в дальнейшем исходный HTML-код созданного таким путем документа, можно долго вспоминать, что же вы там цитировали.

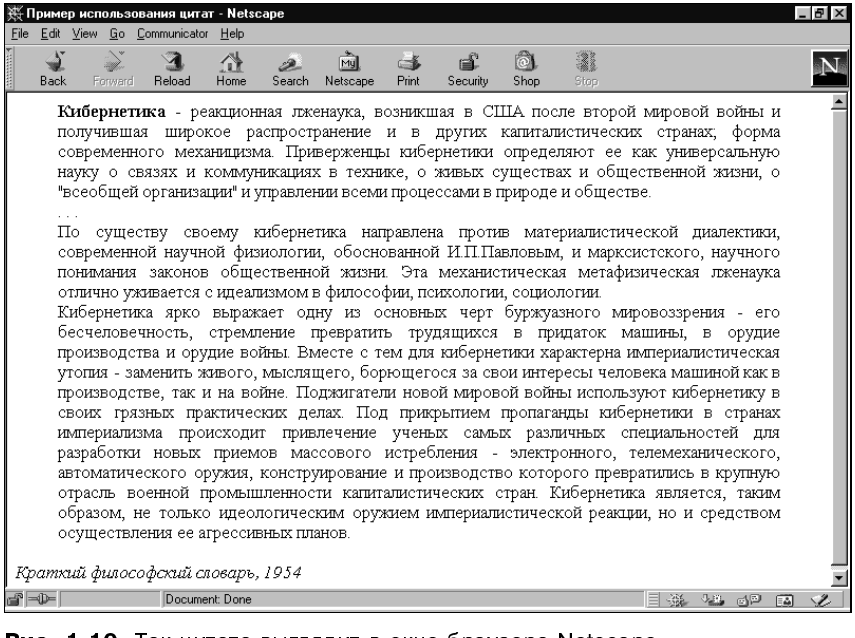


Рис. 1.10. Так цитата выглядит в окне браузера Netscape

Тэг *<ADDRESS>*

Тэг `<ADDRESS>` применяется для идентификации автора документа и для указания адреса автора. Сюда же обычно помещаются сведения об авторских правах. Этот элемент располагается либо в начале, либо в самом конце документа.

Часто в этом элементе указывают дату создания и последнего обновления документа. Это позволяет читателям определить, знакомились ли они ранее с версией, которую просматривают.

Текст, заключенный между этими тэгами, обычно показывается браузерами курсивом.

Специальные символы

Некоторые специальные символы не входят в базовую часть таблицы кодов ASCII. К ним относятся буквы алфавитов части европейских языков, математические и некоторые другие символы. Некоторые символы, непосредственно введенные в HTML-документ, будут интерпретированы не так, как задумал автор. К ним относятся символы "<" и ">", обычно используемые для указания тэгов языка.

В таких случаях можно вводить нужные символы в ваш HTML-документ при помощи специальных кодов. Эти коды состоят из символа амперсанда (&) и следующим за ним именем символа или его десятичным или шестнадцатеричным значением. Заканчиваться специальный символ должен знаком "точка с запятой".

В спецификации HTML приводятся целые таблицы со специальными символами и их значениями. На сегодняшний день браузеры распознают лишь небольшое количество специальных символов, поэтому приводить эти таблицы полностью излишне. Отметим лишь некоторые символы, употребление которых актуально и обеспечено поддержкой браузеров (табл. 1.6).

Таблица 1.6. *Специальные символы HTML*

Запись специального символа	Назначение
&lt	Знак "меньше"
&gt	Знак "больше"
	Неразрывный пробел
&copy	Знак copyright
&amp	Амперсанд
&quot	Знак "кавычки"

Все символами могут быть также заданы своими кодами. Например, символ неразрывного пробела имеет код 160. Он может записываться в десятичном виде как .

Ссылки на другие документы и файлы

Одним из важнейших понятий для HTML-документов являются ссылки. Само название — HTML, язык разметки гипертекста, указывает на принцип организации таких документов. Вы, наверное, уже обратили внимание на ссылки, часто использующиеся в этой книге. Они выполняют ту же роль, что и ссылки на Web-странице, хотя и далеки от последних с точки зрения технического совершенства. Они относят вас к разделу книги, который может быть полезен именно в данный момент. Без этих ссылок вам бы пришлось долго перелистывать книгу в поисках нужной информации.

Значение ссылок в мире Интернета трудно переоценить. Читая книгу, вы имеете ее всегда под рукой. Работая в Web-пространстве, вы часто понятия не имеете, где находится та или иная нужная вам страница. Поэтому ссылки здесь являются единственной возможностью перейти от одного документа к другому.

Гипертекстовый документ — это документ, содержащий ссылки на другие документы, позволяющие при помощи нажатия кнопки мыши быстро перемещаться от одного документа к другому. Часто подобные ссылки можно увидеть и в файлах помощи современных программных продуктов. За основу гипертекста взят принцип организации энциклопедических словарей, в которых во многих статьях есть ссылки на другие.

Существует много типов мультимедийных объектов, которые могут быть размещены на Web-странице. В современных гипертекстовых документах в дополнение к самому тексту часто используют разнообразную графику, видео- и аудиообъекты, а в качестве ссылок часто применяют изображения.

Организация ссылок

Ссылка состоит из двух частей. Первая из них — это то, что вы видите на Web-странице; она называется *указатель ссылки* (anchor). Вторая часть, дающая инструкцию браузеру, называется адресной частью ссылки (URL-адрес). Когда вы щелкаете мышью по указателю ссылки, браузер загружает документ, адрес которого дается URL-адресом. Ниже рассмотрены правила построения отдельных элементов ссылок.

Указателем ссылки может быть слово, группа слов или изображение. Внешний вид ссылки зависит от его типа, способов создания и установок программы просмотра читателя. Указатели бывают двух типов — текстовые и графические.

Текстовые указатели обычно представляют собой слово или несколько слов, выделенных на экране подчеркиванием. Цвет текстового указателя может регулироваться автором и установками программы просмотра.

Приведем пример записи для текстового указателя ссылки:

```
<A HREF="example.html">Этот текст является указателем ссылки</A>
```

В качестве ссылки можно использовать графическое изображение. По принципу действия графические ссылки ничем не отличаются от текстовых. Они не подчеркиваются и не выделяются цветом, а для их выделения браузеры обычно вокруг такого изображения рисуют рамку. Пример графического указателя ссылки:

```
<A HREF="example.html"><IMG SRC="picture.gif"></A>
```

Более подробно о графических указателях рассказывается в главе 3. Специальные возможности создания изображений, фрагменты которого указывают на различные документы, подробно описываются в главе 6.

Второй частью ссылки является URL-адрес. Это не что иное, как адрес Web-страницы, которая будет загружена при щелчке мышью на указателе. Указание адреса может быть относительным или абсолютным.

Если в URL-адресе не указывается полный путь к файлу, то такая ссылка называется относительной. В этом случае определение местоположения файлов выполняется с учетом местоположения документа, в котором имеется такая ссылка. Например, если браузер загрузил страницу, находящуюся по адресу **http://www.mysite.com/page**, то относительный указатель **/picture** подразумевает адрес **http://www.mysite.com/page/picture**, т. е. подкаталог, расположенный на той же машине.

Примечание

Относительный указатель работает по-другому, если в HTML-документе используется тэг <BASE>. Ниже в данной главе показывается, что в этом случае указатель дает адрес относительно URL-адреса, определенного в тэге <BASE>.

Относительные указатели удобны в использовании. Намного проще вставить только имя файла, а не весь длинный URL-адрес. Они также позволяют вам перемещать файлы в пределах вашего сервера без больших изменений в межстраничной адресации.

URL-адрес, полностью определяющий компьютер, каталог и файл, называется абсолютным. В отличие от относительных, абсолютные указатели могут ссылаться на файлы, расположенные на других компьютерах.

Правила записи ссылок

Для организации ссылки необходимо сообщить браузеру, что является указателем ссылки, а также указать адрес документа, на который вы ссылаетесь. Оба действия выполняются при помощи тэга <A>.

Тэг <A>

Тэг <A> имеет единственный параметр HREF, значением которого является URL-адрес. Указатель может быть как относительным, так и абсолютным, например, **http://www.server.com/home/index.htm**. Этот тэг является контейнером, поэтому необходимо поставить закрывающий тэг </A>:

```
<A HREF=URL-адрес>Текстовый указатель ссылки</A>
```

Указатель ссылки может быть относительным или абсолютным. Для облегчения работы с относительными указателями ссылок введен тэг <BASE>. Он располагается в начале документа в разделе HEAD и содержит URL-адрес, относительно которого в документе построена вся адресация. Это указание влияет на любой тэг документа, в котором используется относительная адресация. Если тэг <BASE> отсутствует, то адресация строится относительно адреса текущего документа.

Внутренние ссылки

Кроме ссылок на другие документы, часто бывает полезно включить ссылки на разные части текущего документа. Например, большой документ читается лучше, если он имеет оглавление со ссылками на соответствующие разделы.

Для построения внутренней ссылки сначала нужно создать указатель, определяющий место назначения. Например, если вы хотите сделать ссылку на текст определенной главы документа, нужно разместить там указатель и дать ему имя при помощи параметра NAME тэга <A>. При этом параметр HREF не используется, и браузер не выделяет содержимое тэга <A>. Например:

```
<A NAME=chapter_5> </A>
```

Обратите внимание, что в приведенном примере отсутствует содержимое тэга <A>. Обычно именно так и делают, поскольку здесь нет необходимости как-то выделять текст, а требуется лишь указать местоположение.

После того как место назначения определено, можно приступить к созданию ссылки на него. Для этого, вместо указания в параметре HREF адреса документа, как это делалось ранее, поместим туда имя ссылки с префиксом #, говорящим о том, что это внутренняя ссылка.

```
<A HREF="#chapter_5">Глава 5</A>
```

Теперь, если пользователь щелкнет кнопкой мыши на словах "Глава 5", браузер выведет соответствующую часть документа в окне просмотра.

Ссылки на документы различных типов

Когда пользователь щелкает мышью на ссылке, указывающей на другую Web-страницу, она выводится непосредственно в окне браузера. Если же ссылка указывает на документ иного типа, программа просмотра принимает

документ и затем решает, что с ним делать дальше. Следующими действиями браузера могут быть:

- Браузер знает этот тип документа и умеет с ним обращаться. Например, если вы создали ссылку на графический файл формата GIF и пользователь щелкнул мышью на этой ссылке, его программа просмотра очистит окно и загрузит указанное изображение. В некоторых случаях браузер может дополнительно использовать подключаемый программный модуль (plug-in), без которого задача не была бы решена.
- Браузер не распознает тип принятого документа и не знает, что с ним делать дальше. В этом случае он обратится к вспомогательным программам, имеющимся на машине пользователя. Если подходящая программа найдется, браузер запустит ее и передаст ей полученный документ для обработки. Например, если пользователь щелкнет на ссылке на видеофайл формата AVI, браузер загрузит файл, найдет программу для демонстрации AVI-файлов и запустит ее. Видеофайл будет показан в дополнительном небольшом окне.

Ссылки на другие ресурсы Интернета

Web-пространство является лишь частью сети Интернет. Другие ресурсы начали свое существование задолго до рождения WWW, поэтому накопили уже много достойной внимания информации и имеют достаточно большую аудиторию. Поэтому, разрабатывая свою персональную страницу или документ, вы, возможно, захотите включить ссылки и на другие ресурсы.

Ресурсы Интернета весьма разнообразны по форме и содержанию. Хотя HTML предполагает возможность создания своих собственных версий этих ресурсов с помощью механизма обработки данных форм, есть более простые пути к взаимодействию с системами UseNet, Telnet, FTP, e-mail и другими. Например, вы можете создать документ с использованием различных тэгов форм, текстовых элементов и кнопкой для отправки электронного письма на ваш адрес. Однако будет намного проще для связи указать лишь свой адрес электронной почты. В этом случае упрощается обновление страницы, которое не будет связано с изменениями форм. Кроме того, многие браузеры имеют встроенную поддержку некоторых ресурсов, что дает возможность сократить время на установление связи с ними.

Используя ранее какой-либо ресурс сети, вы, скорее всего, захотите сохранить эту связь. Если у вас уже работает система Gopher с автоматическим обновлением информации, нет смысла переделывать его под Web-сайт. Легче создать ссылку на него с вашей новой Web-страницы.

Самой популярной деятельностью в Интернете является обмен электронными письмами. Пользователей этого ресурса намного больше, чем любого другого. Причина очень простая: если вы подключены к Интернету, у вас обязательно есть свой адрес электронной почты. Большинство современных

программ для обмена электронными сообщениями имеют дружественный интерфейс и просты в использовании.

Если вас интересует отклик читателей на содержание вашего документа, вы захотите поместить на странице свой адрес электронной почты. Это может быть также полезно для сообщений о неработающих ссылках и других проблемах, связанных с просмотром вашей страницы.

Создание ссылки на электронную почту так же просто, как и на другую страницу. Для этого вместо URL-адреса следует указать адрес электронной почты, предварив его словом mailto:.

```
<A HREF="mailto:sergeev@mail.ifmo.ru">Присылайте ваши отзывы
  и предложения</A>.
```

Эта ссылка не будет ничем отличаться от остальных гипертекстовых ссылок вашего документа. То же самое можно сказать и о ссылках на другие ресурсы Интернета. После щелчка мышью на ссылке на ваш адрес браузер откроет собственное окно для работы с электронной почтой. Интерфейсы могут быть разными, но большинство программ автоматически вводят в сообщение адрес и имя пользователя и напоминают о необходимости заполнения строки "Subject".

В заключение приведем пример HTML-кода, в котором используется ряд тэгов, описанных в данной главе:

```
<HTML>
<HEAD>
<TITLE>Личная страница Александра Сергеева</TITLE>
</HEAD>
<BODY>
<H1 ALIGN=CENTER>Добро пожаловать!</H1>
<HR>
Основное содержание страницы
<HR>
<ADDRESS>
Последнее обновление выполнено 01 ноября 1999 г.
</ADDRESS>
<A HREF="mailto:sergeev@mail.ifmo.ru">
Присылайте ваши отзывы и предложения</A>
</BODY>
</HTML>
```

В приведенном примере есть ссылка на электронную почту. При наведении курсора мыши на указатель данной ссылки этот адрес будет виден в нижней части окна браузера (рис. 1.11).

Ссылки на другие ресурсы Интернета записываются аналогично. Правила записи приведены в табл. 1.7.

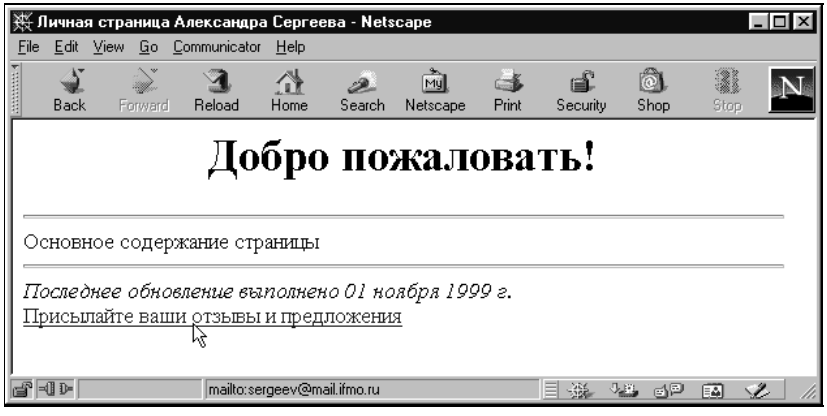


Рис. 1.11. Отображение документа примера в окне браузера

Таблица 1.7. *Ссылки на ресурсы Интернета*

Ресурсы Интернета	Формат ссылки	Пример записи ссылки
Web-страница	`http://sitename`	`http://www.mysite.com/`
e-mail	`mailto:address`	`mailto:me@mysite.com`
Newsgroup	`news:newsgroupname`	`news:news.newusers.questions`
FTP	`ftp://sitename`	`ftp://ftp.mysite.com/`
Gopher	`gopher://sitename`	`gopher://gopher.mysite.com/`
WAIS	`wais://sitename`	`wais://wais.mysite.com/`
TelNet	`telnet://sitename`	`telnet://bbs.mysite.com/`

Прочие тэги

Чтобы полностью охватить весь спектр существующих или использовавшихся ранее тэгов языка HTML, в этом разделе лишь упомянем те тэги, описание которых намеренно не дается на страницах данной книги. В основном это редко используемые тэги, применение которых в большинстве случаев ограничивается одним единственным браузером. Тэги такого рода вряд ли следует рекомендовать к использованию, хотя их разработчики, естественно будут пропагандировать их, частенько не указывая ограниченности применения. Возможно, в будущем какие-то из них станут общепринятыми, а может быть наоборот, устареют, не приобретя популярности.

Перечисление тэгов в данном разделе преследует единственную цель — отразить их существование в указателе, приводимом в конце книги. Тогда чи-

татель, выполняя поиск интересующего его тэга по алфавитному указателю, попадет на данную страницу.

Тэги, распознаваемые только браузером Netscape:

`<LAYER>`, `<ILAYER>`, `<MULTICOL>`, `<KEYGEN>`, `<SPACER>`.

Тэги, распознаваемые только браузером Microsoft Internet Explorer:

`<FIELDSET>`, `<LEGEND>`, `<MARQUEE>`.

Тэг, распознаваемый только браузером Mosaic: `<SOUND>`.

Редко используемые тэги, дальнейшее употребление которых не рекомендуется:

`<ISINDEX>`, `<NEXTID>`.

Тэги, актуальность которых крайне мала: `<BDO>`.

ГЛАВА 2

Списки

В языке HTML предусмотрен специальный набор тэгов для представления информации в виде списков. Списки являются одним из наиболее часто употребляемых форм представления данных как в электронных документах, так и в печатных. Со списками мы встречаемся практически ежедневно, — это может быть список необходимых покупок в магазине, учеников в классе или просто дел, которые необходимо выполнить. Возможность организации списковых структур имеется во многих текстовых редакторах, в частности, мощный текстовый процессор Microsoft Word обладает удобными средствами форматирования списков различного вида (возможности создания HTML-списков при помощи Microsoft Word обсуждаются в главе 8). Приведем ряд случаев, для которых использование списков довольно удобно:

- Объединение фрагментов информации в единую структуру для придания удобочитаемого вида.
- Описание сложных пошаговых процессов.
- Расположение информации в стиле оглавления, пункты которого указывают на соответствующие разделы документа.

Заметим, что приведенные выше пункты как раз и организованы в виде списковой структуры.

В языке HTML предусмотрены следующие основные типы списков: маркированный, нумерованный и список определений. Для реализации списков различных типов используются следующие тэги: `<UL>`, `<OL>`, `<DL>`, `<DIR>`, `<MENU>`. С помощью различных типов встроенных в документ списков могут быть реализованы самые разные возможности, описанию которых и посвящена данная глава. Рассматриваются особенности построения списков различных типов, а также применения вложенных друг в друга списков.

Маркированный список

Одним из типов списков, реализованных в языке HTML, является маркированный список. Иначе списки такого типа называют ненумерованными или

неупорядоченными. Последнее название часто используется как формальный перевод названия соответствующего тэга <UL>, с помощью которого и организуются списки такого типа в HTML-документах (UL — Unordered List, неупорядоченный список).

В маркированном списке для выделения его элементов используются специальные символы, называемые маркерами списка (часто их называют буллетами, что является формальным озвучением английского термина bullet — пуля). Вид маркеров списка определяется браузером, причем при создании вложенных списков браузеры автоматически разнообразят вид маркеров различного уровня вложенности.

Тэги *<UL>* и *<LI>*

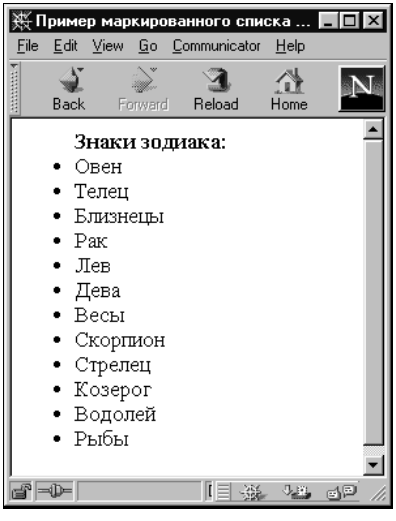
Для создания маркированного списка необходимо использовать тэг-контейнер <UL> </UL>, внутри которого располагаются все элементы списка. Открывающий и закрывающий тэги списка обеспечивают перевод строки до и после списка, отделяя, таким образом, список от основного содержимого документа, поэтому здесь нет необходимости использовать тэги абзаца <P> или принудительного перевода строки <BR>.

Каждый элемент списка должен начинаться тэгом <LI> (LI — List Item, элемент списка). Тэг <LI> не нуждается в соответствующем закрывающем тэге, хотя его наличие в принципе не возбраняется. Браузеры обычно при отображении документа начинают каждый новый элемент списка с новой строчки.

Приведенных сведений достаточно для построения элементарного маркированного списка. Приведем пример HTML-документа, использующего маркированный список, отображение которого браузером показано на рис. 2.1.

```
<HTML>
<HEAD>
<TITLE>Пример маркированного списка</TITLE>
</HEAD>
<BODY>
<UL>
<B>Знаки зодиака:</B>
       <LI>Овен
       <LI>Телец
       <LI>Близнецы
       <LI>Рак
       <LI>Лев
       <LI>Дева
       <LI>Весы
       <LI>Скорпион
       <LI>Стрелец
```

```
        <LI>Козерог
        <LI>Водолей
        <LI>Рыбы
</UL>
</BODY>
</HTML>
```



Заметим, что кроме элементов списка, отмечаемых тэгом <LI>, могут присутствовать и другие HTML-элементы. В приведенном выше примере одним из таких элементов является обычный текст, не являющийся пунктом списка, а играющий роль его заголовка.

Рис. 2.1. Отображение браузером маркированного списка

Примечание

В некоторых учебниках по языку HTML встречается указание, что для задания заголовка списка следует применять тэг-контейнер <LH> (LH — List Header, заголовок списка). В настоящее время этот тэг не распознается ни одним из распространенных браузеров и не входит в спецификацию HTML. Таким образом, его применение становится бессмысленным, хотя и не приведет к каким-либо ошибкам.

В тэге <UL> могут быть указаны два параметра: COMPACT и TYPE.

Параметр COMPACT записывается без значений и применяется для указания браузеру, что данный список следует выводить в компактном виде. Например, может быть уменьшен шрифт или расстояние между строчками списка и т. д.

Примечание

В настоящее время наличие параметра COMPACT в тэге <UL> никак не влияет на отображение списков ведущими браузерами. Поэтому применение данного параметра бессмысленно, тем более что его употребление не рекомендуется спецификацией HTML 4.0.

Параметр TYPE может принимать следующие значения: disc, circle и square. Этот параметр используется для принудительного задания вида маркеров

списка. Конкретный вид маркера будет зависеть от используемого браузера. Типичными вариантами отображения являются следующие:

`TYPE = disc` — маркеры отображаются закрашенными кружками;

`TYPE = circle` — маркеры отображаются не закрашенными кружками;

`TYPE = square` — маркеры отображаются закрашенными квадратиками.

Пример записи: `<UL TYPE = circle>`.

Значением, используемым по умолчанию, является `TYPE = disc`. Для вложенных маркированных списков на первом уровне по умолчанию используется значение `disc`, на втором — `circle`, на третьем и далее — `square`. Именно так делается в последних версиях браузеров Netscape и Internet Explorer. Заметим, что иные браузеры могут иначе отображать маркеры. Например, в спецификации HTML 4.0 для вида маркера, отображаемого при значении `TYPE = square`, указывается незакрашенный квадратик (square outline).

Параметр `TYPE` с теми же значениями может употребляться для указания вида маркеров отдельных элементов списка. Для этого параметр `TYPE` с соответствующим значением разрешено указывать в тэге элемента списка `<LI>`.

Пример записи: `<LI TYPE = circle>`.

Примечание

Браузеры по-разному интерпретируют указание вида маркера для отдельного элемента списка. Браузер Netscape изменяет вид маркера для данного и всех последующих, пока не встретится очередное переопределение вида маркера. Браузер Internet Explorer изменяет вид маркера только для данного элемента.

Графические маркеры списка

В качестве маркеров списка можно использовать графические изображения, что широко применяется для создания привлекательных, красиво оформленных HTML-документов. На самом деле такая возможность не предоставляется непосредственно языком HTML, а реализуется несколько искусственно. Это вовсе не означает, что так делать не рекомендуется или предосудительно, а лишь означает, что здесь не будут применяться никакие специальные языковые конструкции HTML.

Чтобы понять идею, необходимо разобраться в механизме реализации списков на HTML-страницах. Оказывается, что тэг списка `<UL>` (как, впрочем, и тэги списков других типов, рассматриваемых ниже) выполняет единственную задачу — указывает браузеру, что вся информация, располагаемая после данного тэга должна отображаться со сдвигом вправо (отступом) на некоторую величину. Тэги `<LI>`, указывающие на отдельные элементы списка, обеспечивают вывод стандартных маркеров элементов списка.

Если же нам требуется построить список с графическими маркерами, то можно вообще обойтись без тэгов <LI>. Достаточно будет перед каждым элементом списка вставить желаемое графическое изображение. Единственной задачей, которую нужно при этом решить, будет отделение элементов списка друг от друга. Для этого можно использовать тэги абзаца <P> или принудительного перевода строки <BR>. Пример реализации списка с графическими маркерами, отображение которого представлено на рис. 2.2, показан ниже:

```
<HTML>
<HEAD>
<TITLE>Маркированный список</TITLE>
</HEAD>
<BODY>
<UL>
<B>Знаки зодиака:</B><BR>
       <IMG SRC="Green_ball.gif"> Овен<BR>
       <IMG SRC="Green_ball.gif"> Телец<BR>
       <IMG SRC="Green_ball.gif"> Близнецы<BR>
       <IMG SRC="Green_ball.gif"> Рак<BR>
       <IMG SRC="Green_ball.gif"> Лев<BR>
       <IMG SRC="Green_ball.gif"> Дева<BR>
       <IMG SRC="Green_ball.gif"> Весы<BR>
       <IMG SRC="Green_ball.gif"> Скорпион<BR>
       <IMG SRC="Green_ball.gif"> Стрелец<BR>
       <IMG SRC="Green_ball.gif"> Козерог<BR>
       <IMG SRC="Green_ball.gif"> Водолей<BR>
       <IMG SRC="Green_ball.gif"> Рыбы
</UL>
</BODY>
</HTML>
```

Рис. 2.2. Маркированный список с графическими маркерами

В приведенном примере в качестве маркера элементов списка используется графический файл Green_ball.gif. Заметим, что использование графики на HTML-страницах может значительно увеличить объем передаваемой информации. Однако в данном случае это увеличение крайне незначительно. Здесь для всех маркеров используется один и тот же файл,

который будет передан только один раз. Размеры файла, содержащего маленькое изображение, также крайне незначительны.

Примечание

Методы создания списков с графическими маркерами обсуждаются в свою очередь в главе 8.

Нумерованный список

Другим типом списков, реализованных в языке HTML, является нумерованный список. Иначе списки такого типа называют упорядоченными. Последнее название часто используется как формальный перевод названия соответствующего тэга `<OL>`, с помощью которого и организуются списки такого типа в HTML-документах (OL — Ordered List, упорядоченный список).

Списки данного типа обычно представляют собой упорядоченную последовательность отдельных элементов. Отличием от маркированных списков является то, что в нумерованном списке перед каждым его элементом автоматически проставляется порядковый номер. Вид нумерации зависит от браузера и может задаваться параметрами тэгов списка. В остальном реализация нумерованных списков во многом похожа на реализацию маркированных списков.

Тэги *<OL>* и *<LI>*

Для создания нумерованного списка следует использовать тэг-контейнер `<OL> </OL>`, внутри которого располагаются все элементы списка. Открывающий и закрывающий тэги списка обеспечивают перевод строки до и после списка, отделяя таким образом список от основного содержимого документа.

Как и для маркированного списка, каждый элемент нумерованного списка должен начинаться тэгом `<LI>`.

Приведем пример HTML-документа, использующего нумерованный список, отображение которого браузером показано на рис. 2.3.

```
<HTML>
<HEAD>
<TITLE>Пример нумерованного списка</TITLE>
</HEAD>
<BODY>
<OL>
<B>Наиболее яркие звезды, видимые с Земли:</B>
       <LI>Сириус
       <LI>Канопус
```

```
        <LI>Арктур
        <LI>Альфа Центавра
        <LI>Вега
        <LI>Капелла
        <LI>Ригель
        <LI>Процион
        <LI>Ахернар
        <LI>Бета Центавра
        <LI>Бетельгейзе
        <LI>Альдебаран
        <BR>. . .
        <LI value=58>Мицар
        <BR>. . .
        <LI value=75>Полярная
</OL>
</BODY>
</HTML>
```

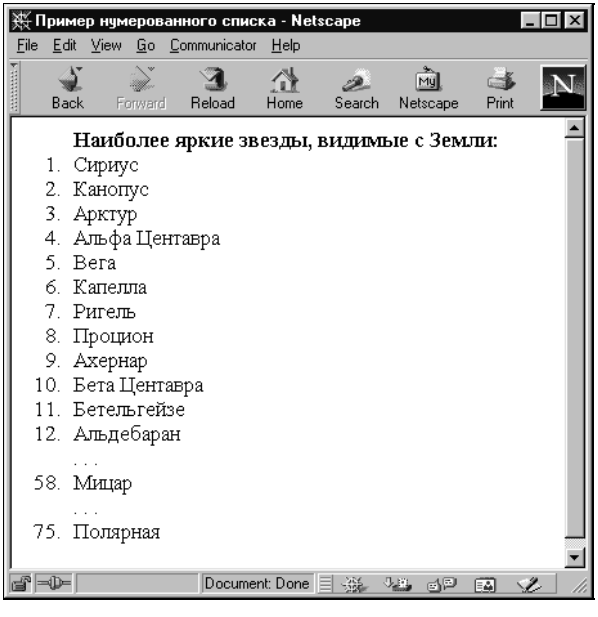


Рис. 2.3. Нумерованный список

В тэге `<OL>` могут быть указаны следующие параметры: `COMPACT`, `TYPE` и `START`.

Параметр `COMPACT` имеет тот же смысл, что и у маркированных списков. Параметр `TYPE` используется для задания вида нумерации списка. Может принимать следующие значения:

`TYPE = A` — задает маркеры в виде прописных латинских букв;

`TYPE = a` — задает маркеры в виде строчных латинских букв;

TYPE = I — задает маркеры в виде больших римских цифр;

TYPE = i — задает маркеры в виде маленьких римских цифр;

TYPE = 1 — задает маркеры в виде арабских цифр.

По умолчанию всегда используется значение TYPE = 1, т. е. нумерация при помощи арабских цифр. Это касается и вложенных нумерованных списков. Здесь, в отличие от маркированных списков, браузеры по умолчанию не делают различной нумерацию на различных уровнях вложенности списков. Заметим, что после номера элемента списка всегда дополнительно выводится знак "точка".

Параметр TYPE с теми же значениями может употребляться для указания вида нумерации отдельных элементов списка. Для этого параметр TYPE с соответствующим значением разрешено указывать в тэге элемента списка <LI>.

Пример записи: <LI TYPE = A>.

Параметр START тэга <OL> позволяет начать нумерацию списка не с единицы. В качестве значения параметра START всегда должно указываться натуральное число, вне зависимости от вида нумерации списка. Приведем пример:

<OL TYPE = A START=5>.

Такая запись определяет нумерацию списка с прописной латинской буквы "E". Для других видов нумерации запись START=5 задаст нумерацию, соответственно, с числа "5", римской цифры "V" и т. д.

Изменение вида нумерации списка и значений номеров допустимо производить и для любого элемента списка. Тэг <LI> для нумерованных списков разрешает использовать параметры TYPE и VALUE. Параметр TYPE может принимать такие же значения, как и для тэга <OL>.

Пример записи: <LI TYPE = A>.

Примечание

Браузеры по-разному интерпретируют указание вида нумерации для отдельного элемента списка. Браузер Netscape изменяет вид нумерации для данного элемента и всех последующих, пока не встретится очередное переопределение. Браузер Internet Explorer изменяет вид номера только для данного элемента.

Значение параметра VALUE тэга <LI> позволяет изменить номер данного элемента списка. При этом изменяется нумерация и всех последующих элементов. Типичным применением являются списки с пропуском некоторых элементов. Пример такого списка был приведен выше (рис. 2.3). В нем дается упорядоченный список наиболее ярких звезд, в котором на 58 и 75 местах расположены звезды, хорошо видимые в наших широтах (Мицар — наиболее яркая звезда созвездия Большая Медведица, а Полярная звезда — Малой Медведицы).

Приведем еще один оригинальный пример использования нумерации различных видов. В приводимом ниже HTML-коде заданы три списка с различной нумерацией. Для удобства просмотра каждый из списков помещен в отдельную ячейку таблицы. Все три списка идентичны и различаются только видом нумерации: в первом столбце таблицы — арабские цифры, во втором — римские, а в третьем нумерация ведется латинскими буквами. Обратим внимание, что элементы списка пусты, т. е. после любого тэга `<LI>` нет никаких данных. Пример такого рода можно использовать в качестве таблицы соответствия между записью чисел арабскими и римскими цифрами. Оказывается, что любой браузер, поддерживающий списки, можно использовать в качестве генератора такой таблицы (рис. 2.4), стоит лишь набрать приводимый HTML-код. Нумерация римскими цифрами правильно работает вплоть до значения 3999. Изучая правый столбец, можно понять, как выполняется нумерация латинскими буквами. По исчерпании однобуквенной нумерации (от A до Z) в качестве следующего номера берется первый двухбуквенный номер — AA и т. д.

```
<HTML>
<HEAD>
<TITLE>Использование различного типа нумерации в списках</TITLE>
</HEAD>
<BODY>
<TABLE BORDER=1 WIDTH=100% CELLPADDING=15>
<TR>
  <TD>
  <OL>
  <LI><LI>
  <BR>. . .
  <LI VALUE=1991><LI><LI><LI><LI><LI><LI><LI><LI><LI><LI><LI><LI><LI>
  </OL>
  </TD>

  <TD>
  <OL TYPE=I>
  <LI><LI>
  <BR>. . .
  <LI VALUE=1991><LI><LI><LI><LI><LI><LI><LI><LI><LI><LI><LI><LI><LI>
  </OL>
  </TD>

  <TD>
  <OL TYPE=A>
  <LI><LI>
  <BR>. . .
  <LI VALUE=1991><LI><LI><LI><LI><LI><LI><LI><LI><LI><LI><LI><LI><LI>
```

```
  </OL>
  </TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

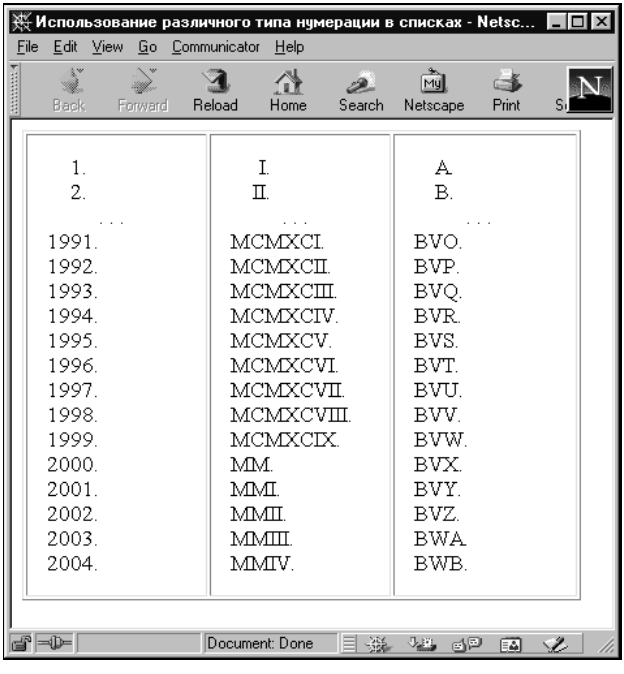


Рис. 2.4. Различные типы нумерации HTML-списков

Список определений

Списки определений, также называемые словарями определений специальных терминов, являются особенным видом списков. В отличие от других типов списков, каждый элемент списка определений всегда состоит из двух частей. В первой части элемента списка записывается определяемый термин, а во второй части — текст в форме словарной статьи, раскрывающий значение термина.

Списки определений задаются с помощью тэга-контейнера `<DL>` (Definition List). Внутри контейнера тэгом `<DT>` (Definition Term) помечается определяемый термин, а тэгом `<DD>` (Definition Description) — абзац с его определением. Для тэгов `<DT>` и `<DD>` можно не записывать соответствующие закрывающие тэги.

В общем, список определений записывается следующим образом:

```
<DL>
<DT>Термин
<DD>Определение термина
</DL>
```

В тексте после тэга <DT> не могут использоваться элементы уровня блока, такие как, например, тэги абзаца <P> или заголовков <H1>—<H6>. Как правило, текст определяемого термина должен располагаться в одной строке. Текст, содержащий определение термина, выводится, начиная со следующей строки (или через строку для некоторых браузеров) после определения термина с отступом вправо. В информации, помещенной после тэга <DD>, могут располагаться элементы уровня блока. Отсюда следует, в частности, что списки определений могут быть вложенными.

В тэге <DL> может быть указан параметр COMPACT, назначение которого аналогично другим спискам, описываемым выше.

Приведем пример HTML-документа, в котором использован список определений:

```
<HTML>
<HEAD>
<TITLE>Пример списка определений</TITLE>
</HEAD>
<BODY>
<DL>
<CENTER>
<H3>Классификация типичных темпераментов человека,<BR>основанная
    на воззрениях Гиппократа</H3>
</CENTER>
  <DT>Флегматик
  <DD>Пассивный, очень трудоспособный, медленно
приспосабливающийся;<BR>настроение устойчивое, мало поддается внешнему
влиянию;<BR>вялость эмоциональных реакций и медлительность в волевой
деятельности<BR><BR>

  <DT>Сангвиник
  <DD>Активный, энергичный, легко приспосабливающийся;<BR>живость
и подвижность эмоциональных реакций, быстрота и сила волевых
проявлений<BR><BR>

  <DT>Холерик
  <DD>Активный, очень энергичный, настойчивый;<BR>порывистость и сила
эмоциональных реакций, бурные волевые проявления<BR><BR>

  <DT>Меланхолик
  <DD>Пассивный, легко утомляющийся, тяжело
приспосабливающийся;<BR>слабость волевых проявлений и преобладание
подавленного настроения, неуверенность в себе
</DL>
```

```
</BODY>
</HTML>
```

Отображение приведенного HTML-документа в браузере показано на рис. 2.5.

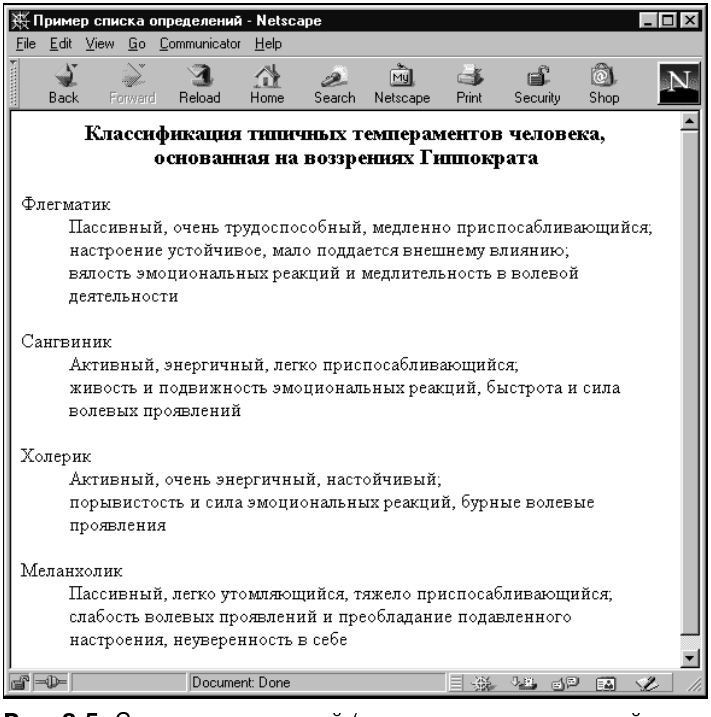


Рис. 2.5. Список определений (напоминает группу статей в словаре)

Списки типа *<DIR>* и *<MENU>*

Списки типа <DIR> и <MENU> в настоящее время практически не используются, хотя их поддержка ведущими браузерами до сих пор обеспечивается. В спецификации HTML 4.0 оба этих типа списка отмечены как отмененные. Вместо них предлагается использовать маркированные списки, задаваемые тэгом <UL>.

Изначально списки этих типов задумывались как более компактные по сравнению с обычными маркированными списками. Согласно правилам записи элементов этих списков в них не разрешалось использовать блочные элементы, что означает невозможность реализации вложенности списков такого типа. Каждый элемент списка представлял собой одну строчку текста.

Для списков типа <DIR> планировалось ввести ограничение на длину текста элемента списка (24 символа). Такое ограничение позволило бы выводить

списки типа <DIR> в виде, подобном выводу списка каталогов в операционных системах UNIX и MS-DOS при использовании ключа /W (в несколько колонок). Кроме этого, для элементов списков такого типа не отображались маркеры.

В настоящее время все эти замыслы не реализованы, поскольку дальнейшее употребление списков данных типов не рекомендуется. Современные версии браузеров отображают списки этих типов полностью аналогично спискам типа <UL>.

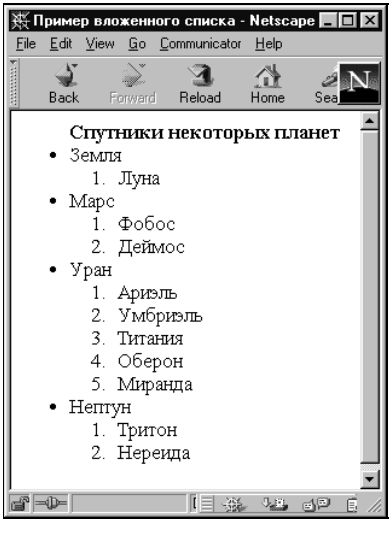
Вложенные списки

Бывают случаи, когда в элемент списка одного типа требуется включить целый список такого же типа или другого. При этом будут организованы многоуровневые или *вложенные* списки. В HTML допустимо произвольное вложение различных типов списков, однако при их организации следует проявлять аккуратность.

Ниже приводится HTML-код документа с вложенными списками, отображение которого показано на рис. 2.6. В этом примере в каждый элемент маркированного списка вложен свой нумерованный список.

```
<HTML>
<HEAD>
<TITLE>Пример вложенного списка</TITLE>
</HEAD>
<BODY>
<UL>
<B>Спутники некоторых планет</B>
<LI>Земля
       <OL>
       <LI>Луна
       </OL>
<LI>Марс
       <OL>
       <LI>Фобос
       <LI>Деймос
       </OL>
<LI>Уран
       <OL>
       <LI>Ариэль
       <LI>Умбриэль
       <LI>Титания
       <LI>Оберон
       <LI>Миранда
       </OL>
```

```
<LI>Нептун
        <OL>
        <LI>Тритон
        <LI>Нереида
        </OL>
</UL>
</BODY>
</HTML>
```



Примечание

Язык HTML не допускает автоматическую многоуровневую нумерацию списков, как это делают мощные текстовые редакторы.

Рис. 2.6. Простейший пример вложенного списка

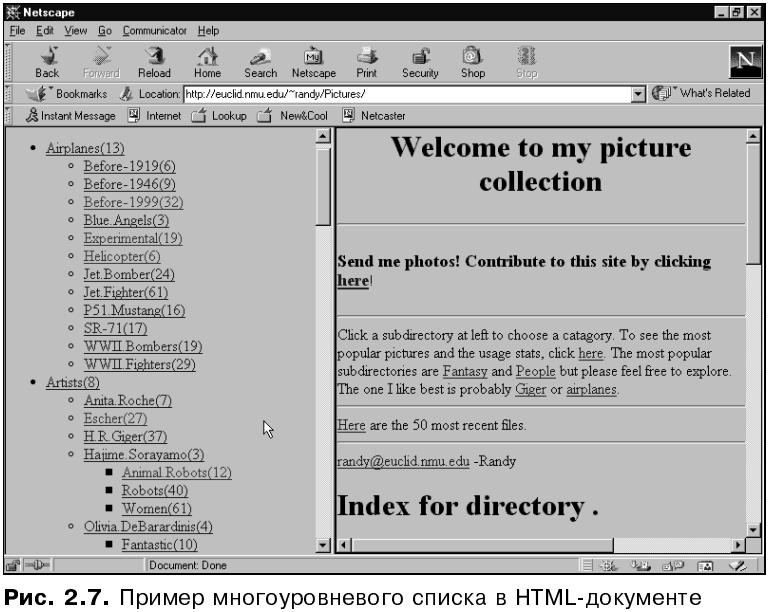


Рис. 2.7. Пример многоуровневого списка в HTML-документе

В заключение приведем пример реально существующей HTML-страницы, на которой имеется маркированный список с тремя уровнями (рис. 2.7). На рисунке видно типовое применение многоуровневого списка для создания структуры типа оглавления разделов документа. Каждый элемент списка всех уровней представляет собой ссылку на соответствующий подраздел.

ГЛАВА 3

Графика

Возможность использования графики трудно переоценить в приложении к любому виду публикации, в том числе и для Web-документов. Без иллюстраций документ однообразен, вял и скучен. Расчетливо подобранная и правильно размещенная в документе графика делает его визуально привлекательнее и, что самое важное, передает одну из основных идей документа.

Изображения могут сделать текст вашего документа более содержательным. Представьте некий сухой HTML-документ, содержащий, например, описание придуманного вами технического устройства. Если он состоит из одного текста, то многим покажется скучным и порой непонятным. А если его "разбавить" несколькими иллюстрациями, размещенными в нужных местах, документ станет более читабельным и визуально привлекательным.

Изображения помогают лучше передать суть и содержание документа. Можете ли вы представить эту книгу без иллюстраций? Вы бы вряд ли ее купили, и авторы не могли бы иметь за это к вам никаких претензий. Вспомните старое правило: лучше один раз увидеть, чем сто раз услышать (в данном случае — прочитать).

Однако во всем нужно чувство меры. Это правило лишний раз подтверждается при просмотре ряда Web-страниц. Довольно часто встречаются Web-документы, загроможденные фоновыми изображениями, ничего не выражающей графикой и раздражающей анимацией. Планируя разместить на своей странице то или иное изображение, убедитесь, что оно действительно необходимо. Если при просмотре печатных материалов вам не составит труда перевернуть страницу, то для Web-документов часто приходится дожидаться окончания его загрузки с тем, чтобы двинуться дальше. Загроможденность графикой также плоха, как и полное ее отсутствие. Дело усугубляет наличие рекламы в Интернет, которая появляется на страницах многих сайтов в виде привлекательных графических плакатов (рекламных баннеров), которые обычно размещаются до основной содержательной части документов. Обычная реакция пользователей на рекламу в Web-документах точно такая же, как и на рекламу, вставляемую посреди вашей любимой телевизионной передачи.

Использование графики в компьютерных документах имеет давние корни. Еще в те прежние времена, когда о мониторах приходилось лишь мечтать, подготовка документов выполнялась на перфолентах, перфокартах и распечатывалась на АЦПУ, предпринимались попытки что-то изобразить графически. С помощью совокупности алфавитно-цифровых символов, накладываемых друг на друга, даже на алфавитно-цифровом печатающем устройстве удавалось построить подобие изображения. Вспомните кинофильм Э. Рязанова "Служебный роман", в котором над рабочим местом одного из героев висела картинка, распечатанная на АЦПУ. Как это ни удивительно, графика такого рода (ее называют ASCII-графикой) до сих пор популярна в среде компьютерщиков. Так, например, в сети FIDOnet многие используют логотипы, составленные из символов кода ASCII. А в сообщениях, которыми обмениваются пользователи электронной почты, принято использовать отдельные совокупности символов для выражения своих эмоций. Поверните голову набок и посмотрите, что вам напоминает следующая совокупность символов :-).

Вернемся в современность. В предыдущих главах вы узнали об основных методах построения HTML-документов. Эта глава дополнит ваш инструментарий средствами включения изображений в страницы, позволяющих сделать их более привлекательными и информативными. Дается обзор различных графических форматов и рекомендации по их применению. Приводятся правила встраивания изображений и примеры их использования.

Общие соображения

Для начала рассмотрим общие вопросы, возникающие на первом этапе работы с графическими изображениями на Web-страницах. Принимая решение о целесообразности включения в документ тех или иных иллюстраций, нужно иметь в виду следующее.

Графические файлы могут иметь значительные размеры, что требует времени для их загрузки. Насыщенность графикой может привести к недопустимо большим затратам времени, требуемым для получения документов, особенно, если используется соединение с помощью модема на небольших скоростях. С другой стороны, одновременная работа нескольких пользователей с большими документами, размещенными на вашем сервере, может также привести к его перегрузке.

Многие пользователи работают в режиме отключения приема графических изображений для увеличения скорости передачи данных. Некоторые пользователи до сих пор используют чисто текстовые программы просмотра. В обоих случаях от полученных документов останется только текстовая часть, которая должна давать информацию о содержательной стороне документа.

Поисковые системы не могут индексировать графику. Поэтому если на ваших страницах расположены только иллюстрации без текстовых пояснений,

то читатели, использующие современные методы поиска, такие страницы не обнаружат.

Следует помнить, что пользователи могут работать с различным разрешением экрана монитора и различной глубиной цвета. Страницы, хорошо смотрящиеся при одном разрешении, могут выглядеть совершенно по-другому при ином разрешении. Сейчас на многих сайтах просто указывают, что его материалы оптимизированы для разрешения 800×600.

Нужно помнить также, что многие изображения защищены законом об авторских правах. Публикация изображений без санкции автора может привести к неприятностям.

В общем, изображения на Web-страницах могут использоваться двумя способами: в качестве фонового изображения, на котором располагаются элементы основного документа, и изображения, встраиваемые в документ. Далее будут рассмотрены особенности применения тех и других изображений. Вопросы использования изображений, отдельные части которых являются ссылками на другие документы (так называемые карты-изображения), вынесены в отдельную главу.

Способы хранения изображений

Рассматривая изображение на экране монитора, вы на самом деле видите большое количество разноцветных точек (пикселов), которые, будучи собранными вместе, образуют некую картинку. Отсюда следует, что графический файл должен содержать информацию о том, как представить этот набор точек на экране. Существует много способов описания графической информации, соответственно имеется значительное количество форматов хранения графических файлов, — порядка нескольких десятков.

Все форматы хранения графической информации можно разделить на два типа: векторный и растровый.

Файлы векторной графики содержат математические данные о том, как перерисовать изображение с помощью отрезков прямых (векторов) при выводе его на экран. Процесс вывода требует дополнительной обработки, но такое представление графической информации имеет важное преимущество: масштаб изображения может быть изменен без потери качества, так как не существует фиксированной связи между тем, как он определен в файле и выводом точек на экран. При масштабировании растровой графики обычно происходит потеря разрешения, что ухудшает качество изображения.

Векторная графика, как правило, употребляется для изображений с четкими геометрическими формами. Примером ее применения являются системы автоматизированного проектирования (CAD). В векторном виде хранится информация для некоторых типов шрифтов.

Растровая графика предполагает хранение данных о каждой точке изображения. Для отображения растровой графики не требуется сложных математических расчетов, достаточно лишь получить данные о каждой точке и отобразить их на экране.

Примечание

В действительности вопрос вывода растровых изображений не так прост. Во-первых, большинство форматов хранения растровых изображений предусматривают сжатие данных, поэтому перед выводом требуется осуществлять процедуру декодирования. Во-вторых, при выводе изображений браузерами допустимо изменять их масштаб. Поэтому при отображении должна решаться довольно нетривиальная задача расчета цвета отдельных точек по исходной совокупности данных. В зависимости от алгоритма, используемого для масштабирования изображения, можно получить различный результат. В-третьих, при отображении иногда требуется решать вопросы несоответствия палитры изображения или глубины цвета режиму работы монитора. Все перечисленные вопросы успешно решаются современными браузерами (причем это выполняется на лету, при загрузке изображений), однако конкретные алгоритмы могут различаться, что приводит к разному представлению одного и того же документа в различных браузерах.

На Web-страницах в подавляющем большинстве случаев используется растровая графика в двух форматах: GIF и JPG. Подробное обсуждение преимуществ и недостатков того и другого формата будет дано ниже. Здесь лишь заметим, что именно эти два формата непосредственно поддерживаются популярными браузерами, а для использования большинства других графических форматов потребуются специальные средства.

Формат BMP является стандартом MS Windows и поддерживается браузером Internet Explorer, однако его употребление не может быть рекомендовано, так как данный формат не поддерживает сжатие данных.

Разработанный недавно формат PNG был призван заменить растровый формат GIF, однако, несмотря на его очевидные преимущества, должного распространения на настоящий момент не получил.

Иные графические форматы (кроме GIF и JPG) в HTML-документах на WWW-серверах практически не встречаются, хотя принципиально это возможно. Использование других форматов не рекомендуется по следующим причинам: во-первых, только для GIF и JPG осуществляется встроенная поддержка в большинстве браузеров, тогда как для иных файлов необходимо подключение внешних программ отображения, во-вторых, структура файлов GIF и JPG наиболее подходит для передачи данных по сети и является независимой от платформы. Так, например, использование старого и доброго формата PCX, который существует уже более 10 лет и распознается практически всеми графическими редакторами, крайне неудачно для сетевых приложений. Во-первых, алгоритм сжатия формата PCX (RLE — метод группового кодирования) дает малую степень сжатия, однако, вследствие своей простоты, позволяет распаковывать данные со скоростью, близкой к скорости

считывания. Для сетевых приложений определяющим фактором является размер файла, от которого непосредственно зависит время передачи данных, по сравнению с которым время распаковки составляет ничтожную величину. Во-вторых, несмотря на последовательную структуру файла PCX (данные в файле всегда хранятся по строчкам, начиная с первой до последней), палитра располагается в конце PCX-файла (это справедливо только для 256 цветов), что не дает возможности начать выдачу изображения по мере считывания файла. Последнего недостатка не имеют 16-цветные PCX-файлы (палитра которых располагается в заголовке), а также последние версии PCX-файлов, которые разрешают хранить изображение в 16,7 млн цветов (для них понятие палитры отсутствует).

Перейдем теперь к общим вопросам включения изображений в HTML-страницы, а лишь затем более подробно обсудим нюансы использования различных форматов.

Фоновые изображения

Разработчики Web-страниц могут управлять цветом фона документа, а также указывать изображения, используемые в качестве фонового. Идея применения фоновых изображений хорошо знакома пользователям системы Windows, в которой предусмотрен ряд возможностей по изменению параметров рабочего стола (desktop). В этой системе в качестве параметров отображения рабочего стола может быть указан как однотонный цвет, так и фоновый узор или рисунок (рис. 3.1).

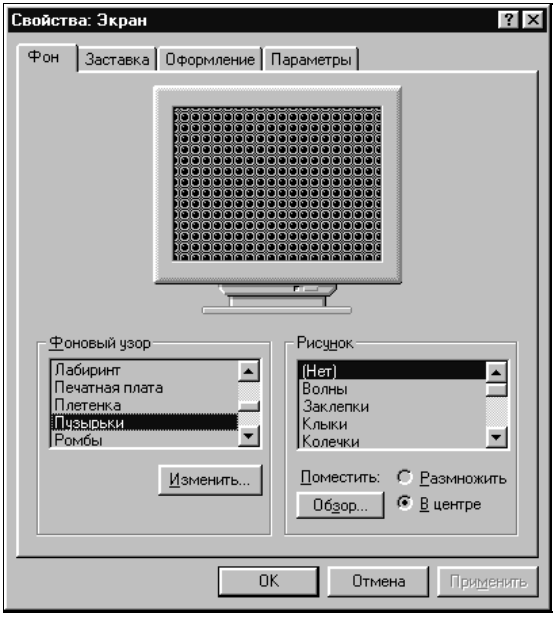


Рис. 3.1. Выбор параметров рабочего стола MS Windows

Во многом аналогично выполняется настройка параметров фона для HTML-документов. Для задания цвета фона употребляется параметр BGCOLOR тэга <BODY>, а фоновое изображение включается в документ при помощи параметра BACKGROUND. В качестве значения параметра BGCOLOR указывается название цвета или его составляющие в шестнадцатеричном коде. В качестве фонового изображения должен использоваться графический файл формата GIF или JPG.

Примечание

Названия стандартных цветов и их шестнадцатеричное представление приведено в приложении П2.

Фоновое изображение для HTML-документа всегда заполняет все окно просмотра (в отличие от рабочего стола Windows, где изображение может не размножаться). Если размер изображения меньше размеров окна просмотра, то оно будет размножено по принципу мозаики. Поэтому фоновые изображения должны создаваться так, чтобы при появлении на экране границы сшивки повторяющихся изображений были бы невидимы. Эта задача напоминает подбор рисунка при оклеивании обоями стен комнаты.

Обычно в качестве фонового берется небольшое изображение, для загрузки которого по сети не требуется значительного времени. Существуют огромные коллекции изображений (текстур), которые можно использовать при разработке своих собственных HTML-документов (рис. 3.2).

Другим часто используемым вариантом является фоновое изображение в виде бледного рельефного логотипа. Такая графика ясно идентифицирует сайт и не мешает восприятию материала.

Приведем пример записи тэга <BODY> с указанием фонового цвета и фонового изображения:

```
<BODY BACKGROUND=texture.gif BGCOLOR=gray>
```

Пример документа с фоновым изображением показан на рис. 3.3. Заметим, что одновременное задание параметров BACKGROUND и BGCOLOR вовсе не обязательно. Любой из них, равно как и оба вместе, могут отсутствовать.

На первый взгляд может показаться, что указание фонового цвета излишне при задании фонового изображения. В действительности все наоборот. Можно рекомендовать всегда указывать цвет фона документа если задается фоновое изображение. Дело в том, что при загрузке документа прежде всего отображается текстовая часть, а на следующем проходе будут загружаться изображения, в том числе и изображение, используемое в качестве фонового. До момента загрузки и отображения фонового изображения цвет фона документа будет определяться значением параметра BGCOLOR или устанавливаться по умолчанию. Опыт работы с HTML-документами, получаемыми по сети, показывает, что до загрузки фонового изображения порой проходит

достаточное количество времени, в течение которого пользователь знакомится с уже загруженным текстом. В какой-то момент проявляется фоновое изображение, изменяя гамму цветов документа. Чтобы предотвратить резкое изменение гаммы цветов, следует задавать значение цвета фона близким к цветам фонового изображения.

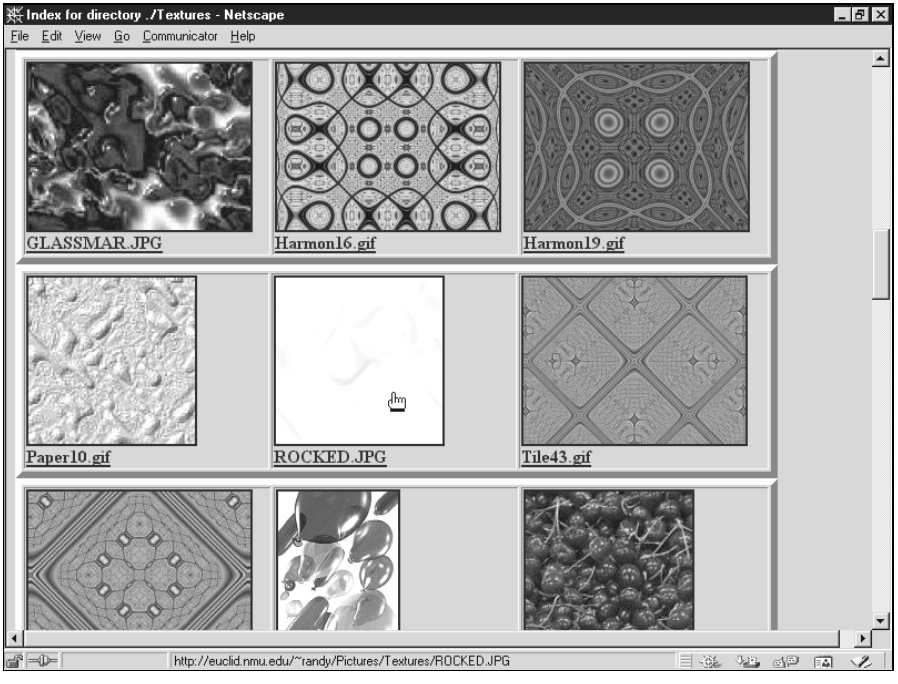


Рис. 3.2. Коллекция текстур

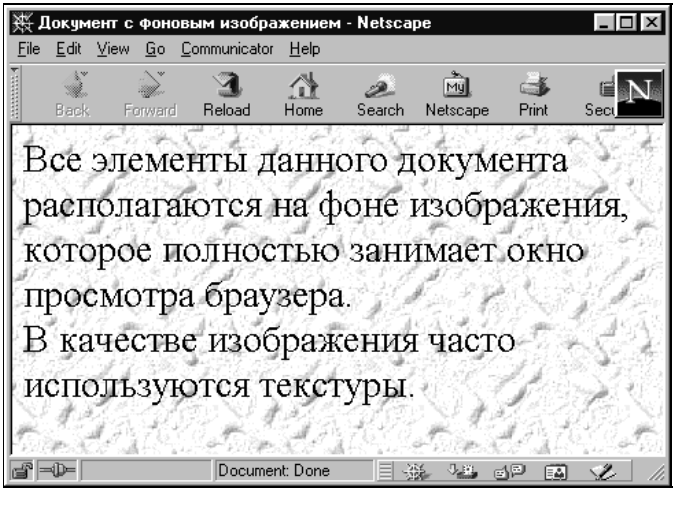


Рис. 3.3. Использование фонового изображения

Примечание

При выборе цвета фона и характера фонового изображения следует не забывать о необходимости контраста между цветом текста и фона. Неудачное соотношение цветов может затруднить чтение текста. Обратите внимание на рис. 3.3. Без фонового изображения прочитать текст было бы гораздо проще.

Есть еще причина, из-за которой задание цвета фона документа следует рекомендовать. Пользователь может отключить загрузку изображений. В этом случае фоновое изображение также не будет загружено.

Еще один вариант предпочтений пользователя исключит и выдачу фонового изображения и указание цвета фона. Конкретный вариант настроек зависит от используемого браузера. Так, например, в браузере Netscape есть режим принудительного задания гаммы цветов, перекрывающих настройки, указанные в документе (пункт **Always use my colors, overriding document** меню **Colors** вкладки **Preferences**). При установке данного флажка цветовые настройки отображаемого документа не будут использоваться, в том числе вообще не будет загружаться фоновое изображение.

Примечание

Фоновые изображения можно использовать не только применительно ко всему документу. Так, многие браузеры разрешают задавать фоновые изображения для отдельных ячеек таблиц.

При помощи таблиц стилей, речь о которых пойдет в последних главах книги, можно задавать фоновые изображения для любых элементов Web-документа. Например, можно тексты всех заголовков документа расположить на своем фоновом изображении.

Встраивание изображений в HTML-документы

Для встраивания изображений в HTML-документ следует использовать тэг `<IMG>`, имеющий единственный обязательный параметр `SRC`, определяющий URL-адрес файла с изображением. Простейший пример встраивания изображения:

```
<IMG SRC=picture.gif>
```

Данный тэг может иметь ряд параметров, обсуждение которых дается ниже.

Выравнивание изображений

При включении графического изображения в документ можно указывать его расположение относительно текста или других элементов страницы. Способ выравнивания изображения задается значением параметра `ALIGN` тэга `<IMG>`. Возможные значения этого параметра приведены в табл. 3.1.

Таблица 3.1. *Значения параметра* `ALIGN`

Значение параметра `ALIGN`	Действие параметра
`TOP`	Верхняя граница изображения выравнивается по самому высокому элементу текущей строки
`TEXTTOP`	Верхняя граница изображения выравнивается по самому высокому текстовому элементу текущей строки
`MIDDLE`	Выравнивание середины изображения по базовой линии текущей строки
`ABSMIDDLE`	Выравнивание середины изображения посередине текущей строки
`BASELINE` или `BOTTOM`	Выравнивание нижней границы изображения по базовой линии текущей строки
`ABSBOTTOM`	Выравнивание нижней границы изображения по нижней границе текущей строки
`LEFT`	Изображение прижимается к левому полю окна. Текст обтекает изображение с правой стороны
`RIGHT`	Изображение прижимается к правому полю окна. Текст обтекает изображение с левой стороны

Поясним действие параметров выравнивания, приведенных в таблице. Сразу же оговоримся, что все значения параметров выравнивания изображений можно условно разделить на две группы по их принципу действия. К одной группе относятся два значения параметра — `LEFT` и `RIGHT`. При использовании любого из этих параметров мы получаем так называемое "плавающее" изображение. В этом случае изображение прижимается к соответствующему краю окна просмотра браузера, а последующий текст (или другие элементы) "обтекают" изображение с противоположной стороны. Здесь текст, размещаемый рядом с изображением, может занимать несколько строчек.

К другой группе значений параметров относятся все остальные. При их использовании изображение как бы встраивается в строчку текста, а параметры выравнивания задают расположение изображения относительно строки текста. Таким образом, в отличие от плавающих изображений, здесь изображение является обычным элементом строки. Это легко понять, если представить, что изображение является просто одной буквой строки текста, правда, достаточно большой (типа буквицы).

Приведем пример HTML-кода, в котором используются изображения, как элемент строки.

```
<HTML>
<TITLE>Выравнивание изображений</TITLE>
<BODY>
```

```
Выравнивание<IMG SRC=eagle.gif ALIGN=top>по верхнему краю
<P>
Выравнивание по<IMG SRC=eagle.gif ALIGN=baseline>базовой линии
</BODY>
</HTML>
```

Отображение браузерами приведенного выше кода показано на рис. 3.4.

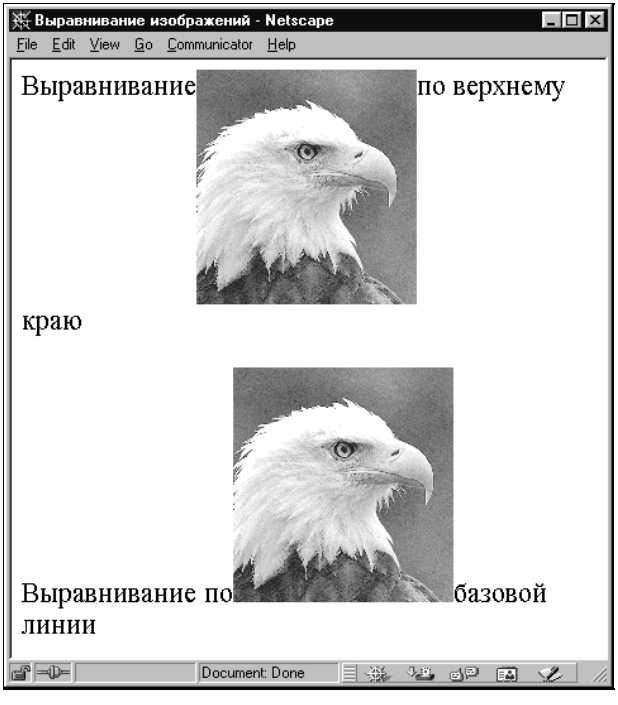


Рис. 3.4. Выравнивание изображений как элементов текстовой строки

Приведем пример плавающего изображения (рис. 3.5). В примере изображение прижато к правому краю окна просмотра браузера, а последующий текст располагается с левой стороны от изображения. Количество строк, располагаемое рядом с изображением, может изменяться в зависимости от размеров шрифта текста, а также размеров окна просмотра. Текст, не поместившийся рядом с изображением, автоматически продолжается ниже. Этот пример построен на следующем исходном коде:

```
<HTML>
<TITLE>Выравнивание изображений</TITLE>
<BODY>
<IMG SRC=spb.gif ALIGN=right>
<P ALIGN=JUSTIFY>
Санкт-Петербург расположен в самой восточной оконечности Финского залива
в устье реки Невы, на 42-х островах ее дельты. Географические координаты
```

```
города: 59"57' северной широты и 30"19' восточной долготы от Гринвича.
Из крупнейших городов мира (с населением свыше одного миллиона человек)
Санкт-Петербург является ближайшим к Северному полюсу, он находится на
одной широте с северной частью Камчатки и южной оконечностью Аляски.
<P ALIGN=JUSTIFY>
Высокоширотным положением города объясняется явление белых ночей. Они
наступают 25-26 мая, когда солнце опускается за горизонт не более чем
на 9', и вечерние сумерки практически сливаются с утренними. Наибольшая
продолжительность дня приходится на 21-22 июня; заканчиваются белые ночи
16-17 июля, продолжаясь в общей сложности более 50 дней.
</BODY>
</HTML>
```

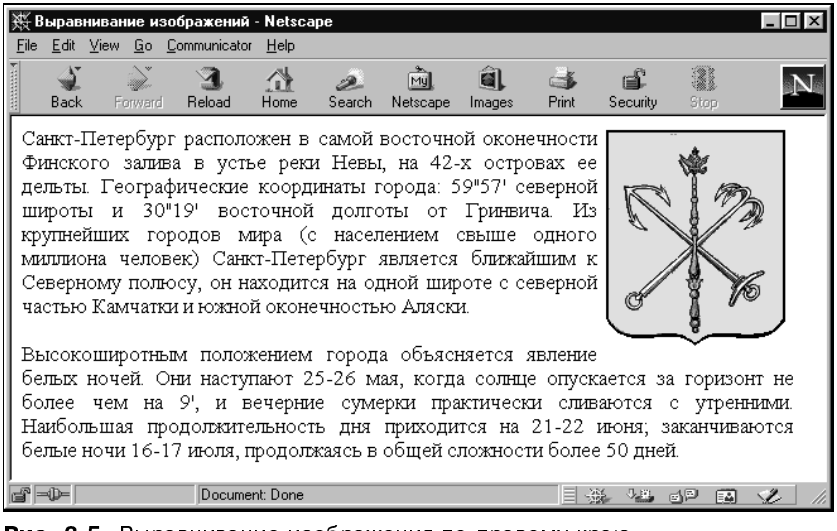


Рис. 3.5. Выравнивание изображения по правому краю

Отметим, что по умолчанию изображения выравниваются по базовой линии.

Примечание

Браузер Netscape реализует выравнивание со значением `ABSBOTTOM` точно так же, как и `BOTTOM`. Аналогично выравнивание со значением `ABSMIDDLE` реализуется так же, как и `MIDDLE`.

Браузер Microsoft Internet Explorer выполняет выравнивание согласно приведенной выше таблице.

Возникает вопрос, в чем разница между базовой линией и нижней границей строки? Или различие между самым высоким элементом строки и самым высоким текстовым элементом строки? Результат действия этих параметров может отличаться в зависимости от содержимого рассматриваемой строки.

Базовая линия (`BASELINE` или `BOTTOM`) — это нижняя часть линии текста, которая проводится без учета нижней части (descender) некоторых символов,

например, букв типа j, q, y. В отличие от выравнивания по базовой линии, при задании значения ABSBOTTOM выравнивание выполняется по нижней части самого низкого элемента в строке, т. е. по одному из символов строки, имеющему элементы, лежащие ниже базовой линии.

Аналогично обстоит дело с различием между параметрами TOP и TEXTTOP. Например, самым высоким элементом в строке может быть графическое изображение, в то время как самым высоким текстовым элементом строки является, как правило, заглавная буква. На рис. 3.6 показаны возможные отличия. Для данного примера был использован следующий HTML-код:

```
<HTML>
<TITLE>Различие параметров выравнивания</TITLE>
<BODY>
<IMG SRC=monkey.gif>
<IMG SRC=mouse.gif ALIGN=top width=160>
Выравнивание ALIGN=TOP

<P>
<IMG SRC=monkey.gif>
<IMG SRC=mouse.gif ALIGN=texttop width=160>
Выравнивание ALIGN=TEXTTOP

</BODY>
</HTML>
```

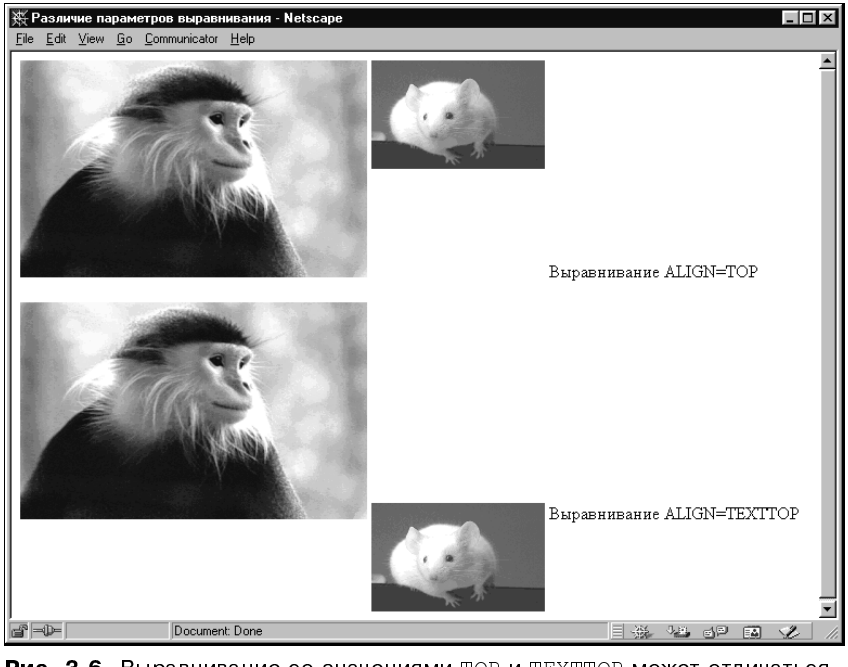


Рис. 3.6. Выравнивание со значениями TOP и TEXTTOP может отличаться

Если в документе используются плавающие изображения, выровненные со значением RIGHT или LEFT, то имеется возможность принудительного прекращения обтекания в заданном месте текста. Это обеспечивается применением тэга принудительного прерывания строки
 с параметром CLEAR. В качестве значений параметра CLEAR можно использовать следующие: LEFT, RIGHT или ALL. Так, для приведенного выше примера в нужном месте текста можно разместить строку:

```
<BR CLEAR=right>.
```

Текст, следующий далее, будет размещаться ниже изображения с новой строки.

Задание размеров выводимого изображения

Тэг встраивания изображений имеет два необязательных параметра, указывающих размеры изображения при отображении — WIDTH и HEIGHT. Значения параметров могут указываться как в пикселах, так и в процентах от размеров окна просмотра.

Значения параметров ширины и высоты изображения могут не совпадать с истинными размерами изображения. В этом случае браузеры автоматически при загрузке изображений выполняют его перемасштабирование. Заметим, что неаккуратное задание параметров может привести к изменению пропорций рисунка и, как следствие, к его искажению.

Любой из этих параметров может быть опущен. Если задан только один из параметров, то при загрузке рисунка второй параметр будет вычисляться автоматически из условий сохранения пропорций. Изменение размеров изображений при помощи задания параметров ширины и высоты может использоваться для просмотра иллюстраций в уменьшенном виде, однако такой подход не сокращает время загрузки изображения. Вопросы использования миниатюрных версий изображений будут рассмотрены ниже.

Если не требуется решать задачу изменения размеров изображения, настоятельно рекомендуется указывать их реальные размеры в пикселах с помощью параметров WIDTH и HEIGHT. Определить действительные размеры используемых вами изображений можно при помощи любой из многих программ работы с растровой графикой. Например, LView Pro или Paint Shop Pro. Указание действительных размеров:

- позволяет читателю, работающему в режиме отключения загрузки изображений, иметь представление о размерах иллюстраций по пустому прямоугольнику, выдаваемому на экран вместо изображения (если размеры не будут указаны, то браузер, не зная их, выведет маленькую пиктограмму и форматирование страницы будет нарушено). Пример будет приведен ниже в разделе, где обсуждается альтернативный текст);

- позволяет ускорить верстку документа на экране. Обычно браузеры должны загрузить все встроенные изображения прежде, чем отформатировать текст на экране. Указание размеров встроенных изображений позволяет выполнить форматирование документа до полной загрузки файлов с изображениями.

Приведем пример HTML-кода, отображение которого показано на рис. 3.7. Действительные размеры отображаемого рисунка — 150×174. Этот рисунок отображен трижды в разном масштабе. Для изменения масштаба использовано указание ширины изображения, а его высота автоматически масштабируется. В итоге рисунки при отображении будут иметь масштаб — 2:1, 1:1 и 1:2.

```
<HTML>
<TITLE>Задание размеров изображений</TITLE>
<BODY>
<IMG SRC=spb.gif WIDTH=300>
<IMG SRC=spb.gif>
<IMG SRC=spb.gif WIDTH=75>
</BODY>
</HTML>
```

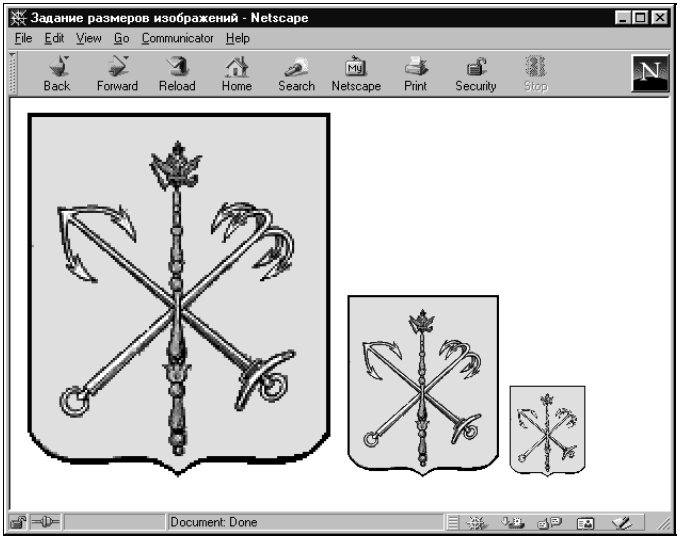


Рис. 3.7. Все три изображения являются отображением одного и того же файла

Отделение изображения от текста

Для тэга `<IMG>` можно задавать параметры `HSPACE` и `VSPACE`, значения которых определяют отступы от изображения, оставляемые пустыми, соответственно, по горизонтали и вертикали. Это гарантирует, что между текстом и изобра-

жением останется пространство, необходимое для нормального восприятия. В приведенном ниже HTML-коде, отображение которого показано на рис. 3.8, со всех четырех сторон изображения задан отступ, равный 20 пикселам. Сравните этот рисунок с рис. 3.5, где отступы от изображения не задавались.

```
<HTML>
<TITLE>Использование параметров HSPACE и VSPACE</TITLE>
<BODY>
<IMG SRC=spb.gif  ALIGN=left  HSPACE=20  VSPACE=20>

(Текст абзаца)

<P>

(Текст абзаца)

</BODY>
</HTML>
```

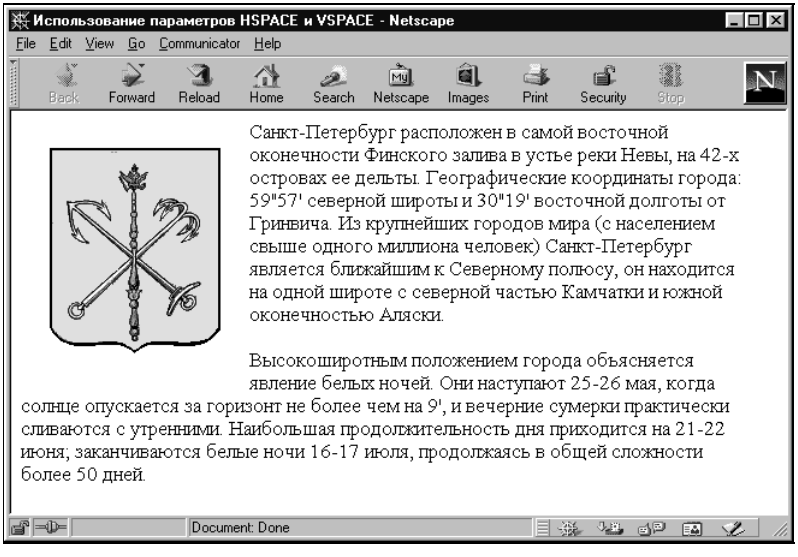


Рис. 3.8. Отступы от изображения улучшают восприятие документа

Рамки вокруг изображений

Изображение, встраиваемое на страницу, можно поместить в рамку различной ширины. Для этого служит параметр `BORDER` тэга `<IMG>`. В качестве значения параметра используется число, означающее толщину рамки в пикселах. По умолчанию рамка вокруг изображений не рисуется. Исключением из этого правила является случай, когда изображение является ссылкой.

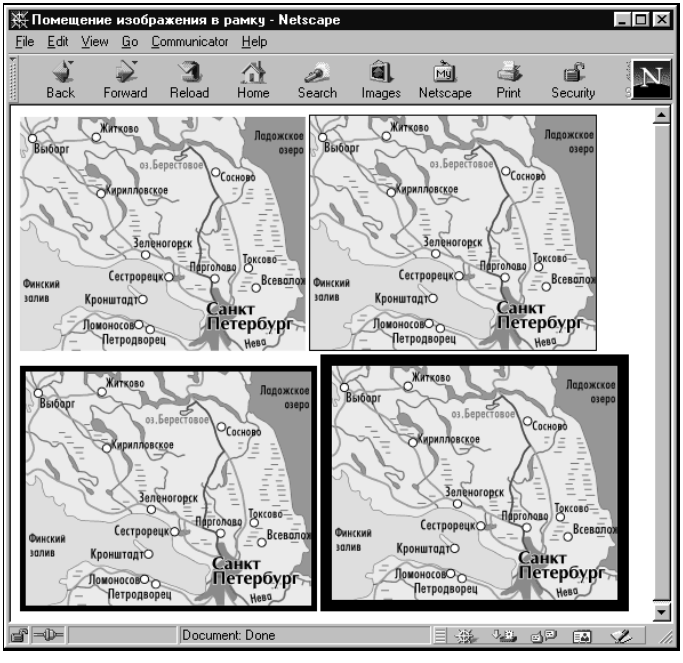


Рис. 3.9. Изображения могут помещаться в рамку различной толщины

На рис. 3.9 показан пример задания рамок различной толщины для одного и того же изображения. HTML-код данного примера приведен ниже:

```
<HTML>
<TITLE>Помещение изображения в рамку</TITLE>
<BODY>
<IMG SRC=map.gif>
<IMG SRC=map.gif BORDER=1>
<IMG SRC=map.gif BORDER=5>
<IMG SRC=map.gif BORDER=10>
</BODY>
</HTML>
```

Примечание

Если изображение является указателем ссылки, то по умолчанию браузеры заключают их в рамку синего цвета. Избежать появления рамки можно, указав значение `BORDER=0`.

Альтернативный текст

Одним из параметров тэга `<IMG>` является параметр `ALT`, определяющий альтернативный текст. Его указание дает возможность пользователям неграфических браузеров или пользователям, работающим в режиме отключения

загрузки изображений, получить некоторую текстовую информацию о встроенных изображениях.

При отключенном изображении вместо них на экране появится альтернативный текст, определенный значением параметра ALT. Значение этого параметра имеет смысл и для случаев, когда загрузка изображений будет выполняться. Поскольку загрузка изображений выполняется на втором проходе после отображения текстовой информации, то изначально на экране на месте изображения появится альтернативный текст, который по мере загрузки будет сменяться изображением.

Современные браузеры будут также отображать альтернативный текст в качестве подсказки (tooltip) при помещении курсора мыши в область изображения. На рис. 3.10 показано отображение документа в режиме отключения загрузки изображений. Заметим, что точно так же документ будет выглядеть и при отсутствии файлов с изображениями. Различие в двух изображениях связано с тем, что для первого из них явно указаны размеры, а для второго размеры не заданы. Для этого примера использовался следующий фрагмент HTML-кода:

```
<IMG SRC=spb.gif  ALT="Герб Санкт-Петербурга" WIDTH=150 HEIGHT=174>
<IMG SRC=moscow.gif  ALT="Герб Москвы">
```

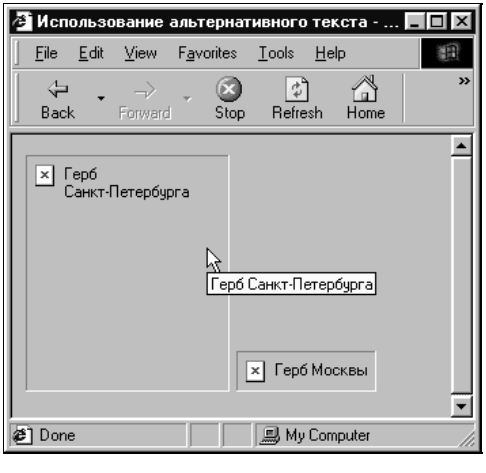


Рис. 3.10. При отключенной загрузке изображений будет показан альтернативный текст

Использование изображения в качестве ссылки

Графические изображения могут использоваться не только в качестве иллюстраций, но и выполнять роль указателей гипертекстовых связей. Для обеспечения работы изображения в качестве ссылки на другие ресурсы достаточно включить изображение внутрь тэга-контейнера <A>. Пример:

```
<A HREF=My_doc.html><IMG SRC=map.gif></A>
```

Любая часть такого изображения будет работать как указатель ссылки на документ My_doc.html. Существуют возможности задания изображений, отдельные фрагменты которого будут указывать на различные ресурсы. Обсуждение этих вопросов дано в главе 6.

Параметр *LOWSRC*

Еще одним параметром, который можно задать в тэге `<IMG>`, является `LOWSRC`. Его значение определяет файл с альтернативным изображением, которое должно появляться при первом проходе выдачи на экран загружаемого документа. Смысл задания двух загружаемых на одно и то же место изображений заключается в следующем. В качестве изображения, указываемого значением параметра `LOWSRC`, рекомендуется выбирать картинку того же содержания, но более низкого разрешения, возможно с меньшей глубиной цвета или даже монохромное. Изображения такого рода занимают значительно меньше места и, как следствие, быстро загружаются. Это позволяет уже на первом проходе формирования документа увидеть общие черты изображений и не ожидать их полной загрузки. Пример записи:

```
<IMG SRC=main.gif  LOWSRC=low.gif>
```

Использованию параметра `LOWSRC` присущ ряд особенностей. Если была загружена картинка, адрес которой был задан параметром `LOWSRC`, то основная картинка будет иметь точно такие же размеры, что и предыдущая. Если изображения исходно имели различные размеры, то основное изображение будет приводиться к размерам первого. При этом могут быть нарушены пропорции основного изображения. Во избежание этого разумно указывать настоящие размеры основного изображения (параметрами `WIDTH` и `HEIGHT` тэга `<IMG>`). Тогда на первом проходе изображение низкого разрешения будет приводиться к размерам настоящего изображения, а на втором проходе — будет заменено им.

Примечание

Параметр `LOWSRC` был предложен компанией Netscape и в настоящее время распознается только этим браузером. Другие браузеры просто проигнорируют наличие незнакомого им параметра.

Использование миниатюрных версий изображений

На Web-страницах часто используются миниатюрные версии изображений (thumbnail) в качестве графических указателей ссылок на полноразмерные изображения (рис. 3.11). Изображения такого рода представляют собой уменьшенные копии оригинальных изображений, имеющие иногда также

меньшую глубину цвета или представленные в оттенках серого цвета. Файлы с такими изображениями занимают значительно меньше места по сравнению с полноразмерными и поэтому гораздо быстрее грузятся. С помощью миниатюрных версий можно быстро просмотреть набор изображений, доступных для загрузки, и выбрать понравившееся.

В Интернете можно найти целый ряд специализированных сайтов, содержащих разного рода изображения и обеспечивающих предпросмотр (preview) с помощью миниатюрных копий. Большое число сайтов не предназначено для детских глаз, но есть и ряд интересных серверов, предлагающих изображения различной тематики. Можно отметить сайт **http://www.snap-shot.com**, одна из страниц которого приведена на рис. 3.11.

Рис. 3.11. Типовое применение миниатюрных версий изображений

Еще одним интересным сервером с огромным набором изображений является сайт, расположенный по адресу **http://euclid.nmu.edu/**, одна из страниц которого была показана на рисунке в предыдущей главе (см. рис. 2.7).

Примечание

Термин thumbnail является общепринятым для обозначения миниатюрных версий изображений. В буквальном переводе с английского языка он означает "ноготь большого пальца" или нечто, имеющее небольшой размер.

Для создания миниатюрных версий изображений можно воспользоваться одним из пакетов для работы с растровыми изображениями. Большинство пакетов такого рода обладают возможностями масштабирования, изменения яркости, контрастности и других параметров изображений. Из простейших пакетов, успешно справляющихся с поставленной задачей, можно назвать Paint Shop Pro и LView Pro. Среди более мощных пакетов можно отметить Adobe Photoshop. Существуют также специализированные программы, ориентированные на решение задачи создания миниатюрных копий изображений.

Примечание

Браузеры предоставляют возможность масштабирования изображений "на лету" при помощи указания требуемых размеров в тэге `<IMG>`. Это свойство может использоваться для выдачи миниатюрных копий на экран, но при этом не решается главная проблема — уменьшение объема передаваемых файлов. Довольно странно загружать полное изображение, чтобы увидеть его маленькую копию. Такой подход рационально использовать только при просмотре локальных файлов, когда время загрузки файлов с жесткого диска или локальной сети невелико.

Формат GIF

Формат файла GIF (Graphics Interchange Format) первоначально был предложен корпорацией Compuserve Incorporated для передачи графических данных по сети. Из-за популярности сети Compuserve формат GIF получил широкое распространение и в настоящее время поддерживается множеством программ работы с графикой, а также реализован на ряде платформ. Популярность формата увеличивается за счет свободного распространения его спецификации и свободного использования. Поскольку изначально формат разрабатывался для передачи данных в потоке, а не как формат для хранения данных в файле, то его последовательная организация как нельзя более подходит для размещения графических изображений на WWW-серверах. К положительным качествам формата можно отнести возможность хранения множественных изображений, внесение перекрывающего текста, отображение ряда изображений с задержкой, задание режимов восстановления предыдущего изображения, введение данных для специфических приложений. К недостаткам следует отнести ограниченное количество цветов (не более 256), реализованных в виде палитры 24-битовых цветов, отсутствие возможностей по хранению градаций серого и данных цветовой коррекции, хранению данных в моделях, отличных от RGB (например, CMYK или HSI). Хотя 256 цветов во многих случаях оказывается достаточно, сохранение фотореалистичных изображений в этом формате может привести к ухудшению цветовой гаммы картинки.

В настоящее время используются две модификации GIF-файлов, которые носят название GIF87a и GIF89a. Последняя модификация является расши-

рением GIF87a. Официальная документация по GIF89a датирована 31 июля 1990 года.

Версия файла определяется дескриптором, который записывается в первые шесть байтов файла. В качестве дескриптора используется строка "GIF87a" или "GIF89a". Дескриптор определяет минимальные требования к программам, работающим с данным файлом. Так, например, для файла, имеющего дескриптор "GIF87a" это означает, что для работы с ним не требуется дополнительных возможностей, введенных в модификации GIF89a. Поэтому даже современные программы, способные сохранять изображения в формате GIF, предлагают оба варианта сохранения. Таким образом, модификация GIF87a не является устаревшей, а представляет собой подмножество GIF89a с полной совместимостью снизу вверх.

Уже в 1987 году в модификации GIF87a были определены следующие возможности:

- наличие нескольких различных изображений, закодированных в одном файле (далее будем называть отдельное изображение, входящее в состав данного файла, *кадром*);
- позиционирование изображения на логической области экрана;
- хранение изображения с чередованием строк (interlacing).

Это означает, что уже более десяти лет назад была потенциальная возможность создавать простейшие анимации, собирая последовательность кадров в одном GIF-файле, однако широкое распространение "анимированных" (animated GIF) файлов произошло только в последние годы, главным образом, за счет широкого использования на Web-страницах.

В GIF-файле определены два различных варианта хранения данных. В одном из них все строки изображения записываются подряд от начальной до конечной (построчное хранение — NonInterlaced). В другом варианте строки сохраняются в определенном порядке (хранение с чередованием строк — Interlaced). Для последнего варианта порядок хранения строго определен, а именно, строки изображения с чередованием размещаются в четыре прохода:

- каждая 8-я строка, начиная с 0-й;
- каждая 8-я строка, начиная с 4-й;
- каждая 4-я строка, начиная с 2-й;
- каждая 2-я строка, начиная с 1-й.

Вариант хранения изображения задается при его создании или сохранении после редактирования и является параметром самого файла изображения. В зависимости от варианта хранения выполняется и вывод изображения на экран — либо картинка разворачивается сверху вниз, либо она постепенно проявляется и улучшается от прохода к проходу. Очевидно, что браузеры

графических файлов могут использовать свои варианты появления изображения на экране, применяя различные эффекты вне зависимости от схемы хранения, однако для этого необходимо первоначальное получение всего файла с последующей его обработкой для получения нужного эффекта. Для WWW-браузеров характерно отображение файлов в процессе их получения, что определяет однозначную связь между схемой хранения и процессом выдачи на экран.

Заметим, что на первом проходе заполняется 1/8 часть всех строк изображения. На каждом последующем проходе число заполненных строк растет вдвое и, таким образом, за четыре прохода изображение принимает оригинальный вид. При чересстрочной схеме хранения уже после первого прохода можно увидеть контуры появляющегося изображения и, при необходимости, остановить дальнейшую загрузку, чтобы не загружать сеть перекачкой бесполезной для вас информацией. При построчной схеме хранения, получив данные для 1/8 части всех строк, пользователь увидит верхнюю 1/8 часть изображения сразу в оригинальном качестве, по которой, скорее всего, не удастся определить содержание рисунка.

Связь между размерами уже полученной части файла с изображением и числом строк, отображенных браузером по этим данным, вообще говоря, не прямая. Дело в том, что формат GIF предусматривает сжатие данных, поэтому размер файла и структура данных будут существенно зависеть от его содержимого.

Покажем пример отображения файла формата GIF, загрузка которого была прервана после первого прохода (рис. 3.12). Размер изображения составляет 570×495 (256 оттенков серого), файл имеет размер 243 Кб. Для получения данных первого прохода потребовалось загрузить 32 Кб, что и составляет примерно 1/8 от размеров всего файла.

Примечание

Конечно, браузеры не обладают возможностью остановки загрузки GIF-файлов после получения данных для первого (или любого другого) прохода. Здесь нами был использован искусственный прием, единственной целью которого являлась иллюстрация данного примера.

Можно было ожидать, что на изображении после первого прохода будет прорисована только каждая восьмая строка, а остальные будут иметь цвет фона. На самом деле браузеры поступают более умно. Очевидно, что для многих изображений близлежащие пикселы имеют близкий цвет, поэтому, выполняя интерполяцию по пикселам с известным значением цвета, можно примерно угадать цвет других пикселов. Браузеры применяют простейший вариант такого подбора, а именно, получив данные для пикселов какой-либо строки, окрашивают в такой же цвет все последующие строчки изображения, для которых данных пока еще нет. Для первого прохода, при получении данных для одной строки цвет ее пикселов распространяется еще на семь смежных строк. Использование такого подхода позволяет довольно

быстро сформировать контуры изображения, что хорошо видно на рис. 3.12. При получении очередных данных пикселы отдельных строчек будут изменять свой цвет, давая эффект проявления изображения. Для сравнения качества приведем оригинальное изображение (рис. 3.13).

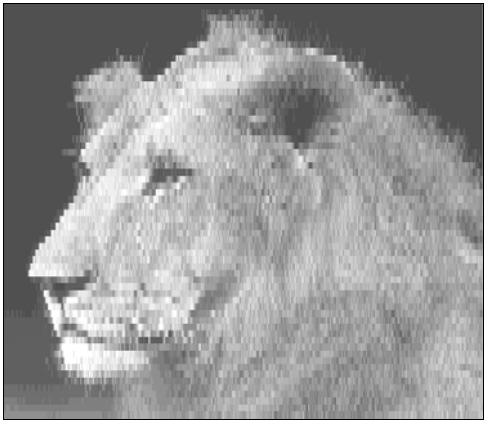


Рис. 3.12. Отображение чересстрочного GIF-файла, полученное после первого прохода

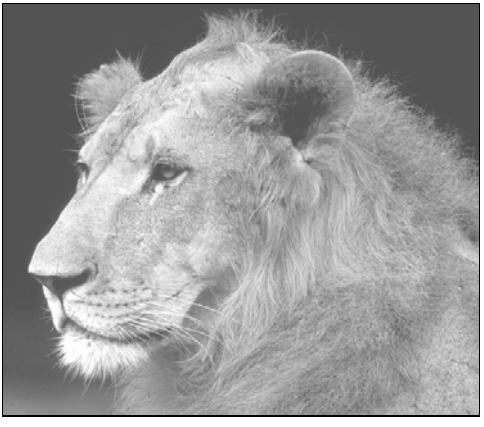


Рис. 3.13. Оригинальное изображение

Примечание

Для неискушенных читателей заметим, что термины Interlaced и NonInterlaced (NI) часто встречаются при описании режимов работы мониторов. Хотя на первый взгляд речь идет о схожих вещах, а именно о порядке заполнения строк экрана, однако применительно к мониторам эти термины отражают технический аспект работы развертки кадра монитора, что никак не влияет на работу программ в общем, и метод выдачи строк изображения формата GIF, в частности.

Модификация GIF89a расширила возможности управления выводом изображений, разрешив определение следующих параметров:

- время (в сотых долях секунды), по истечении которого начнется выдача следующего кадра;
- ожидание ввода пользователя для перехода к следующему кадру;
- определение прозрачного цвета;
- включение комментариев, которые не отражаются при выводе изображений;
- включение строк текста;
- определение режима восстановления области экрана, занятой изображением, после завершения отображения данного кадра;
- задание внутри файла данных, специфичных для отдельного приложения.

Примечание

Перечисленные пункты являются лишь потенциальными возможностями управления выводом изображений, определенные спецификацией GIF89a. В действительности, большинство программ работы с графическими изображениями поддерживают только часть из них.

Рассмотрим кратко перечисленные возможности.

Время задержки определяет промежуток между окончанием выдачи текущего кадра и началом выдачи следующего. Точнее, перед началом выдачи следующего кадра выполняется восстановление области экрана, занятого изображением, в соответствии с заданным режимом.

Режим ожидания ввода пользователем позволяет интерактивно управлять сменой кадров при просмотре. Для отдельного кадра может быть задано и время задержки, и режим ожидания ввода. При этом смена кадра осуществится при наступлении любого из этих событий — окончании времени задержки или ввода пользователя. Браузерами данный режим не поддерживается.

Один из цветов палитры изображения может быть определен как прозрачный (transparent). Это указывает браузеру, что при выводе изображения те пикселы, цвет которых объявлен прозрачным, не требуется изменять на экране. Чаще всего в качестве прозрачного цвета задается цвет фона изображения.

Включение комментариев в GIF-файл используется для текстовых пояснений к изображениям, которые не являются частью самого изображения. Размер комментариев не ограничивается, однако следует иметь в виду, что излишне большое их количество может привести к неоправданному увеличению размеров файла.

Включение строк текста в файл позволяет визуализировать его в простой форме в виде фрагмента графического изображения. Визуализация будет выполняться с использованием сетки ячеек символов, определяемой параметрами в этом блоке файла. Текстовые данные будут представлены как моноширинные символы по одному символу на ячейку наиболее подходящим шрифтом и размером. Данные будут представимы до тех пор, пока не будет достигнут конец данных или сетка ячеек не будет заполнена. Браузеры не поддерживают отображение строк текста.

Режим восстановления области экрана задает способ, которым будет обрабатываться фрагмент графики после его отображения. Определены следующие способы:

- не делать ничего;
- оставить как есть, что в большинстве случаев то же самое, что ничего не делать;

- восстановить предыдущее состояние, т. е. заполнить фрагмент экрана тем изображением, которое было до вывода текущего изображения;
- заполнить цветом подложки (фоновым цветом или изображением).

Браузеры не поддерживают восстановление предыдущего состояния фрагмента экрана. Это наиболее сложный режим, так как требует предварительного запоминания содержимого части экрана.

Задание внутри GIF-файла данных, специфичных для отдельного приложения, позволяет практически неограниченно расширять возможности работы с этими файлами. В одном GIF-файле может быть несколько разных блоков, каждый из которых определяет данные для конкретного приложения и не используется другими. Netscape использует такой блок для задания параметров цикла смены изображений. Внутри блока записывается единственный параметр, представляющий собой число от 0 до 32760, который означает количество полных циклов смены кадров, выполняемых при отображении. Число нуль означает бесконечный цикл. Отсутствие этого блока приводит к одноразовой выдаче всех кадров файла с соответствующими временными задержками. Версия Netscape 2.0 при любом числовом значении давала бесконечный цикл. Начиная с версии 3.0, браузер Netscape выполняет заданное число циклов. Внутри блоков, специфичных для приложения, имеется 8-байтовое поле идентификатора приложения и 3-байтовое поле кода приложения. Для блоков задания цикла в качестве идентификатора используется слово "NETSCAPE", а в качестве кода приложения "2.0". Это можно обнаружить, "заглянув" внутрь любого GIF-файла с циклом.

Примечание

Версия Netscape 2.0 на сегодняшний день представляется архаичной. Однако это вовсе не означает, что следует менять значение поля кода приложения. Это значение указывает на номер версии, начиная с которой обеспечивается распознавание такого блока браузером Netscape. Заметим, что данный блок распознается также и браузером Internet Explorer.

Если Netscape обнаруживает в файле предназначенный для него блок данных, то он выполняет загрузку всего файла в кэш компьютера и начинает выполнение цикла анимации. Отметим некоторые особенности отображения:

- если файл слишком велик для размещения в кэше, то цикл выполняться не будет, чтобы не загружать сеть повторными считываниями;
- в процессе выполнения цикла смены кадров изображения из кэша логотип Netscape в правом верхнем углу браузера не показывает падающих звезд, как это происходит в случае получения данных из сети;
- кнопка **Stop** панели навигации браузера горит во время выполнения замены кадров. Просмотр анимации может быть остановлен нажатием кнопки **Stop** или **Esc**. Остановка может произойти на любом кадре из цикла. Возобновление цикла возможно при просмотре изображения от-

дельно от всего документа (View Image) или при перезагрузке (Reload) документа, но не при обновлении экрана (Refresh).

Формат JPG

Формат файлов графических изображений JPG (JPEG) был разработан Объединенной группой экспертов в области фотографии (Joint Photographic Experts Group) как средство для хранения изображений, имеющих большую глубину цвета (24 бита на пиксел, что обеспечивает 16,7 Мб возможных цветов).

Не останавливаясь на деталях хранения информации в этом формате, отметим, что файлы JPG следует использовать, прежде всего, для хранения фотореалистичных изображений. Такого рода изображения можно получить при использовании сканера, оцифровке отдельных видеокадров или с цифровой фотокамеры. Ограничение в 256 цветов, присущее GIF, может снизить качество изображения, что исключается при использовании JPG. Поскольку JPG основан на сжатии данных с потерями, учитывающими особенности восприятия изображения человеком, то без значительного ухудшения картинки можно обеспечить значительную степень сжатия и, как следствие, небольшой размер файла. Аналогичный файл GIF в большинстве случаев будет иметь больший размер.

В настоящее время файлы формата JPG поддерживаются большинством программ работы с растровой графикой. Отметим лишь важное обстоятельство, проявляющееся при сохранении данных в этом формате. Поскольку формат предусматривает потери при сжатии, то уровень потерь (а соответственно, и однозначно связанную с ним степень сжатия) может быть изменен пользователем в широких пределах. Во многих пакетах по умолчанию установлен некий приемлемый уровень, при котором изображение не очень сильно искажается при сохранении в данном формате. Значение этого уровня обычно задается параметрами настроек определенного пакета. Как правило, это число, изменяемое в пределах от 1 до 99, смысл которого для разных программ различен. Более того, для одних программ увеличение такого параметра повышает уровень потерь, а для других наоборот. Здесь можно рекомендовать пробовать сохранять изображения с различным уровнем потерь и визуально наблюдать различия. Одновременно можно следить за изменением размеров получаемых файлов.

Еще одним важным параметром файлов JPG является схема их хранения. Различают две схемы — обычная и прогрессивная (progressive). Прогрессивная схема хранения такова, что при выводе таких изображений создается впечатление постепенного проявления рисунка на экране со все большим уточнением отдельных деталей. Это напоминает проявление изображения при работе с чересстрочными файлами формата GIF, однако здесь уточнение производится не построчно, а, как правило, по прямоугольным облас-

тям размера 8×8 или более. При сохранении изображения в обычной форме его отображение будет выполняться путем разворачивания изображения сверху вниз. Из сказанного можно сделать вывод, что хранение изображений, предназначенных для загрузки по сети, лучше осуществлять в прогрессивной форме.

Покажем пример изображения, сохраненного в формате JPG с высокой степенью сжатия (рис. 3.14). Видна размытость изображения, что вызвано потерями при сжатии. Такое изображение могло также получиться при отображении файла JPG с приемлемым качеством на определенном шаге проявления изображения (по мере загрузки его из сети). Единственным условием для этого является хранение в прогрессивном формате. Сравните это изображение с оригиналом, приведенным выше (см. рис. 3.13), а также с частично загруженным GIF-файлом (см. рис. 3.12).

Рис. 3.14. Формат JPG позволяет хранить изображения со значительной степенью сжатия

Файлы формата JPG, в отличие от файлов формата GIF, не могут иметь несколько изображений, которые будут при просмотре сменять друг друга. Кроме того, для них нет возможности назначить прозрачный цвет.

Какой формат предпочесть — GIF или JPG

В каких случаях предпочтительнее использование формата GIF, а в каких — JPG?

Формат GIF следует использовать для изображений, создаваемых программным путем или рисуемых вручную с помощью графических редакторов, например, графики, гистограммы, несложные рисунки и т. д. (так называемый line art). Ограничение формата — одновременное использование не более чем 256 цветов, для таких изображений в большинстве случаев не играет роли. Алгоритм сжатия, используемый в GIF-формате (LZW — алго-

ритм, названный по фамилиям Lempel-Ziv-Welch), выполняющий сжатие без потерь, обеспечивает точное восстановление изображения и для несложных рисунков достаточно хорошую степень сжатия.

Некоторые проблемы со свободным использованием этого формата возникли в 1995 году. Алгоритм компрессии LZW имеет патент, который в настоящее время принадлежит компании Unisys Corporation. Хотя GIF-формат является свободно распространяемым и используемым, однако компания Unisys Corporation в 1995 году решила обязать всех коммерческих продавцов программных продуктов, использующих LZW-компрессию, лицензировать ее применение. Чтобы избежать уплаты гонорара вместо GIF-формата было решено создать альтернативный формат PNG (Portable Network Graphic, читается "пинг"), поддерживающий прозрачный цвет и чередование строк, однако не допускающий нескольких изображений в одном файле. Считалось, что формат PNG станет достойной заменой формата GIF, поскольку снимает многие ограничения последнего и является абсолютно открытым. Кроме того, в формате PNG предусмотрена еще более хитрая схема хранения данных об отдельных точках изображения по сравнению с чересстрочной схемой формата GIF. Если для формата GIF при чересстрочной схеме хранения можно было увидеть контуры всего изображения после загрузки примерно 1/8 части данных, то для формата PNG то же самое можно сделать, загрузив 1/64 часть данных. В этом формате предусматривается не только чересстрочное, но и чересписксельное (если можно так выразиться) хранение. Несмотря на значительные преимущества, на текущий момент формат PNG широкого распространения не получил, и по-прежнему во всем мире на WWW-серверах лежат GIF-файлы.

Подводя итоги сказанному, сделаем вывод, что формат GIF лучше всего подходит для следующих типов изображений:

- изображений с ограниченным количеством используемых цветов;
- изображений, имеющих четкие границы и края, что свойственно большинству изображений типа меню, кнопок и графиков;
- изображений, в состав которых входит текст.

Формат JPG больше подходит для хранения следующих изображений:

- фотографий, полученных со сканера или цифровой камеры, а также фотореалистичных изображений, построенных на основе компьютерных расчетов (ray-tracing rendering);
- графики со сложным сочетанием цветов и оттенков;
- любое изображение, которое требует более 256 цветов.

Эти рекомендации не носят абсолютного характера и могут не приниматься во внимание, тем более, что не всегда можно провести строгое разграничение между многоцветными фотореалистичными изображениями и рисован-

ной графикой в стиле "line art". Если уже имеется файл с изображением, хранящийся в одном из двух форматов — GIF или JPG, то никогда не следует преобразовывать один формат в другой. Преобразование GIF в JPG может ухудшить качество изображения за счет алгоритма сжатия с потерями. Например, в областях, заполненных одним цветом, обязательно появится муар. Размер файла при таком преобразовании несложных рисунков может даже увеличиться. Преобразование JPG в GIF ограничит палитру цветов до 256 и в подавляющем большинстве случаев приведет к увеличению размера файла.

Некоторые проблемы использования цвета

Рассмотрим некоторые проблемы, связанные с выбором цвета. Сразу оговоримся, что речь будет идти об изображениях с палитрой 256 цветов, и их просмотре в режиме монитора также с 256 цветами. Большая глубина цвета — 15, 16 или 24 бит (т. е. 32 Кб, 64 Кб или 16,7 Мб одновременно отображаемых различных цветов) пока доступна не всем. Кроме того, выбор видеорежима влияет на характеристики частоты развертки монитора. Современные мониторы (вкупе с соответствующим видеоадаптером) позволяют обеспечить вертикальную частоту развертки (частоту смены кадров) до 90—120 Гц. Повышение разрешения (числа пикселов по высоте и ширине экрана) и глубины цвета снижает максимально достижимую частоту смены кадров до значений 72, 60 и даже 56 Гц, что существенно влияет на утомляемость пользователя. Поэтому опытные операторы, проводящие по несколько часов ежедневно у компьютера, без необходимости не пользуются режимами с повышенными характеристиками.

Рассмотрим следующий пример. Создадим в любом графическом редакторе самый простой рисунок, а именно прямоугольник произвольного размера, все пикселы которого закрашены одним цветом. Сохраним изображение в формате GIF с 256 цветами. Просматривая такой файл с помощью большинства графических программ, можно действительно увидеть однотонный прямоугольник. Иначе может получиться, если такой файл использовать в HTML-документе, который будет просматриваться в Netscape Navigator или Microsoft Internet Explorer. Эти браузеры работают с использованием своей собственной палитры, которая может не совпадать с палитрой изображения. Преобразование изображения к палитре браузера выполняется при его загрузке, при этом используется метод Dither, подбирающий недостающий цвет сочетанием нескольких пикселов доступного цвета. В итоге вместо однотонного прямоугольника при внимательном рассмотрении можно увидеть повторяющиеся группы разноцветных пикселов, которые в целом неплохо имитируют требуемый цвет. Отметим, что при просмотре такого файла отдельно в Netscape Navigator (браузеры позволяют загружать не только HTML-документы со ссылками на графические файлы, но и непосредст-

венно просматривать файлы типа GIF) приведение палитры не выполняется, а используется палитра изображения. Эффект приведения палитры может несколько исказить восприятие рисунка, поэтому для изображений, ориентированных на использование в WWW, рекомендуется заранее выполнить приведение к нужной палитре. Для описываемого примера вместо произвольного цвета следует выбрать какой-либо цвет, имеющийся в палитре браузера, и тогда при просмотре всегда будет виден строго однотонный прямоугольник. Изменение палитры используемых цветов доступно во многих программах, предназначенных для редактирования растровых изображений. Приведение палитры к требованиям WWW-браузеров выполняется рядом специальных пакетов, в частности GIF Construction Set и VideoCraft GIF Animator, описываемых ниже. В первом из них предлагается выполнить приведение к палитре Netscape Navigator, используя замену цветов на ближайшие (Remap) или метод Dither. Эти возможности заложены непосредственно в пакет.

Термин dither (или dithering) в среде профессионалов обычно используется без перевода, поскольку подходящего русского термина, который бы кратко и точно отражал суть дела, пока не найдено. В англо-русском словаре терминов по компьютерной графике Д. В. Волкова, А. Н. Ефлеева, Н. Г. Шагуриной (Мир ПК, 1994, № 4, с. 43—52) предлагается переводить словом "клиширование". Часто используются термины "диффузия цвета" или "смешение цветов". Процитируем также пояснение термина dithering, взятое из того же источника: "Способ получения дополнительных градаций цветов в изображении за счет использования цветовых шаблонов, образованных различными сочетаниями цветов пикселов базовой палитры. Метод обеспечивает расширение цветовой палитры при ухудшении разрешающей способности".

Палитра цветов, используемая в Netscape Navigator, крайне проста — все трехмерное RGB-пространство возможных значений цвета равномерно разделено на равные части по каждой координате, и полученные значения подряд записаны в палитру. Для каждой составляющей R, G и B выбраны 6 значений — 0, 51, 102, 153, 204, 255, все возможные сочетания которых дают $6 \times 6 \times 6 = 216$ различных цветов. Дополнительно к 216 цветам используются стандартные цвета Windows. Пакет GIF Construction Set при необходимости выполнит приведение всех возможных цветов к описанным 216.

Создание анимации на основе GIF-файлов

Как уже было сказано выше, структура файлов формата GIF позволяет хранить несколько изображений (кадров) в одном файле и указывать параметры для их смены при отображении. Этим обстоятельством стали широко пользоваться разработчики Web-страниц для создания наборов сменяющихся (анимированных) изображений. Рассмотрим порядок построения анимированных GIF-файлов.

Процесс создания анимации на базе GIF-файлов состоит из трех этапов:

- подготовка отдельных кадров;
- сбор отдельных кадров в единый файл;
- задание параметров цикла выдачи, временных задержек между кадрами и другой информации.

Подготовка отдельных кадров выполняется с помощью любого графического редактора, сохраняющего данные в растровом формате в отдельных GIF-файлах. Простейшим средством является утилита Paint (в Windows 3.1 была утилита Paintbrush), входящий в состав Windows. Подробное обсуждение методов работы с изображениями выходит за рамки данной книги, поскольку здесь в минимальной степени присутствуют особенности сетевых приложений.

При создании отдельных кадров анимации следует помнить, что на сегодняшний день далеко не все браузеры могут выполнять смену кадров GIF-файла. Ряд браузеров, не имеющих таких возможностей, покажут лишь первый кадр. Кроме того, при просмотре в браузерах цикл анимации может быть остановлен пользователем в любой момент, при этом на экране может остаться любой из кадров. В некоторых случаях цикл анимации будет выполнен только один раз и закончится показом последнего кадра. Поэтому при разработке последовательности кадров изображения следует уделить внимание всем кадрам, которые не должны оказаться недоработанными.

Большинство пакетов для работы с графическими изображениями, таких как CorelDRAW! версий со 2 по 5, PhotoPaint, Adobe Photoshop, Paint Shop Pro и другие, поддерживают формат GIF, однако не могут работать с несколькими изображениями, расположенными в одном файле. Чтение файла формата GIF с несколькими изображениями в любом из перечисленных пакетов позволит редактировать лишь первое изображение, причем дальнейшее сохранение файла приведет к потере всех остальных изображений и управляющих блоков. Пользователь может сразу не заметить этой потери, так как никаких предупреждающих сообщений не появляется. Следует принять за правило иметь копии отдельных изображений в отдельных файлах и редактировать только их при помощи любого пакета.

Для сбора в один файл формата GIF с добавлением управляющей информации можно воспользоваться специальными пакетами. Для Windows-платформы одним из наиболее популярных является пакет GIF Construction Set фирмы Alchemy Mindworks Inc. (Beeton, Ontario, Canada), порядок работы с которым будет описан ниже. Информацию о пакете можно получить по адресу:

http://www.mindworkshop.com/alchemy/gifcon.html

Среди других пакетов можно назвать VideoCraft GIF Animator, доступный по адресу:

http://www.andatech.com/vidcraft/demo.html

Пакет сочетает в себе средства для работы с множественными изображениями в GIF-файле и типичные средства для обработки отдельных изображений (настройка яркости, контрастности, повороты изображения, изменение размеров и т. п.). Имеются возможности работы с файлами AVI, выполнения морфинга (Morph), стилизации изображений.

Обзор вопросов построения GIF-файлов можно найти по адресу:

http://members.aol.com/royalef/gifanim/htm

Программа GIF Construction Set

Пакет GIF Construction Set для Windows является мощным средством для работы с GIF-файлами, содержащими несколько блоков. Пакет позволяет:

- создавать файлы, содержащие несколько изображений, из существующих отдельных изображений;
- добавлять, редактировать и удалять блоки комментариев, блоки с текстом, накладывающимся на изображение;
- сохранять изображения в режиме чередования строк или построчном режиме;
- задавать признак прозрачности цвета для отдельных изображений;
- задавать параметры временных задержек, а также цикла анимации для Netscape, которые также распознаются Microsoft Internet Explorer;
- просматривать отдельные изображения;
- импортировать файлы, хранящие изображения во многих популярных форматах, в том числе и 24-битных;
- изменять параметры месторасположения отдельных изображений на логическом экране.

Этот программный продукт существует уже очень давно и, как многие программы, многократно видоизменялся. На настоящий момент последней является версия 2.0a, датированная августом 1999 года. Эта версия сохранила все функциональные возможности предыдущих версий, однако имеет несколько отличный интерфейс. Для большинства задач вполне достаточно наличия и более ранних версий. Так, примеры будут даваться для версий от 1.0K до 1.0P, применение которых вполне допустимо и сегодня, несмотря на то, что появление этих версий относится к 1996 году.

Отличительной особенностью программы является простота работы с ней. Освоение типовых действий для создания анимации требует весьма небольших затрат времени.

Рассмотрим основные возможности программы и методы работы с ней. После запуска программы GIF Construction Set и считывания GIF-файла на экран выдается список блоков файла (рис. 3.15).

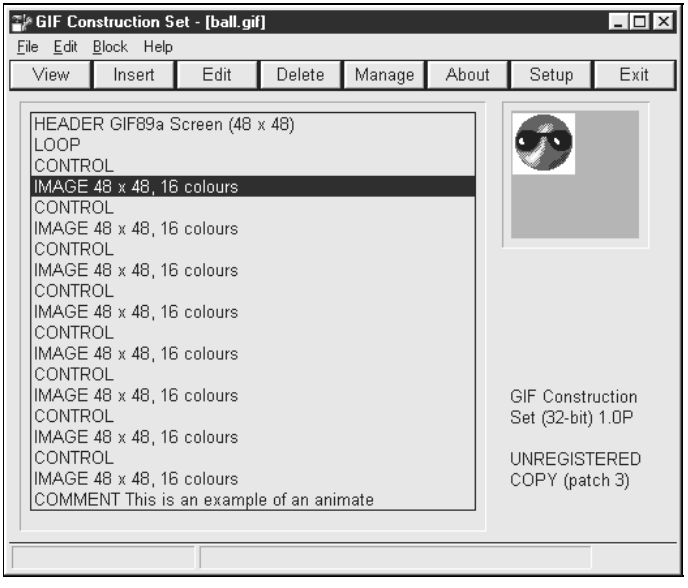


Рис. 3.15. Список блоков GIF-файла

Файл всегда содержит один блок заголовка (HEADER) и один или несколько блоков с отдельными изображениями IMAGE. Остальные блоки могут отсутствовать. В правом верхнем углу окна программы отводится место для быстрого просмотра изображений в уменьшенном виде. Предоставляется возможность удалить, добавить или отредактировать параметры любого из блоков.

Блок HEADER

Блок HEADER (рис. 3.16) содержит информацию о размерах логического экрана в пикселах. Эти параметры не влияют на размеры самих изображений, но для Web-браузеров определяют размеры прямоугольной области, внутри которой будут располагаться все изображения данного GIF-файла. Каждое изображение имеет определенный размер по горизонтали и вертикали и заданное смещение в пикселах от левого верхнего угла логического экрана. В простейшем случае смещение равно нулю, и тогда размер логического экрана должен быть равен максимальному из размеров всех изображений отдельно по горизонтали и по вертикали. При смещении, большем нуля, к размеру изображения добавляется величина смещения. Пользователь может самостоятельно задать размеры логического экрана, однако следует иметь в виду, что отображение картинок, выходящих за пределы логического экрана в Netscape Navigator, вызывает неустранимую ошибку GPF (General Protection Fault). Поэтому программа GIF Construction Set при сохранении файла автоматически вычисляет необходимые размеры логического экрана с учетом размеров отдельных изображений и их смещений независимо от значений, заданных пользователем.

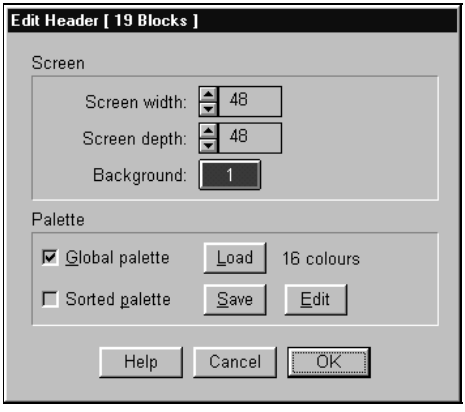


Рис. 3.16. Задание общих параметров GIF-файла

Блок LOOP

Блок LOOP (рис. 3.17) определяет параметры цикла анимации для Netscape. Блок содержит единственный параметр — число повторений цикла, которое можно изменять в пределах от нуля до 32760. Нулевое значение определяет бесконечный цикл. Выдаваемое сообщение о том, что Netscape игнорирует значение счетчика цикла, уже неактуально для Netscape 3.0 и выше. Блок LOOP может отсутствовать, при этом все изображения будут выданы один раз, и на экране останется последнее. При наличии блока LOOP он всегда располагается вторым в списке блоков после HEADER.

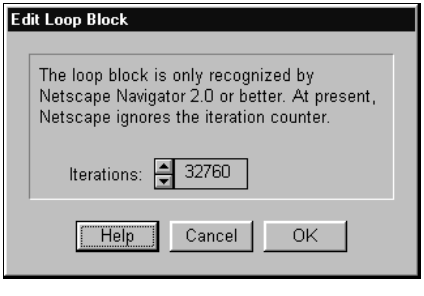


Рис. 3.17. Задание параметров цикла анимации

Блок CONTROL

Перед каждым блоком с описанием изображения IMAGE может располагаться блок CONTROL с управляющими параметрами (рис. 3.18) для последующего блока IMAGE. Задается флаг наличия прозрачного цвета (Transparent colour) и номер этого цвета, который вводится вручную или указывается непосредственно на рисунке при помощи специального маркера, пиктограмма которого представлена в правом верхнем углу. При указании маркером прозрачным становится цвет того пиксела, который был отмечен на

изображении. Так как обычно в качестве прозрачного выбирается цвет фона рисунка, занимающего значительную площадь изображения, то проблема попадания в пиксел нужного цвета не возникает.

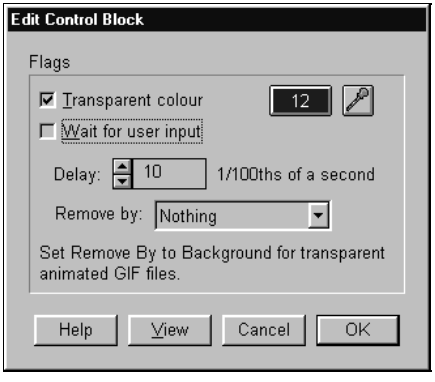


Рис. 3.18. Задание параметров блока управления

Следующий параметр блока (Wait for user input) задает ожидание нажатия клавиши пользователем перед сменой изображения. Браузеры не поддерживают это свойство, и поэтому значение данного параметра не играет роли.

Параметр **Delay** задает время задержки данного изображения в сотых долях секунды. Заметим, что реальное время задержки при отображении в браузере может быть больше указанного, например, при нескольких файлах GIF со сменяющимися изображениями на экране или при недостаточной скорости процессора и/или возможностей видеокарты. При разработке HTML-документов следует учитывать, что выполнения операций по смене изображений могут потребоваться значительные затраты мощностей процессора, что замедлит выполнение других операций и создаст неудобства в работе. Рекомендуется вставлять задержки в несколько секунд в конец цикла анимации для освобождения процессора для выполнения других операций.

Приведем результаты эксперимента по анализу загрузки процессора при выполнении операций такого рода. Был подготовлен GIF-файл, содержащий 5 изображений размером 150×174. Для смены изображений установлен бесконечный цикл. Исследовалась загрузка процессора при отображении этого файла браузером в отсутствии других задач. Результаты исследования определяются большим числом факторов, поэтому приведем все необходимые сведения об условиях проведения эксперимента. Отображение выполнялось браузером Netscape 4.7 в режиме 800×600×256. Процессор Pentium-200 MMX, RAM 64 Мб, видеокарта S3 Virge с 2 Мб памяти. Операционная система Windows 95. Проанализировано два варианта отображения GIF-файла, которые отличаются величинами временных задержек между сменой изображений. Загрузка процессора изучалась с помощью программы "Системный монитор", входящей в состав операционной системы. Результаты анализа показаны на рис. 3.19. Представлена загрузка процессора в процентах при

временных отсчетах, равных 5 с. Левая часть графика (примерно одна четверть всего временного интервала) характеризует загрузку процессора при отображении файла с изображениями, сменяющимися без задержек. Видно, что загрузка составляет величину порядка 40%. Скорее всего, при наличии более быстродействующей видеокарты загрузка процессора была бы еще больше. Далее график показывает загрузку процессора при отображении файла с изображениями, при смене которых задана задержка, равная 0,1 с, а в конце цикла смены изображений установлена задержка 2 с. Загрузка процессора составила примерно 10%. В самом конце графика загрузка упала практически до нуля (осталось 2%), что произошло при нажатии кнопки **Stop** в браузере и прекращении цикла анимации.

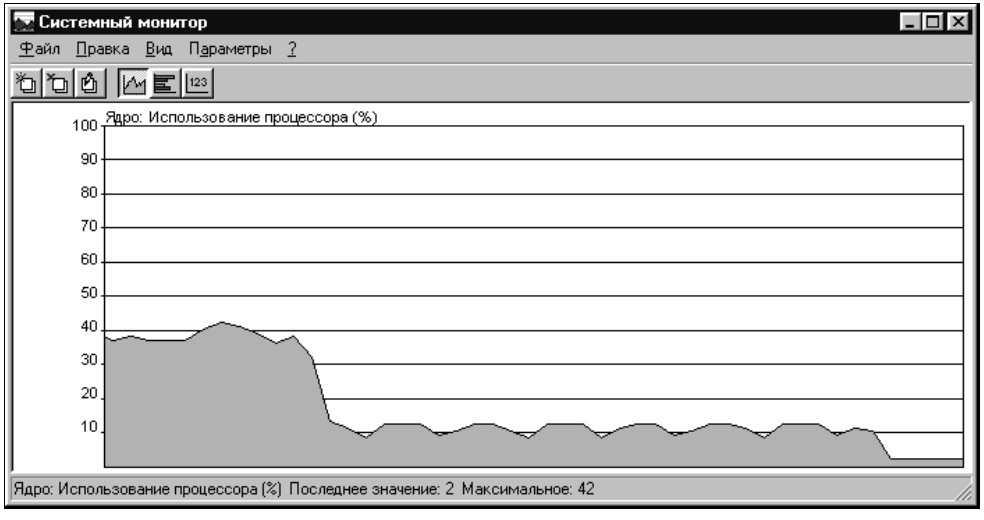


Рис. 3.19. Загрузка процессора при отображении циклически сменяющихся изображений с различными задержками

Из этого примера можно сделать вывод, что неразумное задание параметров анимации может привести к значительной трате ресурсов процессора, часто бесполезной. Несколько анимированных GIF-файлов на странице с минимальными задержками, и неудобства при просмотре обеспечены. К сожалению, в браузерах отсутствует возможность настройки режима демонстрации анимированных изображений. Можно либо не загружать изображения вообще, либо вручную останавливать загрузку кнопкой **Stop**, что приведет, в том числе, и к остановке циклов анимации.

Примечание

Для браузера Netscape известны хакерские приемы, останавливающие анимацию после прокрутки одного цикла. Для некоторых конкретных версий браузера известно, в каких адресах EXE-файла браузера нужно внести изменения, в ре-

зультате которых браузер будет выполнять цикл анимации только один раз независимо от настроек в файле с изображениями. Конечно, подобные приемы нельзя пропагандировать, но раз это делается, значит, актуальность налицо.

Параметр **Remove by** (рис. 3.20) определяет способ восстановления данных на месте изображения после завершения времени задержки изображения. Напомним, что режим Previous image (восстановление предыдущего состояния фрагмента экрана) не реализуется браузерами. Типичным вариантом восстановления является режим Background, обеспечивающий заполнение цветом фона или фоновым изображением, которое в сочетании с прозрачным цветом и изменяющимся смещением внутри логического экрана для отдельных кадров может создать хорошую иллюзию перемещающегося изображения. Если все изображения GIF-файла одинакового размера, имеют одинаковое смещение и не имеют прозрачного цвета, то в качестве режима восстановления достаточно указать Nothing (ничего не делать) или Leave as is (оставить как есть).

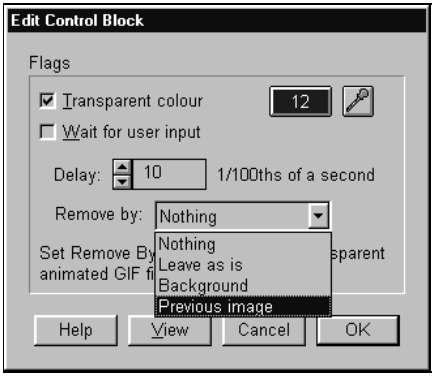


Рис. 3.20. Задание способа восстановления изображения

Примечание

Хотя для параметра **Remove by** можно задавать четыре различных значения, по существу предоставляется всего две возможности: режим Background или любой другой режим из трех оставшихся, которые реализуются браузерами одинаково.

Блок IMAGE

Блок IMAGE непосредственно содержит информацию об отдельном изображении (рис. 3.21). Размеры изображения (Image width и Image depth) приводятся в качестве справочных, их изменение здесь невозможно, а параметры Image left и Image top определяют смещение в пикселах данного изображения относительно левого верхнего угла логического экрана, которое задается пользователем. В данном блоке также задается тип хранения изображения — с чередованием строк (Interlaced) или без, наличие локальной

палитры (Local Palette) и заголовок блока (Block title), который не отражается при выводе и является краткой текстовой характеристикой изображения. Использование локальной палитры для отдельного изображения не рекомендуется. Каждая локальная палитра увеличивает размер файла на 779 байт и может ухудшить отображение цвета при смене кадров. При сохранении файла пакет GIF Construction Set предупреждает о наличии локальной палитры в одном из блоков и предлагает выполнить ее приведение к глобальной.

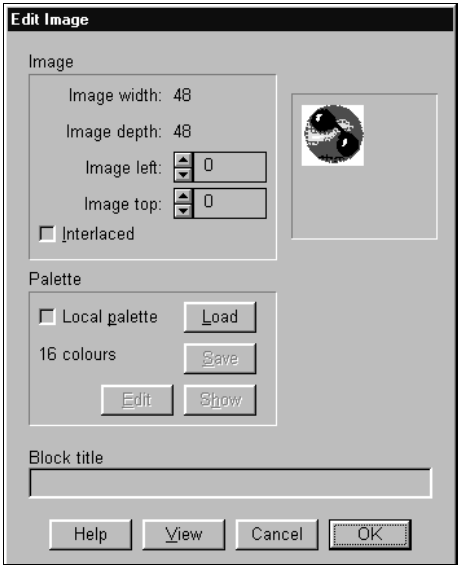


Рис. 3.21. Окно редактирования параметров изображения

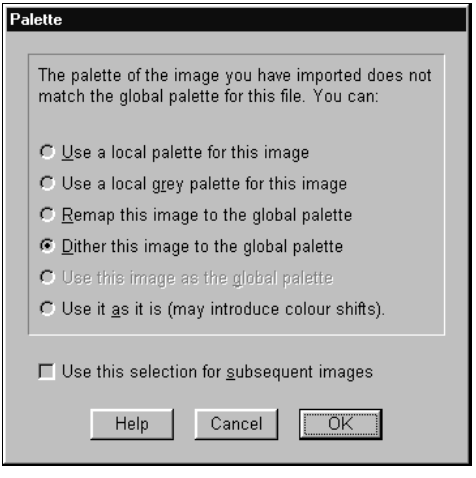


Рис. 3.22. Окно выбора способа приведения палитры

При добавлении (Insert) изображения в список блоков файла выполняется сравнение глобальной палитры с палитрой добавляемого изображения. Предварительно для 24-битовых изображений выполняется преобразование к 256 цветам. Если какие-либо цвета отсутствуют в глобальной палитре, то предлагается сделать выбор варианта преобразования (рис. 3.22). Не углубляясь в детали, отметим, что для фотореалистичных изображений наилучшим выбором является метод Dither, при котором делается попытка имитировать цвет, отсутствующий в палитре совокупностью нескольких разноцветных пикселов из палитры. Для иных изображений обычно достаточно метода Remap, при котором отсутствующие цвета просто заменяются на ближайшие из палитры.

Использование мастера анимации

Создание анимации из отдельных кадров может быть осуществлено вручную использованием пунктов меню пакета, описанных выше. Однако можно

быстро создать нужную анимацию с использованием мастера анимации Animation Wizard, входящего в состав программы GIF Construction Set. Вся процедура создания заключается в ответе на ряд вопросов, последовательно задаваемых пользователю. Отдельные кадры этой процедуры показаны на рис. 3.23 и рис. 3.24.

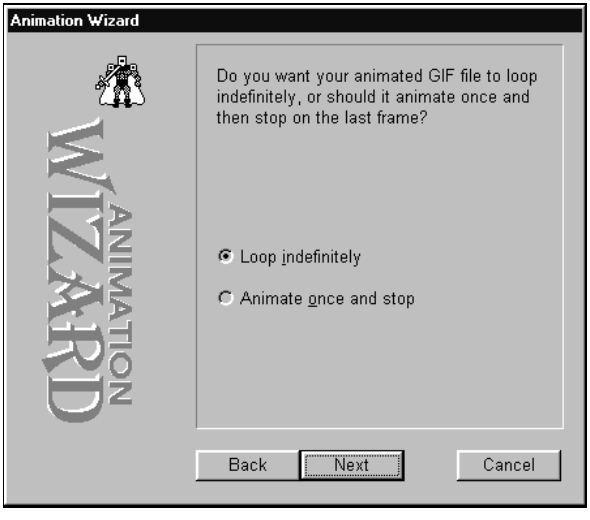


Рис. 3.23. Подготовка анимации с помощью мастера Animation Wizard

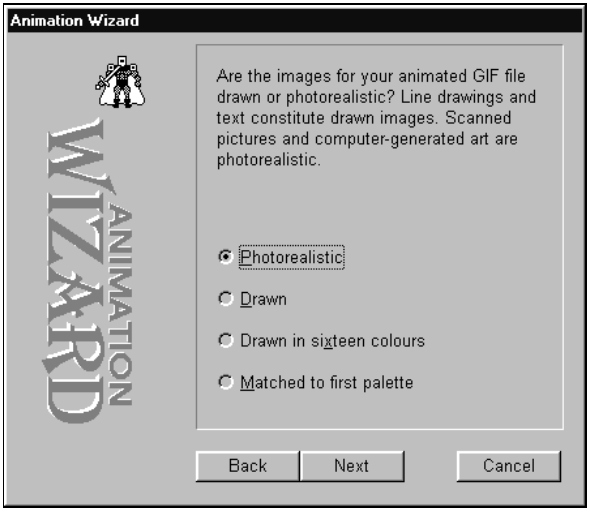


Рис. 3.24. Один из шагов подготовки анимации

Ускорить процесс задания параметров кадров анимации можно с использованием пункта меню **Manage** (рис. 3.25), который позволяет определить ряд общих значений параметров для нескольких кадров за одну операцию.

Еще одна возможность, предоставляемая пакетом, заключается в преобразовании анимации в формате AVI в GIF-файл с набором кадров. Преобразо-

вание может занять значительное время и создать файл очень большого размера. Для AVI-файлов типичны размеры в единицы и десятки мегабайт, однако GIF-файлы такого размера, как правило, не используются.

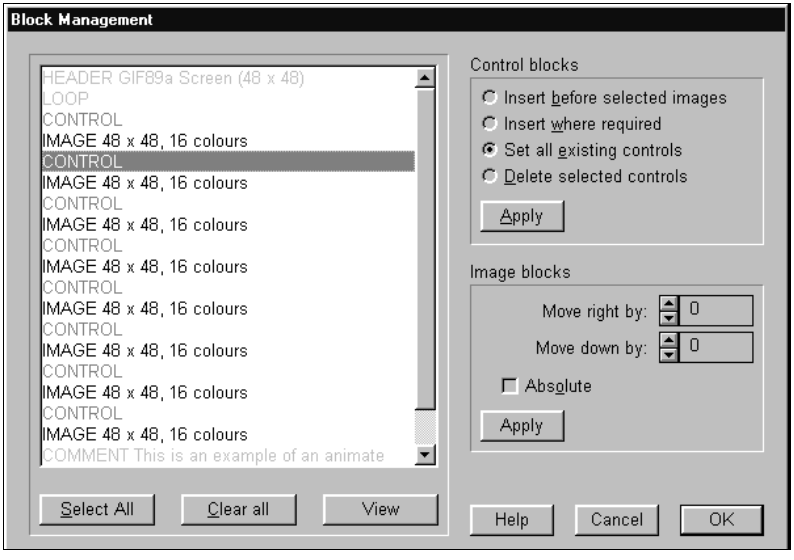


Рис. 3.25. Установка общих свойств блоков

В список блоков, составляющих файл GIF, можно добавлять и ряд других (рис. 3.26), но создание и редактирование произвольных блоков, например, специфичных для какого-либо приложения, не предусмотрено.

Рис. 3.26. Окно вставки блоков

Заметим, что сохранение файла программой GIF Construction Set всегда производится с дескриптором GIF89a.

Дополнительные возможности пакета GIF Construction Set

Современные версии пакета GIF Construction Set обладают некоторыми дополнительными возможностями, которые позволяют создавать GIF-файлы специального содержания. Эти возможности можно найти в пункте меню

Edit пакета. Заметим, что реализация описываемых ниже возможностей связана не с какими-либо конкретными особенностями формата файла GIF или браузеров, которые не были использованы до этого, а с некоторыми сервисными инструментами быстрого создания файлов с множественными изображениями. Перечислим эти возможности в порядке их появления в меню. Отметим, что в таком же порядке они появлялись с развитием пакета от версии 1.0K, в которой не было ни одной из них, и до версии 1.0P, начиная с которой в программу были включены все описываемые ниже пункты.

Banner. Создание бегущей строки. В действительности пакет генерирует ряд изображений, содержащий текст или его фрагменты, которые по мере смены отдельных изображений создают иллюзию бегущей строки.

Transition. Имитация одного из выбираемых методов появления изображения на экране. Различные методы появления изображений широко используются в ряде пакетов работы с графикой. Одним из примеров является пакет презентационной графики PowerPoint, входящий в состав Microsoft Office. В отличие от таких пакетов Web-браузеры не имеют средств для реализации методов появления изображения (не считая вариантов формата GIF — построчное хранение или чередование строк и вариантов формата JPG — обычный или прогрессивный). Использование данного пункта меню позволяет быстро создать ряд изображений, содержащих отдельные фазы смены изображения с заданными временными задержками и числом кадров.

Wide Palette GIF. Использование палитры, содержащей более чем 256 одновременно используемых цветов. Название этого пункта может привести в недоумение, поэтому сразу же еще раз подчеркнем, что структура файла GIF не позволяет хранить изображение, имеющее более 256 цветов. Однако внутри одного GIF-файла может размещаться несколько изображений, каждое из которых имеет свою локальную палитру. Это свойство применяется для искусственного увеличения числа используемых цветов. Все изображение разбивается на ряд отдельных кадров одинакового размера в зависимости от количества используемых цветов. Например, для 510 цветов достаточно двух кадров. Каждый из кадров имеет свою палитру и включает пикселы, цвет которых соответствует локальной палитре данного кадра. Остальные пикселы задаются одним определенным цветом, который объявляется прозрачным. Таким образом, каждый кадр может содержать до 255 уникальных цветов, один цвет отводится под прозрачный. При последовательном выводе нескольких кадров на одно и то же место изменяются лишь те пикселы, цвет которых отличен от прозрачного. Вывод всей последовательности кадров обеспечит требуемое разнообразие цветов. Такие изображения не могут иметь одного прозрачного цвета, поскольку прозрачность используется для специальных целей, как описано выше. Изображения с расширенной палитрой в принципе могут быть анимированными, однако это приведет к медленной смене кадров, так как каждый из них, по существу, состоит из

нескольких. Исходное изображение должно содержать 24-битовый цвет, например, это может быть файл формата BMP, PCX или TIFF. Отметим, что данный режим поддерживается только 32-битной версией пакета. Эффект большого количества цветов хорошо смотрится только в режиме монитора HiColor или TrueColor, при просмотре в режиме 256 цветов изображение может оказаться просто ужасным. Все эти особенности заставляют подходить к использованию режима расширенной палитры с большой осторожностью.

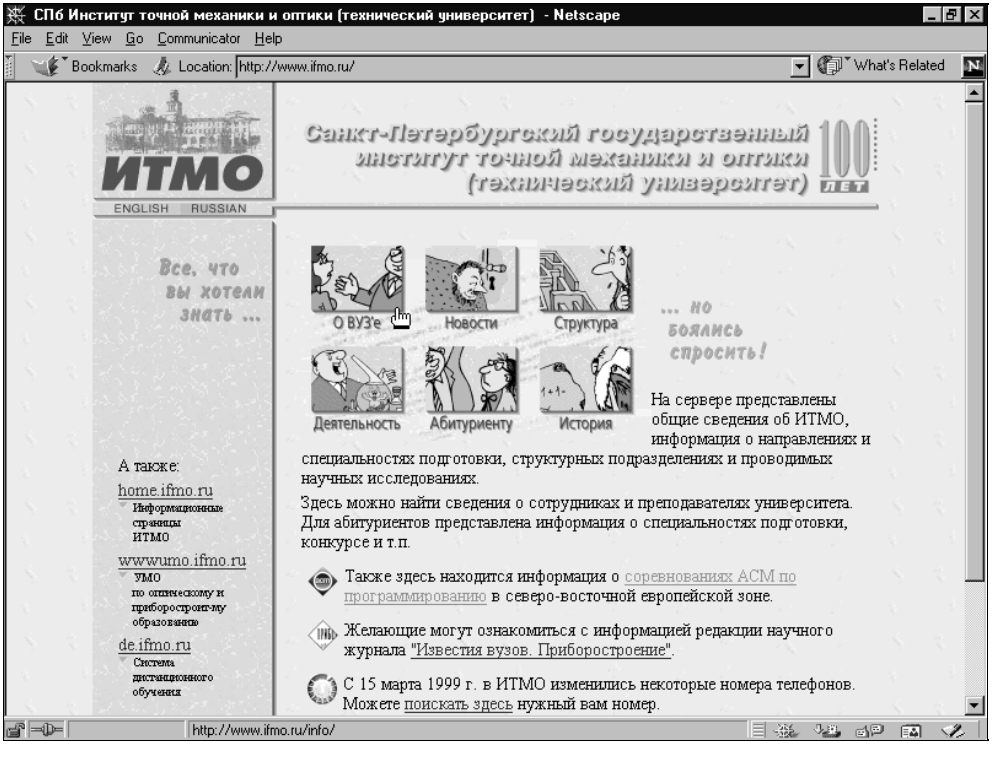


Рис. 3.27. Пример удачно оформленной страницы

LED Sign. Создание бегущей строки, напоминающей по характеру световую рекламу или табло электронных приборов. В тексте строки при помощи специальных знаков задается цвет символов. Длина строки с учетом спецзнаков не может превосходить 260 символов.

В целом все описанные возможности приводят к созданию файлов с множественными изображениями, что может значительно увеличить их размер. Получаемый при этом эффект следует сопоставлять с дополнительными затратами на его осуществление, состоящими в увеличении времени передачи

файлов по сети и общего трафика. Отдельные Web-страницы с излишне большим количеством эффектов часто свидетельствуют только об отсутствии чувства меры у создателей и лишь затрудняют восприятие.

В заключение разговора о применении графики на Web-страницах приведем пример реально существующей страницы (рис. 3.27), на которой умело подобраны и размещены графические изображения, часть которых играет роль заставки, а некоторые изображения являются ссылками на другие ресурсы.

ГЛАВА 4

Таблицы в HTML

Одним из наиболее мощных и широко применяемых в HTML средств являются *таблицы*. Понятие табличного представления данных не нуждается в дополнительном пояснении. В HTML таблицы используются не только традиционно, как метод представления данных, но и как средство форматирования Web-страниц. Приведем примеры реально существующих документов, в которых табличное представление является удобным способом построения документа. На рис. 4.1 показан типичный пример использования таблиц для представления числовых данных, разбитых по строкам и столбцам. На рис. 4.2 использование таблицы служит лишь целям форматирования документа, задания взаимного расположения элементов страницы. При просмотре такого документа сразу не видно, что для его построения используется таблица, так как рамки вокруг ее ячеек не прорисовываются.

Первая версия языка HTML не предусматривала специальных средств для отображения таблиц, так как была в основном предназначена для написания простого текста. С развитием сфер применения HTML-документов стала актуальной задача представления данных, для которых типично наличие ряда строк и столбцов. Создание документов, содержащих выровненные по колонкам данные, на первых порах осуществлялось использованием преформатированного текста, внутри которого необходимое выравнивание обеспечивалось введением нужного количества пробелов. Напомним, что текст внутри пары тэгов `<PRE>` и `</PRE>` выводится моноширинным шрифтом, и все пробелы и символы табуляции являются значащими. Работа по выравниванию такого текста выполнялась вручную, что существенно замедляло создание документов. Поддержка табличного представления данных стала стандартом де-факто, поскольку изначально была реализована во всех ведущих браузерах и лишь по прошествии значительного времени была закреплена в спецификации HTML 3.2.

Специальные средства для создания таблиц, впрочем, не отменяют возможности использования преформатированного текста. Использование таблиц не ограничивается только данными, состоящими из рядов и колонок. Одним из применений является организация расположения разнообразных данных

на странице, которые могут состоять из простого текста, изображений, других таблиц и т. д. Правилам создания таблиц и примерам их использования посвящена данная глава.

Курсы обмена наличной валюты в коммерческих банках Санкт-Петербурга - Netscape

File Edit View Go Communicator Help

Bookmarks Location: http://www.metrocom.ru/Finances/CurrencyMarket/nal_.htm

Курсы обмена наличной валюты в коммерческих банках Санкт-Петербурга

Дата: 27.09.1999 Время: 16:10

Валюта	Макс. покупка	Мин. продажа	Сред. покупка	Сред. продажа
USD	25.10	25.40	24.87	25.55
DEM	13.00	13.59	12.32	13.76
FIM	4.35	4.47	4.08	4.52

Наименование банка	Время	USD Пок./Прод.	DEM Пок./Прод.	FIM Пок./Прод.	GBP Пок./Прод.	SEK/10 Пок./Прод.	CHF Пок./Прод.	По
Автобанк	10:21	25.00/25.65	12.20/14.00	4.05/4.60	-/-	-/-	-/-	-/-
Альфа-банк	11:37	24.90/25.50	12.60/13.60	4.30/4.50	-/-	-/-	-/-	-/-
Балтийский Банк	10:22	24.90/25.96	12.80/14.14	4.00/4.65	-/-	-/-	-/-	-/-
Балтонэксимбанк	10:25	24.95/25.50	11.58/13.61	3.80/4.47	-/-	-/-	-/-	-/-
Башкредитбанк	10:35	25.00/25.60	12.70/14.00	4.20/4.60	-/-	-/-	-/-	-/-
Викинг	09:58	24.60/25.49	11.64/13.69	3.82/4.49	35.53/41.80	-/-	-/-	-/-
Внешторгбанк России	10:30	25.00/25.50	13.00/14.00	4.32/4.62	40.60/42.60	-/-	-/-	-/-
Газпромбанк	10:37	24.80/25.50	12.80/13.60	4.20/4.50	-/-	-/-	-/-	3.6
ГанзаКомБанк	10:27	25.00/25.60	12.40/14.40	4.31/4.55	37.00/42.00	28.00/32.00	-/-	3.8
Петербургский	10:30	24.80/25.50	12.40/13.80	4.10/4.48	-/-	-/-	-/-	-/-

Document: Done

Рис 4.1. Типичный пример HTML-таблицы

Гербы Санкт-Петербурга и городов Ленинградской области - Netscape

File Edit View Go Communicator Help

Bookmarks Location: http://heraldry.hobby.ru/blazon/spb.html

Гербы Санкт-Петербурга и городов Ленинградской области

Герб Ленинградской области утвержден 9 декабря 1997 года. Описание герба: "Герб Ленинградской области представляет собой изображение на геральдическом щите с отношением ширины к высоте 8:9 серебряного якоря, пересеченного золотым ключом на лазоревом (голубом) поле. В верхней части щита изображена серебряная зубчатая стена в перевязь на красном поле.

Document: Done

Рис 4.2. Пример таблицы без рамок

Создание простейших HTML-таблиц

Рассмотрим сначала минимальный набор тэгов и их параметров, необходимый и достаточный для создания несложных таблиц, а затем перейдем к их детальному описанию.

Описание таблиц должно располагаться внутри раздела документа <BODY>. Документ может содержать произвольное число таблиц, причем допускается вложенность таблиц друг в друга. Каждая таблица должна начинаться тэгом <TABLE> и завершаться тэгом </TABLE>. Внутри этой пары тэгов располагается описание содержимого таблицы. Любая таблица состоит из одной или нескольких строк, в каждой из которых задаются данные для отдельных ячеек.

Каждая строка начинается тэгом <TR> (Table Row) и завершается тэгом </TR>. Отдельная ячейка в строке обрамляется парой тэгов <TD> и </TD> (Table Data) или <TH> и </TH> (Table Header). Тэг <TH> используется обычно для ячеек-заголовков таблицы, а <TD> — для ячеек-данных. Различие в использовании заключается лишь в типе шрифта, используемого по умолчанию для отображения содержимого ячеек, а также расположению данных внутри ячейки. Содержимое ячеек типа <TH> отображается полужирным (Bold) шрифтом и располагается по центру (ALIGN=CENTER, VALIGN=MIDDLE). Ячейки, определенные тэгом <TD> по умолчанию отображают данные, выровненные влево (ALIGN=LEFT) и посередине (VALIGN=MIDDLE) в вертикальном направлении.

Тэги <TD> и <TH> не могут появляться вне описания строки таблицы <TR>. Завершающие коды </TR>, </TD> и </TH> могут быть опущены. В этом случае концом описания строки или ячейки является начало следующей строки или ячейки, или конец таблицы. Завершающий тэг таблицы </TABLE> не может быть опущен.

Количество строк в таблице определяется числом открывающих тэгов <TR>, а количество столбцов — максимальным количеством <TD> или <TH> среди всех строк. Часть ячеек могут не содержать никаких данных, такие ячейки описываются парой следующих подряд тэгов — <TD>, </TD>. Если одна или несколько ячеек, располагающихся в конце какой-либо строки, не содержат данных, то их описание может быть опущено, а браузер автоматически добавит требуемое количество пустых ячеек. Отсюда следует, что построение таблиц, в которых в разных строчках располагается различное количество столбцов одного и того же размера, не разрешается.

Таблица может иметь заголовок, который заключается в пару тэгов <CAPTION> и </CAPTION>. Описание заголовка таблицы должно располагаться внутри тэгов <TABLE> и </TABLE> в любом месте, однако вне области описания любого из тэгов <TD>, <TH> или <TR>. Согласно спецификации языка HTML расположение описания заголовка регламентировано более строго:

оно должно располагаться сразу же после тэга `<TABLE>` и до первого `<TR>`. Мы рекомендуем придерживаться этого правила.

По умолчанию текст заголовка таблицы располагается над ней (`ALIGN=TOP`) и центрируется в горизонтальном направлении.

Перечисленные тэги могут иметь параметры, число и значения которых различны. Однако в простейшем случае тэги используются без параметров, которые принимают значения по умолчанию.

Этих сведений вполне достаточно для построения элементарных таблиц. Приведем пример простейшей таблицы, состоящей из двух строк и двух столбцов, отображение которой показано на рис. 4.3.

```
<HTML>
<HEAD>
<TITLE>Пример простейшей таблицы</TITLE>
</HEAD>
<BODY>
<TABLE BORDER>
<TR>
<TD>Ячейка 1 строки 1</TD>
<TD>Ячейка 2 строки 1</TD>
</TR>
<TR>
<TD>Ячейка 1 строки 2</TD>
<TD>Ячейка 2 строки 2</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

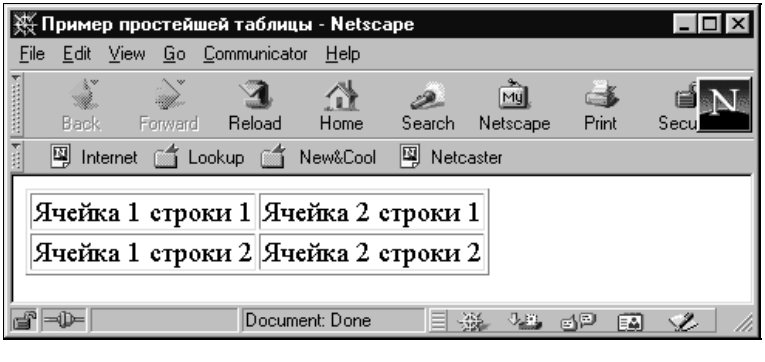


Рис. 4.3. Пример простейшей таблицы

Представление таблиц на странице

Рассмотрим назначение различных параметров, которые могут использоваться в тэгах, описывающих таблицы.

Заголовок таблицы *<CAPTION>*

Тэг заголовка таблицы `<CAPTION>` имеет единственный допустимый параметр `ALIGN`, принимающий значения `TOP` (заголовок над таблицей) или `BOTTOM` (заголовок под таблицей). Параметр `ALIGN` может быть опущен, что соответствует значению `ALIGN=TOP`. В горизонтальном направлении заголовок таблицы всегда располагается по ее центру. Таблица может не иметь заголовка. В качестве заголовка таблицы в большинстве случаев используется простой текст, что регламентируется спецификацией HTML, однако в действительности между тэгами `<CAPTION>` и `</CAPTION>` допустимо записывать любые HTML-элементы, употребляемые в разделе `<BODY>`. Приведем пример записи заголовка таблицы:

```
<CAPTION ALIGN=BOTTOM>Заголовок, располагаемый внизу таблицы</CAPTION>
```

Если данное описание заголовка добавить к приведенному выше примеру, то таблица будет отображаться так, как показано на рис. 4.4.

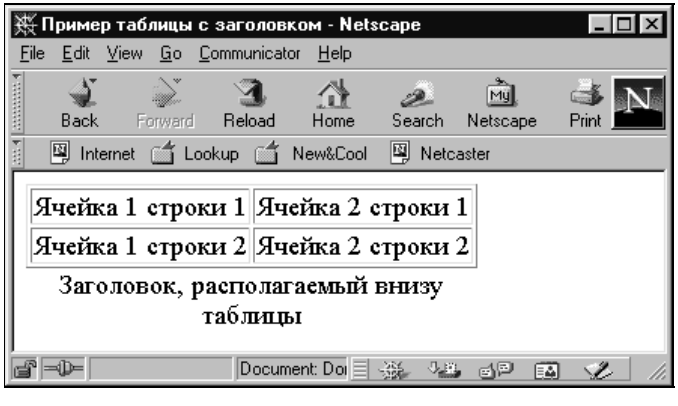


Рис. 4.4. Таблица с заголовком

Браузер Microsoft Internet Explorer предоставляет дополнительные возможности для выбора расположения заголовка. Параметр `ALIGN` допускает значения `LEFT`, `CENTER` и `RIGHT` для горизонтального выравнивания наряду с описанными выше значениями. Отметим, что это один из редких случаев, когда широко распространенный параметр `ALIGN` может использоваться и для горизонтального выравнивания, и для вертикального. Например, запись `ALIGN=RIGHT` обеспечит расположение заголовка, прижатого к правой стороне и размещенного над таблицей. Если записать `ALIGN=BOTTOM`, то так же, как и

в приведенном выше примере, заголовок будет расположен под таблицей. Однако двойное использование в одном заголовке параметра ALIGN недопустимо. Поэтому дополнительно введен специальный параметр для вертикального выравнивания — VALIGN, принимающий значения TOP или BOTTOM. Например, для заголовка, располагаемого внизу таблицы с выравниванием влево, описание имеет следующий вид:

```
<CAPTION ALIGN=LEFT VALIGN=BOTTOM>Заголовок, располагаемый внизу с выравниванием влево</CAPTION>
```

Таблица с данным описанием заголовка в Microsoft Internet Explorer будет отображена следующим образом (рис. 4.5). Если данный пример просматривать в Netscape, то заголовок будет размещен по умолчанию, т. е. над таблицей и посередине в горизонтальном направлении.

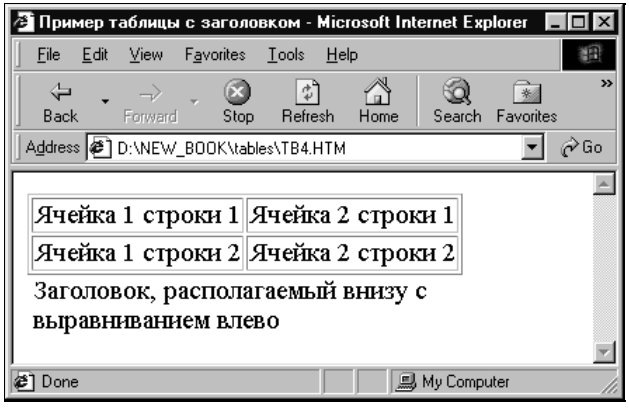


Рис. 4.5. Горизонтальное выравнивание заголовка таблицы браузером Microsoft Internet Explorer

Возможности горизонтального выравнивания заголовка таблицы являются расширением спецификации HTML, не поддерживаются браузером Netscape Navigator, и поэтому ими следует пользоваться только в крайней необходимости.

Параметры тэга *<TABLE>*

Основным тэгом, применяемым при создании таблиц, является тэг <TABLE>. Он может использоваться с рядом параметров, каждый из которых допустимо опускать. Набор допустимых параметров зависит от браузера. Согласно спецификации HTML в тэге <TABLE> могут использоваться следующие параметры: BORDER, CELLSPACING, CELLPADDING, WIDTH, ALIGN. Браузеры Netscape и Microsoft Internet Explorer разрешают кроме перечисленных пяти параметров использовать параметры HEIGHT и BGCOLOR. Отдельные браузеры позволяют также задавать и другие параметры. Рассмотрим назначение общеупотребительных параметров тэга <TABLE>.

Параметр *BORDER*

Параметр BORDER управляет изображением рамки вокруг каждой ячейки, которые, по сути, дают линии сетки таблицы, и вокруг всей таблицы. По умолчанию рамки не рисуются, и на экране пользователь увидит лишь ровно расположенный текст ячеек таблицы. Существует немало ситуаций, когда использование таблиц без рамок вполне оправданно, например, для многоколонных списков, реализованных при помощи таблиц, или задания точного взаимного расположения рисунков и текста. Однако в большинстве случаев для традиционного использования таблиц ее ячейки полезно отделить друг от друга линиями сетки, что облегчает восприятие и понимание информации, содержащейся в таблице.

Для добавления в таблицу рамок необходимо включить в код <TABLE> параметр BORDER, который может иметь численное значение.

Например, <TABLE BORDER> или <TABLE BORDER=10>.

Численное значение параметра определяет толщину рамки в пикселах, рисуемую вокруг всей таблицы, однако на толщину рамок вокруг каждой ячейки это значение не влияет. При отсутствии численного значения обычно оно принимается равным минимальному значению (1), хотя для различных браузеров стиль показа рамок может отличаться. Возможность независимого управления отображением рамки вокруг всей таблицы и рамками вокруг ячеек отсутствует.

Пример таблицы с рамкой толщиной 10 пикселов приведен на рис. 4.6.

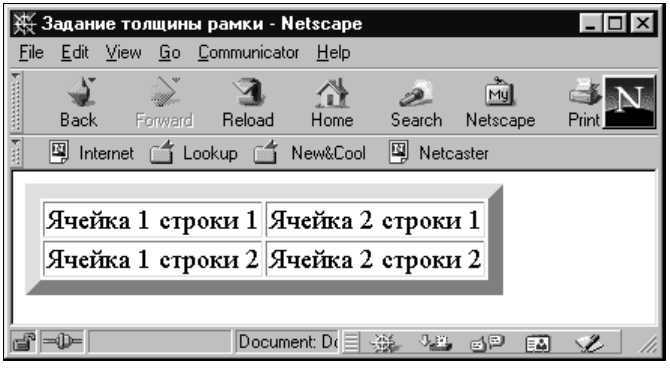


Рис. 4.6. Таблица с рамкой толщиной 10 пикселов

В спецификации HTML 3.0 не было включено значение для параметра BORDER. Это сделано лишь в HTML 3.2. Так, в частности, ранние версии Microsoft Internet Explorer не разрешали задания численного значения.

Отметим, что при отсутствии параметра BORDER рамки не прорисовываются, но место под них оставляется (это относится только к Netscape). Общий размер таблицы при отсутствии параметра BORDER или его наличии не изме-

няется (исключением является случай задания BORDER=0). Таким образом, минимальное расстояние между двумя соседними ячейками в этих случаях будет равно удвоенной толщине рамки, т. е. двум пикселам. Расположить ячейки как можно ближе друг к другу возможно заданием BORDER=0, что означает отсутствие рамок. Некоторые браузеры могут не поддерживать задание численного значения параметра BORDER, тогда значение, равное нулю, будет проигнорировано, и таблица будет прорисована с рамками.

Приведем несколько примеров:

```
<TABLE BORDER>
<TABLE BORDER=0>
<TABLE>
```

Все три приведенных примера браузером Netscape будут отображены поразному. Заметим, что здесь имеет место довольно уникальный случай, когда нельзя говорить о значении по умолчанию. Третий пример, в котором параметр BORDER опущен, отличается от любого примера, где этот параметр присутствует. Для Microsoft Internet Explorer второй и третий примеры идентичны, поэтому для этого браузера значение по умолчанию параметра BORDER равно нулю.

Параметр *CELLSPACING*

Форма записи параметра: CELLSPACING=num, где num — численное значение параметра в пикселах, которое не может быть опущено. Величина num определяет расстояние между смежными ячейками (точнее между рамками ячеек) как по горизонтали, так и по вертикали. По умолчанию значение принимается равным двум. Заметим, что традиционно в издательских системах смежные ячейки таблицы имеют общую границу. В HTML-таблицах по умолчанию между ними оставляется место, что хорошо видно на приведенном выше рисунке (рис. 4.6). При задании CELLSPACING=0 рамки смежных ячеек сольются и создадут впечатление единой сетки таблицы (рис. 4.7).

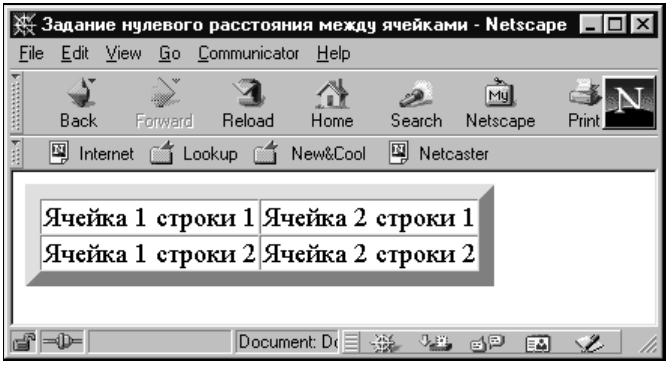


Рис. 4.7. Таблица со значением CELLSPACING=0

Параметр *CELLPADDING*

Форма записи параметра аналогична CELLSPACING. Величина num определяет размер свободного пространства (отступа) между рамкой ячейки и данными внутри ячейки. По умолчанию значение принимается равным единице. Установка параметра CELLPADDING равным нулю может привести к тому, что некоторые части текста ячейки могут касаться ее рамки, что выглядит не очень эстетично.

На рис. 4.8 показан пример таблицы со значением CELLPADDING=10.

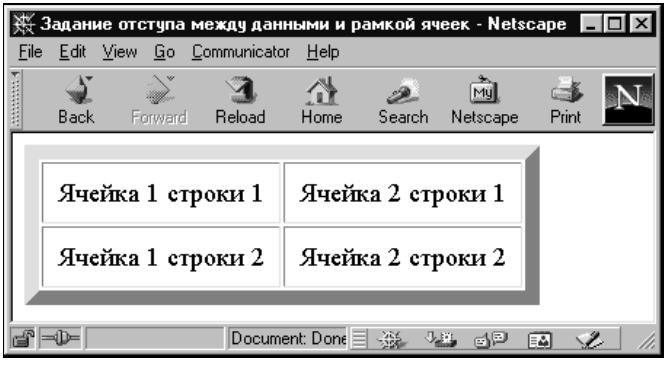


Рис. 4.8. Таблица со значением CELLPADDING=10

Действие параметров CELLPADDING и CELLSPACING очень похоже друг на друга. Для таблицы без рамок изменение того или другого параметра приводит к одному и тому же результату. Оба параметра влияют на соответствующие отступы одновременно по горизонтали и по вертикали. К сожалению, раздельного управления горизонтальными и вертикальными отступами так, как это сделано, например, для отступов от изображений (параметры HSPACE и VSPACE тэга <IMG>), не предусмотрено.

Все три параметра — BORDER, CELLPADDING и CELLSPACING действуют независимо друг от друга, если какой-нибудь из них опущен, то берется его значение, принятое по умолчанию. В частности, если опущены все перечисленные параметры, то минимальное расстояние между данными из смежных ячеек будет равно 6 пикселам (для Netscape). Это значение складывается из двух пикселов для CELLSPACING, одного пиксела для CELLPADDING и одного пиксела для рамки каждой из ячеек. Наиболее компактная таблица будет получена заданием следующего описания:

```
<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0>
```

Только в таком варианте ячейки будут расположены вплотную друг к другу. Примером использования может служить таблица, все ячейки которой содержат рисунки одинакового размера, которые необходимо расположить рядом друг с другом.

Параметры *WIDTH* и *HEIGHT*

При отображении таблиц их ширина и высота автоматически вычисляются браузером и зависят от многих факторов: значений параметров, заданных в описании всего документа, данной таблицы, отдельных ее строк и ячеек, содержимого ячеек, а также параметров, задаваемых при просмотре документа в том или ином браузере, например, типа и размеров шрифта, размеров окна просмотра и др. При отображении расчет размеров таблиц выполняется автоматически с учетом этих факторов, при этом делается попытка представить таблицу в наиболее удобном виде — расположить таблицу так, чтобы она помещалась в окне просмотра. Общая схема просмотра больших документов, как правило, сводится к линейной прокрутке содержимого документа по вертикали и чтении текста, перемежаемого различными таблицами, изображениями и т. п. Это относится как к HTML-документам, так и к обычным документам, создаваемым в любых текстовых редакторах. Большинство как текстовых редакторов (например, Microsoft Word), так и HTML-браузеров автоматически форматируют текст так (если возможно), чтобы длина строк не превосходила ширину окна просмотра. Это позволяет избежать необходимости горизонтальной прокрутки документа. Аналогичные действия предпринимаются браузерами с таблицами — по возможности форматировать их таким образом, чтобы ширина таблицы не превосходила ширины окна просмотра. Можно сделать вывод, что ширина таблиц является более важным, первостепенным параметром, расчет которого выполняется в первую очередь по сравнению с высотой.

В большинстве случаев динамическое определение размеров таблицы дает в результате эстетически соразмерное изображение с эффективным использованием реальной площади окна просмотра. Однако бывает необходимо принудительно указывать ширину или высоту таблицы. Для этой цели используются параметры `WIDTH` (ширина таблицы) и `HEIGHT` (высота таблицы) тэга `<TABLE>`. Форма записи: `WIDTH=num` или `WIDTH=num%`, где `num` — численное значение ширины всей таблицы в пикселах или в процентах от всего размера окна. Заметим, что допустимо задавать значения, большие 100%, хотя трудно представить себе случай, где это необходимо. Пример:

```
<TABLE WIDTH=200>.
```

Аналогичные параметры могут задаваться и для отдельных ячеек. Заметим, что задание конкретного значения параметра, например `WIDTH=200`, не означает, что таблица в любом случае будет иметь указанную ширину, а лишь определяет рекомендуемую ширину, которая будет выдержана по возможности. Поясним это на примерах. Пусть имеется таблица, которая в данных условиях по умолчанию имела бы ширину, меньшую заданной. В этом случае браузер увеличит ширину таблицы до требуемой путем пропорционального расширения всех колонок таблицы. При сужении окна просмотра ширина таблицы изменяться не будет, и, возможно, для ее просмотра потребуется горизонтальная прокрутка. Если же таблица по умолчанию имеет

ширину, большую заданной, то браузер сделает попытку уменьшить ее ширину за счет, во-первых, сокращения ширины отдельных колонок, для которых заданная ширина больше необходимой, во-вторых, разбиением текста в отдельных ячейках на несколько строк с увеличением высоты таблицы. Эти действия могут не обеспечить требуемого размера таблицы, и тогда она будет иметь минимально возможную ширину. Такие же действия предпринимаются для таблиц, у которых не указаны размеры, при сужении окна просмотра.

Конкретные алгоритмы настройки таблиц для различных браузеров могут несколько отличаться.

Параметр *ALIGN*

Данный параметр тэга `<TABLE>` определяет горизонтальное расположение таблицы в области просмотра. Допустимые значения — `LEFT` (выравнивание влево) и `RIGHT` (выравнивание вправо). По умолчанию таблицы выровнены по левому краю. Заметим, что среди допустимых значений нет типичного значения для параметра выравнивания — `CENTER`. В некоторых источниках по языку HTML значение `CENTER` (по центру) приводится в качестве допустимого в данном случае. Это соответствует спецификации HTML, но на практике и Netscape Navigator, и Microsoft Internet Explorer реализуют только два значения. Дело в том, что присутствие параметра `ALIGN` в тэге `<TABLE>` не только определяет месторасположение таблицы, но и разрешает выполнить обтекание таблицы текстом с противоположной стороны аналогично обтеканию картинок. Обтекание таблицы текстом с двух сторон не предусматривается ни в каких случаях. Для более точного управления обтеканием следует использовать тэг `<BR>` с параметром `CLEAR` так же, как это выполняется для `<IMG>`. Если параметр `ALIGN` опущен, то место справа и/или слева от таблицы всегда будет пустым независимо от ее ширины. Если таблица не требует обтекания текстом, то можно добиться ее расположения по центру окна просмотра. Для этого, например, можно все описание таблицы поместить внутри пары тэгов `<CENTER>` и `</CENTER>`.

Приведем пример таблицы с обтекающим текстом, отображение которой показано на рис. 4.9.

```
<HTML>
<HEAD>
<TITLE>Таблица с обтекающим ее текстом</TITLE>
</HEAD>
<BODY>
<TABLE ALIGN=LEFT WIDTH=70%>
<CAPTION><H3>Наиболее употребительные мужские имена<BR> взрослого
населения Санкт-Петербурга</H3>
</CAPTION>
<UL>
```

```
<TR><TD VALIGN=TOP>
<LI>Абрам <LI>Александр <LI>Алексей <LI>Альберт <LI>Анатолий <LI>Андрей
<LI>Аркадий <LI>Борис <LI>Вадим <LI>Валентин <LI>Валерий <LI>Василий
<LI>Виктор <LI>Виталий <LI>Владимир <LI>Владислав <LI>Вячеслав
<LI>Геннадий <LI>Георгий <LI>Герман <LI>Григорий <LI>Дмитрий
</TD>
<TD VALIGN=TOP>
<LI>Евгений <LI>Ефим <LI>Иван <LI>Игорь <LI>Илья <LI>Иосиф <LI>Константин
<LI>Лев <LI>Леонид <LI>Михаил <LI>Николай <LI>Олег <LI>Павел <LI>Петр
<LI>Роман <LI>Семен <LI>Сергей <LI>Станислав <LI>Эдуард <LI>Юрий <LI>Яков
</TD>
</UL>
</TR>
</TABLE>
<BR><BR><BR><BR><BR><BR>
Приведенные данные получены на основе анализа репрезентативной выборки,
содержащей сведения о 5000 мужчин в возрасте старше 18 лет, проживающих
в Санкт-Петербурге.<BR> Указанные 43 наиболее часто встречаемых имени
охватывают 92% выборки.<BR> Частота встречаемости каждого из остальных
имен не превосходит 0.3%
</BODY>
</HTML>
```

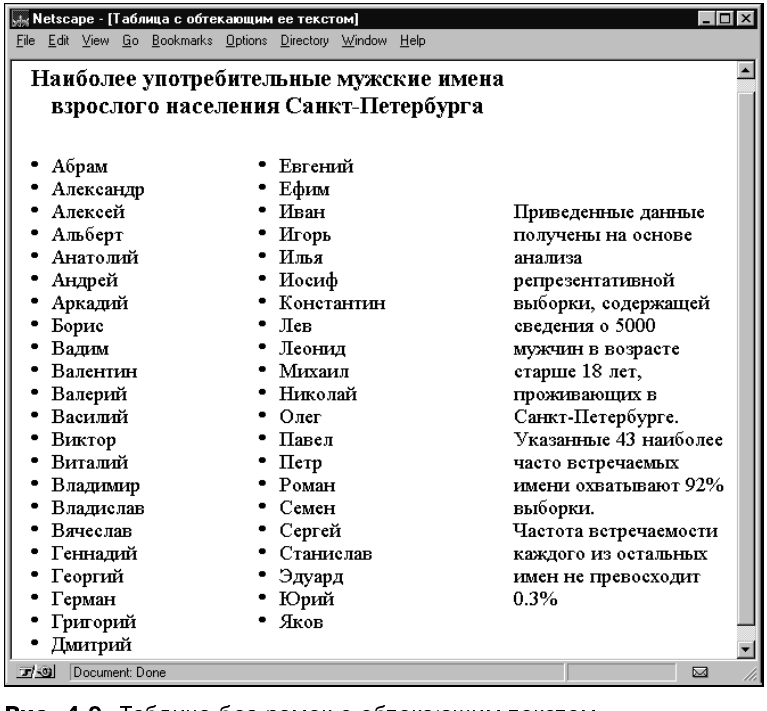


Рис. 4.9. Таблица без рамок с обтекающим текстом

Этот документ состоит из таблицы без рамок с параметром выравнивания ALIGN=LEFT, что позволяет тексту, следующему за таблицей, расположиться справа от нее. Таблица состоит всего из одной строки, в которой содержится две ячейки. Каждая ячейка содержит часть ненумерованного списка <UL>. Использование таблицы для вывода списка — это один из способов принудительного расположения списка в несколько колонок, что также иллюстрирует данный пример. Текст, расположенный справа от таблицы, может весь там не поместиться, при этом он будет продолжен после таблицы. Попробуйте на данном примере уменьшать ширину окна просмотра браузера, и в какой-то момент весь текст окажется снизу таблицы. Напомним, что для принудительного прерывания обтекания текста вдоль таблицы (например, если последующий текст логически не связан с таблицей и должен располагаться ниже ее) следует воспользоваться кодом
 с установленным параметром CLEAR. Для данного примера нужно записать <BR CLEAR=LEFT> или <BR CLEAR=ALL>. Некоторые браузеры разрешают запись параметра CLEAR без значения, но этого делать не рекомендуется. Для осуществления той же задачи задание нескольких переводов строки
 без параметра CLEAR (как это сделано в примере перед текстом для его сдвига вниз на несколько строк) или нескольких кодов начала нового абзаца <P> — неверное решение.

Приведем несколько иной пример для создания подобной страницы, отображение которой показано на рис. 4.10.

```
<HTML>
<HEAD>
<TITLE>Таблица без обтекающего текста</TITLE>
</HEAD>
<BODY>
<TABLE>
<CAPTION><H3>Наиболее употребительные мужские имена<BR>
взрослого населения Санкт-Петербурга</H3>
</CAPTION>
<UL>
<TR><TD VALIGN=TOP>
<LI>Абрам <LI>Александр <LI>Алексей <LI>Альберт <LI>Анатолий <LI>Андрей
<LI>Аркадий <LI>Борис <LI>Вадим <LI>Валентин <LI>Валерий <LI>Василий
<LI>Виктор <LI>Виталий <LI>Владимир <LI>Владислав <LI>Вячеслав
<LI>Геннадий <LI>Георгий <LI> Герман <LI>Григорий <LI>Дмитрий
</TD>
<TD VALIGN=TOP>
<LI>Евгений <LI>Ефим <LI>Иван <LI>Игорь <LI>Илья <LI>Иосиф <LI>Константин
<LI>Лев <LI>Леонид <LI>Михаил <LI>Николай <LI>Олег <LI>Павел <LI>Петр
<LI>Роман <LI>Семен <LI>Сергей <LI>Станислав <LI>Эдуард <LI>Юрий <LI>Яков
</TD>
</UL>
<TD WIDTH=200>
```

```
Приведенные данные получены на основе анализа репрезентативной выборки,
содержащей сведения о 5000 мужчин в возрасте старше 18 лет, проживающих
в Санкт-Петербурге.<BR> Указанные 43 наиболее часто встречаемых имени
охватывают 92% выборки.<BR> Частота встречаемости каждого из остальных
имен не превосходит 0.3%
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

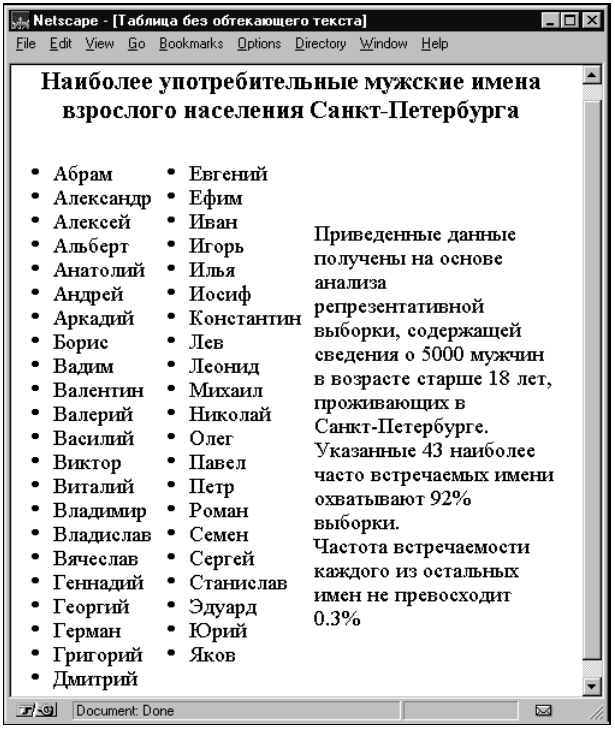


Рис. 4.10. Таблица без рамок, содержащая три столбца

В отличие от предыдущего примера здесь нет текста, обтекающего таблицу. Весь документ состоит из одной таблицы с заголовком, содержащей три ячейки в одной строке. Первые две ячейки полностью повторяют предыдущий пример. В третьей ячейке располагается текст, который комментирует содержание первых двух ячеек. Здесь нет необходимости задавать принудительное обрывание текста, как это описано в предыдущем случае. Весь текст, относящийся к таблице, должен располагаться внутри третьей ячейки, а последующий текст — после окончания описания всей таблицы `</TABLE>`. Оба примера при просмотре на полном экране выглядят одинаково, за исключением заголовка, который в первом случае расположен посередине

двухколонного списка, а во втором — располагается посередине всех трех колонок таблицы. Однако при уменьшении области просмотра в последнем примере никакая часть текста не может перейти ниже таблицы, тем самым нарушив ее структуру.

Форматирование данных внутри таблицы

Каждую отдельную ячейку внутри таблицы можно рассматривать как область для независимого форматирования. Все правила, которые действуют для управления отображением текста, могут быть использованы для форматирования текста внутри ячейки. Внутри ячейки допустимо использование практически всех элементов HTML, которые могут появляться внутри тела документа `<BODY>`, в том числе тэги, управляющие расположением текста — `<P>`, `<BR>`, `<HR>`, коды заголовков — от `<H1>` до `<H6>`, тэги форматирования символов — `<B>`, `<I>`, `<STRONG>`, `<BIG>`, `<EM>`, `<FONT SIZE>`, `<FONT COLOR>`, тэги вставки графических изображений `<IMG>`, гипертекстовых ссылок `<A>` и т. д. Сразу же подчеркнем, что область действия тэгов, заданных внутри отдельной ячейки, ограничивается пределами этой ячейки независимо от наличия завершающего тэга. Например, если внутри ячейки определен цвет текста — `<FONT COLOR=RED>`, то даже при отсутствии завершающего кода `</FONT>` или расположения его через несколько ячеек или строк таблицы, текст следующей ячейки будет отражен цветом по умолчанию.

Для форматирования данных внутри ячеек таблицы предусмотрены следующие параметры.

Параметры выравнивания содержимого ячеек — `ALIGN` и `VALIGN`. Могут применяться в кодах `<TR>`, `<TD>` и `<TH>`. Параметр горизонтального выравнивания `ALIGN` может принимать значения `LEFT`, `RIGHT` и `CENTER` (по умолчанию `LEFT` для `<TD>` и `CENTER` для `<TH>`). Параметр вертикального выравнивания `VALIGN` может принимать значения `TOP` (по верхнему краю), `BOTTOM` (по нижнему краю), `MIDDLE` (посередине), `BASELINE` (по базовой линии). По умолчанию — `MIDDLE`. Выравнивание по базовой линии обеспечивает привязку текста отдельной строки во всех ячейках к единой линии. Задание параметров выравнивания на уровне кода `<TR>` определяет выравнивание для всех ячеек данной строки, при этом в каждой отдельной ячейки строки может быть определены свои параметры, переопределяющие действие параметров, заданных в `<TR>`.

Приведем пример таблицы, в которой данные в ячейках первого столбца выровнены вправо, второго столбца — по центру, а третьего — влево (значение по умолчанию):

```
<HTML>
<HEAD>
<TITLE>Выравнивание элементов таблицы</TITLE>
</HEAD>
```

```
<BODY>
<TABLE BORDER WIDTH=100%>
<TR>
<TD ALIGN=RIGHT>Ячейка 1</TD>
<TD ALIGN=CENTER>Ячейка 2</TD>
<TD>Ячейка 3</TD>
</TR>
<TR>
<TD ALIGN=RIGHT>Ячейка 4</TD>
<TD ALIGN=CENTER>Ячейка 5</TD>
<TD>Ячейка 6</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Отображение этого примера браузером показано на рис. 4.11.

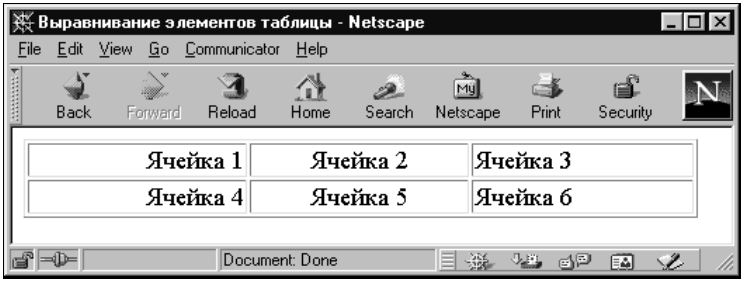


Рис. 4.11. Выравнивание данных в ячейках таблицы

Параметр `NOWRAP` отключает возможность автоматического разбиения текста ячейки на строки. Может применяться в кодах `<TR>`, `<TD>` и `<TH>`. Следует избегать неоправданного применения этого параметра, так как это может значительно сократить возможности динамического изменения размеров таблиц и ухудшить их восприятие. В большинстве случаев достаточно применить `NOWRAP` для отдельных ячеек, действительно требующих запрещения переноса слов на новую строчку. Перенос слов осуществляется только по разделителям между словами (пробелам), и в ряде случаев для запрещения разрыва текста в отдельных местах следует вместо символа пробела задавать код неразрывного пробела ` ` (NonBreaking Space). В качестве примеров можно привести случаи, где разрыв не рекомендуется — между числовым значением и единицами измерения данной величины; между фамилией и инициалами. Так, текст `650 км` или `Ельцин Б.Н.` рекомендуется записывать в виде `650 км` и `Ельцин Б.Н.`

Параметры `WIDTH` и `HEIGHT` могут применяться в кодах `<TD>` и `<TH>`. Их синтаксис аналогичен синтаксису этих параметров для тэга `<TABLE>`. Их значе-

ние определяет ширину или высоту ячейки, для которой записаны данные параметры. Значения могут задаваться в пикселах или в процентах от размеров всей таблицы. Microsoft Internet Explorer разрешает задавать значение WIDTH только в пикселах. Поскольку таблица представляет собой связную структуру, состоящую из строк и колонок, то задание ширины для какой-либо ячейки влияет на ширину всей колонки, в которой расположена ячейка, а задание высоты влияет на всю строчку. Если в колонке значение ширины указано лишь в одной ячейке, то данное значение становится шириной всей колонки. Если таких указаний несколько, то выбирается максимальное значение. Те же свойства характерны и для строк.

Для сложных таблиц характерна потребность в объединении нескольких смежных ячеек по горизонтали или по вертикали в одну. Данная возможность реализуется с помощью параметров COLSPAN (COLumn SPANning) и ROWSPAN (ROW SPANning), задаваемых в кодах <TD> или <TH>. Форма записи: COLSPAN=num, где num — числовое значение, определяющее, на сколько столбцов следует расширить текущую ячейку по горизонтали. Применение параметра ROWSPAN аналогично, только здесь указывается количество строк, которые должна захватить текущая ячейка по вертикали. По умолчанию для этих параметров устанавливается значение, равное единице. Допустимо одновременное задание значений обоих параметров для одной ячейки. Правильная установка значений этих параметров может оказаться не очень простой задачей, тем более, что большинство HTML-редакторов позволяют визуально конструировать с последующей генерацией HTML-кодов лишь простейшие таблицы.

На рис. 4.12 показан пример отображения таблицы, полученный по следующему HTML-коду:

```
<HTML>
<HEAD>
<TITLE>Использование параметров COLSPAN и ROWSPAN</TITLE>
</HEAD>
<BODY>
<TABLE BORDER>
<TR>
<TD ROWSPAN=2>Ячейка, захватывающая две строки</TD>
<TD COLSPAN=2>Ячейка, захватывающая два столбца</TD>
</TR>
<TR>
<TD>Ячейка 3</TD>
<TD>Ячейка 4</TD>
</TR>
<TR>
<TD>Ячейка 5</TD>
<TD>Ячейка 6</TD>
```

```
<TD>Ячейка 7</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Рис. 4.12. Таблица с ячейками, распространяющимися на несколько строк или столбцов

Невнимательное задание значений параметров раздвижки ячеек может привести к их взаимному перекрытию и конфликтам, при которых результат непредсказуем. Характерное применение протяженных ячеек — общий заголовок для нескольких смежных колонок или строк.

Приведем пример кода HTML (отображение которого показано на рис. 4.13), в котором протяженные ячейки сформированы некорректно.

```
<HTML>
<HEAD>
<TITLE>Неверное использование протяженных ячеек</TITLE>
</HEAD>
<BODY>
<TABLE BORDER WIDTH=100%>
  <TR ALIGN=CENTER>
    <TD>Ячейка 1</TD>
    <TD>Ячейка 2</TD>
    <TD ROWSPAN=3>
        Ячейка 3<BR>(распространенная<BR>на три<BR>строчки)
    </TD>
  </TR>
  <TR ALIGN=CENTER><TD>Ячейка 4</TD><TD>Ячейка 5</TD></TR>
  <TR ALIGN=CENTER>
    <TD>Ячейка 6</TD>
    <TD COLSPAN=2>Ячейка 7 (распространенная на два столбца)</TD>
  </TR>
```

```
</TABLE>
</BODY>
</HTML>
```

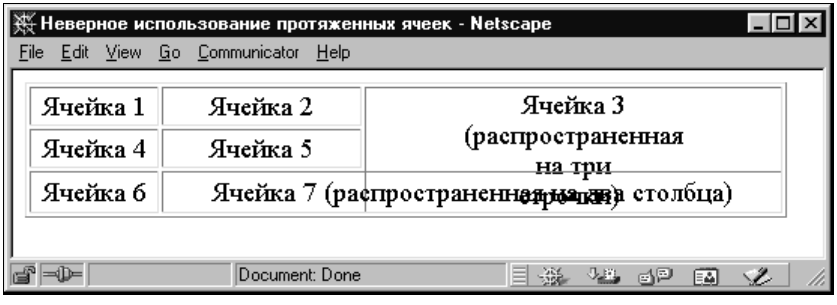


Рис. 4.13. Результат некорректного определения протяженных ячеек (наложение текста)

Параметр `BGCOLOR` задает цвет подложки всей таблицы, отдельных строк или ячеек. Может встречаться в тэгах `<TABLE>`, `<TR>`, `<TD>` и `<TH>`. Эта возможность не предусматривается спецификацией HTML, однако поддерживается как Netscape, так и Microsoft Internet Explorer. Форма записи такая же, как и для тэга `<BODY>`, а именно: `BGCOLOR=значение`, где в качестве значения задается содержание цвета в RGB-формате или его название.

Пример: `<TD BGCOLOR=#FF0000>` или `<TD BGCOLOR=red>`.

Вложенные таблицы

Отдельные ячейки таблицы могут содержать практически любые тэги языка и данные, разрешенные в разделе `<BODY>` документа. В том числе, внутри ячейки таблицы может быть целиком размещена другая таблица. Такие таблицы называются вложенными. Правила их построения не отличаются от построения таблиц и не нуждаются в отдельном описании. Отметим лишь, что не все браузеры, поддерживающие таблицы, правильно отражают сложные таблицы с несколькими уровнями вложенности, поэтому их использование требует осторожности.

Приведем пример таблицы, использующей один уровень вложенности.

```
<HTML>
<HEAD>
<TITLE>Города Ленинградской области</TITLE>
</HEAD>
<BODY>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
<CAPTION><H2>Города Ленинградской области</H2>
Н — население города (тыс.жит.,1992 г.)   
```

```
Р — расстояние от Санкт-Петербурга (км)</CAPTION>
<TR><TD VALIGN=TOP>
<TABLE BORDER CELLPADDING=3 CELLSPACING=0>
<CAPTION><STRONG>Города, подчиненные Санкт-Петербургу</STRONG></CAPTION>
<TR><TH>Город</TH><TH>Н</TH><TH>Р</TH></TR>
<TR><TD>Зеленогорск </TD><TD ALIGN=RIGHT> 13.6</TD><TD ALIGN=RIGHT>
50</TD></TR>
<TR><TD>Колпино </TD><TD ALIGN=RIGHT>144.6</TD><TD ALIGN=RIGHT>
26</TD></TR>
<TR><TD>Кронштадт </TD><TD ALIGN=RIGHT> 45.2</TD><TD ALIGN=RIGHT>
48</TD></TR>
<TR><TD>Ломоносов </TD><TD ALIGN=RIGHT> 42.0</TD><TD ALIGN=RIGHT>
40</TD></TR>
<TR><TD>Павловск </TD><TD ALIGN=RIGHT> 25.4</TD><TD ALIGN=RIGHT>
30</TD></TR>
<TR><TD>Петродворец </TD><TD ALIGN=RIGHT> 83.8</TD><TD ALIGN=RIGHT>
29</TD></TR>
<TR><TD>Пушкин </TD><TD ALIGN=RIGHT> 95.1</TD><TD ALIGN=RIGHT>
24</TD></TR>
<TR><TD>Сестрорецк </TD><TD ALIGN=RIGHT> 34.9</TD><TD ALIGN=RIGHT>
35</TD></TR>
</TABLE>
<P>
<CENTER>
Все города, подчиненные<BR>администрации<BR>Санкт-Петербурга,
имеют<BR>прямые городские<BR>телефонные номера.
</CENTER>
</TD>
<TD WIDTH=50></TD>
<TD VALIGN=TOP>
<TABLE BORDER CELLPADDING=3 CELLSPACING=0>
<CAPTION><STRONG>Города областного подчинения</CAPTION>
<TR><TH>Город</TH><TH>Н</TH><TH>Р</TH></TR>
<TR><TD>Бокситогорск </TD><TD ALIGN=RIGHT> 21.6</TD><TD
ALIGN=RIGHT>245</TD></TR>
<TR><TD>Волхов </TD><TD ALIGN=RIGHT> 50.3</TD><TD
ALIGN=RIGHT>122</TD></TR>
<TR><TD>Всеволожск </TD><TD ALIGN=RIGHT> 32.9</TD><TD ALIGN=RIGHT>
24</TD></TR>
<TR><TD>Выборг </TD><TD ALIGN=RIGHT> 80.9</TD>
<TD ALIGN=RIGHT>130</TD></TR>
<TR><TD>Высоцк </TD><TD ALIGN=RIGHT> 1.0</TD><TD
ALIGN=RIGHT>159</TD></TR>
<TR><TD>Гатчина </TD><TD ALIGN=RIGHT> 80.9</TD>
<TD ALIGN=RIGHT> 46</TD></TR>
<TR><TD>Ивангород </TD><TD ALIGN=RIGHT> 11.9</TD><TD
ALIGN=RIGHT>147</TD></TR>
<TR><TD>Каменногорск </TD><TD ALIGN=RIGHT> 5.9</TD>
<TD ALIGN=RIGHT>157</TD></TR>
```

```
<TR><TD>Кингисепп </TD><TD ALIGN=RIGHT> 51.5</TD><TD
ALIGN=RIGHT>138</TD></TR>
<TR><TD>Кириши </TD><TD ALIGN=RIGHT> 53.8</TD><TD
ALIGN=RIGHT>115</TD></TR>
<TR><TD>Кировск </TD><TD ALIGN=RIGHT> 23.8</TD><TD ALIGN=RIGHT>
55</TD></TR>
<TR><TD>Лодейное Поле</TD><TD ALIGN=RIGHT> 27.3</TD><TD
ALIGN=RIGHT>244</TD></TR>
<TR><TD>Луга </TD><TD ALIGN=RIGHT> 41.8</TD><TD ALIGN=RIGHT>139</TD></TR>
</TABLE>
</TD>
<TD WIDTH=50></TD>
<TD VALIGN=TOP>
<TABLE BORDER CELLPADDING=3 CELLSPACING=0>
<CAPTION><STRONG>(продолжение таблицы)</CAPTION>
<TR><TH>Город</TH><TH>Н</TH><TH>Р</TH></TR>
<TR><TD>Любань </TD><TD ALIGN=RIGHT> 4.7</TD><TD ALIGN=RIGHT>
85</TD></TR>
<TR><TD>Новая Ладога </TD><TD ALIGN=RIGHT> 11.2</TD><TD
ALIGN=RIGHT>141</TD></TR>
<TR><TD>Отрадное </TD><TD ALIGN=RIGHT> 22.9</TD><TD
ALIGN=RIGHT>40</TD></TR>
<TR><TD>Пикалево </TD><TD ALIGN=RIGHT> 25.1</TD><TD
ALIGN=RIGHT>246</TD></TR>
<TR><TD>Подпорожье </TD><TD ALIGN=RIGHT> 23.1</TD><TD
ALIGN=RIGHT>285</TD></TR>
<TR><TD>Приморск </TD><TD ALIGN=RIGHT> 6.7</TD><TD
ALIGN=RIGHT>137</TD></TR>
<TR><TD>Приозерск </TD><TD ALIGN=RIGHT> 20.5</TD><TD
ALIGN=RIGHT>145</TD></TR>
<TR><TD>Светогорск </TD><TD ALIGN=RIGHT> 15.8</TD><TD
ALIGN=RIGHT>201</TD></TR>
<TR><TD>Сланцы </TD><TD ALIGN=RIGHT> 42.6</TD><TD
ALIGN=RIGHT>192</TD></TR>
<TR><TD>Сосновый Бор </TD><TD ALIGN=RIGHT> 57.6</TD><TD ALIGN=RIGHT>
81</TD></TR>
<TR><TD>Тихвин </TD><TD ALIGN=RIGHT> 72.0</TD><TD
ALIGN=RIGHT>200</TD></TR>
<TR><TD>Тосно </TD><TD ALIGN=RIGHT> 33.8</TD><TD ALIGN=RIGHT>
53</TD></TR>
<TR><TD>Шлиссельбург </TD><TD ALIGN=RIGHT> 12.5</TD><TD ALIGN=RIGHT>
64</TD></TR>
</TABLE>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Netscape - [Города Ленинградской области]

File Edit View Go Bookmarks Options Directory Window Help

Города Ленинградской области

Н - население города (тыс.жит.,1992 г.) Р - расстояние от Санкт-Петербурга (км)

Города, подчиненные Санкт-Петербургу

Город	Н	Р
Зеленогорск	13.6	50
Колпино	144.6	26
Кронштадт	45.2	48
Ломоносов	42.0	40
Павловск	25.4	30
Петродворец	83.8	29
Пушкин	95.1	24
Сестрорецк	34.9	35

Все города, подчиненные администрации Санкт-Петербурга, имеют прямые городские телефонные номера.

Города областного подчинения

Город	Н	Р
Бокситогорск	21.6	245
Волхов	50.3	122
Всеволожск	32.9	24
Выборг	80.9	130
Высоцк	1.0	159
Гатчина	80.9	46
Ивангород	11.9	147
Каменногорск	5.9	157
Кингисепп	51.5	138
Кириши	53.8	115
Кировск	23.8	55
Лодейное Поле	27.3	244
Луга	41.8	139

(продолжение таблицы)

Город	Н	Р
Любань	4.7	85
Новая Ладога	11.2	141
Отрадное	22.9	40
Пикалево	25.1	246
Подпорожье	23.1	285
Приморск	6.7	137
Приозерск	20.5	145
Светогорск	15.8	201
Сланцы	42.6	192
Сосновый Бор	57.6	81
Тихвин	72.0	200
Тосно	33.8	53
Шлиссельбург	12.5	64

Пуск Netscape - [Города ... Paint Shop Pro - [tb11.gif [... 12:54

Рис. 4.14. Пример вложенных таблиц

Результат отображения данного примера показан на рис. 4.14.

На первый взгляд кажется, что в примере нет вложенности таблиц. На самом деле весь документ представляет собой таблицу, не имеющую рамок и состоящую из заголовка и всего одной строки, содержащей пять ячеек. Организация такой таблицы служит единственной цели — расположению данных на странице. Внутри первой ячейки располагается другая таблица, имеющая свой заголовок и состоящая из трех столбцов, после которой идет текст, выровненный посередине. Третья и пятая ячейки также содержат отдельные таблицы. Вторая и четвертая ячейки — пустые, они не содержат никаких данных и имеют единственный параметр `WIDTH`, определяющий ее ширину. Их назначение — задать отступ между первой и третьей, а также третьей и пятой ячейками, в которых располагаются таблицы. Это один из возможных вариантов задания такого отступа. Другой вариант — использование параметра `CELLSPACING`, определяющего расстояние между ячейками, однако этот параметр задает отступы одновременно и по горизонтали, и по вертикали, что в данный момент не требуется. Кроме того, пустая ячейка с заданной шириной при сужении области просмотра будет сокращаться в отличие от пространства, заданного параметром `CELLSPACING` (равно как и `CELLPADDING`). Попробуйте на данном примере уменьшать ширину области просмотра в браузере или, что приведет к тем же результатам, увеличивать размеры шрифта, которым отображается текст. Расстояние между таблицами

сократится до нуля, давая возможность видеть одновременно всю информацию как можно дольше, однако дальнейшее изменение не приведет к порче таблицы, а предоставит возможность горизонтальной прокрутки. По аналогичной схеме можно организовать размещение информации, состоящей не только из таблиц, но и изображений, фрагментов текста и т. д.

Особенности построения таблиц

В данном разделе рассматриваются некоторые специфичные возможности отдельных браузеров, а также отдельные тонкости построения и отображения таблиц.

Отображение пустых ячеек в таблицах

Одной из особенностей представления таблиц различными браузерами является отображение пустых ячеек. Согласно описанию языка все браузеры должны дополнять строки пустыми ячейками, если в какой-либо строке их количество задано меньшим, чем в остальных строках. Кроме того, в любом месте таблицы могут находиться ячейки, не содержащие данных. Существует различие между пустыми ячейками и ячейками, содержащими невидимые данные. В пустых ячейках внутри пары тэгов `<TD>` и `</TD>` не содержится никакой информации или один или более пробелов, которые не трактуются как данные. Ячейки, содержащие невидимые данные, к примеру, могут содержать код ` ` или код перевода строки `<BR>`, или любой текст, цвет которого совпадает с цветом фона ячейки. Если ячейки, содержащие данные (пусть даже невидимые), отображаются всеми браузерами одинаково, то пустые ячейки будут показаны по-разному. Браузер Netscape пустую ячейку не показывает, т. е. место, где располагается данная ячейка, будет закрашено цветом фона страницы, а не цветом фона ячейки в отличие от ячеек, содержащих данные. Вокруг пустых ячеек не прорисовывается рамка. Пример таблицы с пустой ячейкой приведен на рис. 4.15.

Рис. 4.15. Пустая ячейка таблицы отображается по-разному различными браузерами

Microsoft Internet Explorer и те, и другие ячейки отображает цветом фона ячеек. Такой браузер как NSCA Mosaic предоставляет пользователю возможность самому определить характер выдачи пустых ячеек таблицы с помощью выбора соответствующих опций. Знание таких особенностей позволит разрабатывать таблицы, которые будут отображены подходящим образом, вне зависимости от выбранного пользователем браузера. В ряде случаев достаточно для этого вместо некоторых пустых ячеек создавать ячейки, содержащие единственный код ` `.

Выравнивание данных в столбцах таблицы

Характерной проблемой при создании таблиц является задание параметров выравнивания для отдельных строк или столбцов. Для выравнивания содержимого всех ячеек текущей строки достаточно задать требуемые параметры в коде `<TR>`. Однако чаще необходимо обеспечить одинаковое выравнивание для всех элементов одного столбца, так как в большинстве случаев в столбце располагаются однородные данные. В ранних версиях HTML для этого предлагалось использовать параметр `COLSPEC` (COLumn SPECification), который задавался в тэге `<TABLE>` и определял выравнивание и ширину каждой колонки таблицы. Для примера, задание `COLSPEC="L40 R50 C80"` определяло для трех колонок таблицы выравнивание данных в ячейках: для первой колонки — `LEFT`, для второй — `RIGHT` и для третьей — `CENTER`, а также ширину каждой колонки. По мере развития языка HTML от использования этого параметра отказались, и в настоящее время он не входит в спецификацию языка и не поддерживается большинством браузеров. В итоге для решения такой задачи в Netscape Navigator не имеется специальных средств, и единственным вариантом остается либо использование выравнивания по умолчанию, либо задание соответствующих значений в каждой ячейке, где это необходимо.

В Microsoft Internet Explorer предусмотрены специальные тэги — `<COL>` и `<COLGROUP>`. Эти тэги должны располагаться сразу же за описанием `<TABLE>` перед первым появлением тэга `<TR>`.

Параметрами тэгов `<COL>` и `<COLGROUP>` могут быть `SPAN`, определяющий количество смежных колонок, на которые распространяется действие значений параметров, и `ALIGN`, определяющий горизонтальное выравнивание данных во всех ячейках соответствующего столбца (или столбцов). Допустимыми значениями параметра `ALIGN` являются `LEFT`, `RIGHT` и `CENTER`. Для параметра `SPAN` значение по умолчанию равно единице.

Тэг `<COLGROUP>` дополнительно позволяет задавать параметр `VALIGN`, определяющий вертикальное выравнивание данных в ячейках. Допустимыми значениями параметра `VALIGN` являются `MIDDLE`, `TOP` и `BOTTOM`.

Различие между тэгами `<COLGROUP>` и `<COL>` заключается в том, что первый из них, помимо задания параметров выравнивания данных для столбцов, вы-

полняет также условное объединение нескольких столбцов в группу. Эффект такого объединения проявляется при использовании параметра RULES, который описывается ниже. По умолчанию все столбцы таблицы считаются одной группой. Тэг <COL> должен использоваться только для определения выравнивания данных в отдельных столбцах в группе.

Приведем пример. Пусть необходимо построить таблицу, содержащую 6 столбцов, причем данные в первых трех из них должны быть выровнены вправо, а следующих трех — посередине. Для решения этой задачи следует записать такой фрагмент HTML-кода:

```
<TABLE>
<COLGROUP SPAN=3 ALIGN=RIGHT>
<COLGROUP SPAN=3 ALIGN=CENTER>
 (данные для таблицы)
</TABLE>
```

Результат отображения этого кода показан на рис. 4.16.

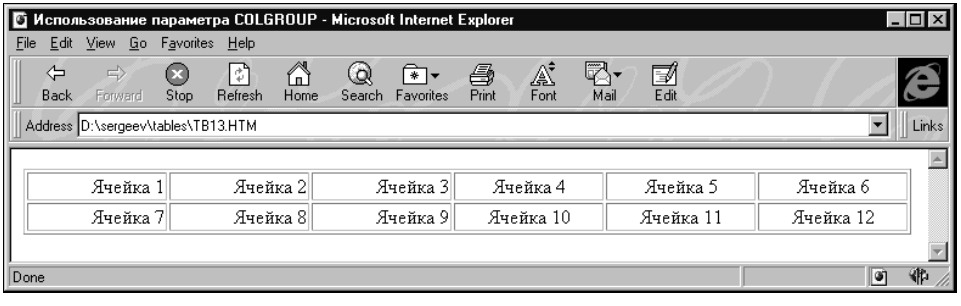


Рис. 4.16. Таблица с разными параметрами выравнивания данных в группах ячеек

Другой пример. Пусть в предыдущей таблице первые две колонки должны быть выровнены вправо, а третья — посередине, причем все три колонки необходимо объединить в группу. Последующие три колонки также должны быть объединены в группу и иметь выравнивание, аналогичное первой группе. Для решения этой задачи следует записать такой фрагмент HTML-кода:

```
<TABLE>
<COLGROUP>
<COL SPAN=2 ALIGN=RIGHT>
<COL ALIGN=CENTER>
<COLGROUP>
<COL SPAN=2 ALIGN=RIGHT>
<COL ALIGN=CENTER>
(данные для таблицы)
</TABLE>
```

В этом примере после тэга <COLGROUP> задаются настройки отдельных столбцов данной группы. При этом в тэге <COLGROUP> при необходимости могли бы быть заданы параметры выравнивания, значения которых распространяются на все столбцы данной группы. Значения параметров, заданные в тэге <COL>, переопределяют значения из тэга <COLGROUP>. Заметим, что в тэге <COLGROUP> в данном примере, в отличие от предыдущего, отсутствует параметр SPAN. Здесь его употребление бессмысленно, так как количество элементов в группе будет определяться следующими за тэгом <COLGROUP> тэгами <COL>. Поэтому любое заданное значение параметра SPAN тэга <COLGROUP> будет переопределено.

На рис. 4.17 показан результат реализации приведенного выше кода, а также вариант отображения такой таблицы с записью RULES=GROUPS в тэге <TABLE>, из которого виден смысл объединения в группы.

Совет

Поскольку область применения тэгов <COLGROUP> и <COL> ограничивается единственным браузером Microsoft Internet Explorer, то следует пользоваться ими с осторожностью. Удобство использования этих тэгов очевидно, но на практике большинство таблиц строится с использованием соответствующего параметра выравнивания ALIGN для каждой ячейки таблицы, где это необходимо, что значительно увеличивает объем исходного кода таблицы, однако обеспечивает возможность просмотра в любом браузере.

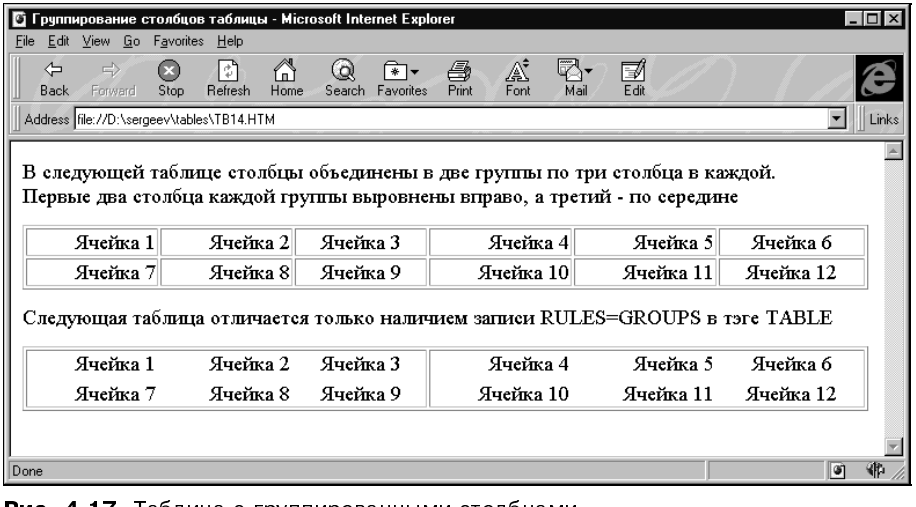


Рис. 4.17. Таблица с группированными столбцами

Задание цвета рамок таблицы

Еще несколько параметров, характерных только для Microsoft Internet Explorer, позволяют выбирать цвет рамок таблиц — BORDERCOLOR, BORDERCOLORLIGHT

и BORDERCOLORDARK. Эти параметры могут задаваться в тэгах <TABLE>, <TD>, <TH> и <TR>. В качестве значения этих параметров может использоваться название цвета или его шестнадцатеричное значение. Параметр BORDERCOLOR определяет цвет всех элементов рамок таблицы, а другие два параметра задают цвет отдельных составляющих рамок, переопределяя значение BORDERCOLOR. Параметр BORDERCOLORLIGHT окрашивает в заданный цвет левый и верхний края всей таблицы и соответственно правый и нижний края каждой ячейки. Второй параметр BORDERCOLORDARK задает цвета противоположных краев. За счет сочетания действия этих параметров таблица будет выглядеть несколько приподнятой над поверхностью страницы или углубленной. Все зависит от выбранного сочетания цветов.

Примечание

Браузер Netscape версии 4.x также поддерживает параметр BORDERCOLOR.

Задание фонового рисунка для таблицы

Браузер Microsoft Internet Explorer (а также браузер Netscape версии 4.x) разрешает использовать параметр BACKGROUND, определяющий фоновый рисунок для таблицы так, как это может быть сделано для всего HTML-документа. Этот параметр может задаваться в тэгах <TABLE>, <TD> и <TH>.

Тэги структурирования таблицы *<THEAD>*, *<TBODY>* и *<TFOOT>*

Браузер Microsoft Internet Explorer позволяет использовать ряд новых тэгов для структурирования таблиц и гибкого управления прорисовкой рамок и линий сетки.

Тэги <THEAD>, <TBODY> и <TFOOT> более строго задают структуру описания таблицы, выделяя ячейки заголовка таблицы, основное содержимое таблицы и итоговую строку. Эти тэги могут встречаться только в описании таблиц внутри пары тэгов <TABLE> и </TABLE>.

Тэги <THEAD> и <TFOOT> используются для описания верхнего и нижнего колонтитулов таблицы. Эти тэги могут встречаться в таблице не более одного раза. Завершающий тэг для них можно опускать. Использование данных тэгов удобно при создании больших таблиц, выходящих за пределы одной страницы.

Тэг <TBODY> может встречаться многократно в описании таблицы, при этом требуется использование завершающего тэга </TBODY>. Этот тэг выполняет логическое группирование данных так же, как и тэг <COLGROUP>, выполняющий группирование смежных столбцов.

При использовании новых тэгов появляется возможность более гибко управлять рамками и линиями сетки таблицы.

Управление прорисовкой рамок вокруг таблицы осуществляется параметром FRAME тэга <TABLE>, а линий сетки таблицы — параметром RULES. Например, становится возможным провести только вертикальные линии между колонками и вместо рамки вокруг всей таблицы дать горизонтальные линии сверху и снизу таблицы.

Параметр FRAME может принимать следующие значения:

- BOX или BORDER — рамка рисуется со всех четырех сторон
- ABOVE — только с верхней стороны
- BELOW — только с нижней стороны
- HSIDES — рисуется нижняя и верхняя сторона
- VSIDES — рисуется левая и правая сторона
- LHS — только с левой стороны
- RHS — только с правой стороны
- VOID — таблица без внешних рамок

Параметр RULES управляет прорисовкой внутренних линий сетки таблицы и может принимать следующие значения:

- ALL — рисуются все внутренние линии
- GROUPS — рисуются только линии, разделяющие группы
- ROWS — рисуются линии, разделяющие строки
- COLS — рисуются линии, разделяющие столбцы
- NONE — внутренние линии не рисуются

Пример: <TABLE BORDER FRAME=HSIDES RULES=GROUPS>.

Примечание

Прорисовка линий сетки таблицы и рамок будет осуществляться только при наличии параметра BORDER тэга <TABLE>. При отсутствии этого параметра или его нулевом значении линии сетки и рамки будут отсутствовать при любых значениях параметров FRAME и RULES.

Приведем пример полного HTML-кода, создающего таблицу с использованием описанных возможностей:

```
<HTML>
<HEAD>
<TITLE>Выделение заголовка и итоговой строки</TITLE>
</HEAD>
<BODY>
<TABLE BORDER=5 WIDTH=100% RULES=GROUPS FRAME=HSIDES>
<COLGROUP ALIGN=CENTER>
```

```
<COLGROUP ALIGN=CENTER>
<COLGROUP ALIGN=CENTER>
<CAPTION><H3>
Пример гибкого управления линиями<BR>сетки таблицы</H3>
</CAPTION>
<THEAD>
<TR>
<TH>Заголовок столбца 1</TH>
<TH>Заголовок столбца 2</TH>
<TH>Заголовок столбца 3</TH>
</TR>
</THEAD>
<TBODY>
<TR><TD>Данные</TD><TD>Данные</TD><TD>Данные</TD></TR>
<TR><TD>Данные</TD><TD>Данные</TD><TD>Данные</TD></TR>
<TR><TD>Данные</TD><TD>Данные</TD><TD>Данные</TD></TR>
<TR><TD>Данные</TD><TD>Данные</TD><TD>Данные</TD></TR>
<TR><TD>Данные</TD><TD>Данные</TD><TD>Данные</TD></TR>
<TR><TD>Данные</TD><TD>Данные</TD><TD>Данные</TD></TR>
</TBODY>
<TFOOT>
<TR><TD>Итог</TD><TD>Итог</TD><TD>Итог</TD></TR>
</TFOOT>
</TABLE>
</BODY>
</HTML>
```

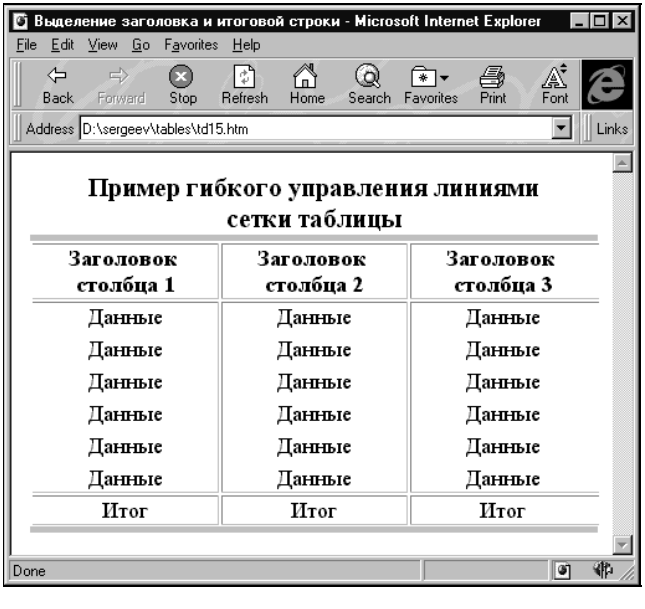


Рис. 4.18. Гибкая прорисовка линий сетки таблицы браузером Microsoft Internet Explorer

В этом примере, отображение которого браузером представлено на рис. 4.18, показывается один из возможных вариантов управления линиями сетки и рамками вокруг таблицы. Вокруг таблицы рисуется рамка толщиной 5 пикселов (BORDER=5) только с верхней и нижней стороны (FRAME=HSIDES). Внутри таблицы рисуются линии сетки, разделяющие группы данных (RULES=GROUPS). Группы данных определены, во-первых, наличием трех тэгов <COLGROUP ALIGN=CENTER>, каждый из которых объявляет отдельный столбец таблицы группой. Во-вторых, тэги <THEAD>, <TBODY> и <TFOOT> также разбивают данные таблицы на группы, что определяет прорисовку внутренних горизонтальных линий.

Задание числа столбцов таблицы

Браузер Microsoft Internet Explorer (а также браузер Netscape версии 4.x) разрешает задавать в тэге <TABLE> параметр COLS, значение которого определяет число столбцов в таблице. Запись этого параметра позволяет ускорить процедуру верстки таблицы при отображении в браузере, так как появляется возможность определить количество столбцов до окончания загрузки кода описания таблицы. На текущий момент включение этого параметра никак не влияет на ход загрузки документа.

Вертикальное выравнивание таблиц

Последний параметр тэга <TABLE>, свойственный только Microsoft Internet Explorer, это — VALIGN, определяющий вертикальное выравнивание таблицы относительно текста. Его действие подобно такому же параметру для изображений.

Примечание

Отметим, что использование одного и того же параметра может существенно различаться как по назначению, так и по возможным значениям для разных тэгов даже для одного браузера и в рамках спецификации языка. Поэтому невозможно составление сводной таблицы по использованию различных параметров вне контекста их применения. Например, параметр ALIGN только в таблицах используется в трех различных вариантах:

- для тэга <TABLE> параметр ALIGN может принимать значения LEFT или RIGHT, и означает расположение таблицы, прижатой к левому или правому краю соответственно;
- для тэга <CAPTION> параметр ALIGN принимает значения TOP или BOTTOM, и означает расположение заголовка таблицы над таблицей или под ней;
- для тэгов <TR>, <TD> и <TH> параметр ALIGN принимает значения LEFT, RIGHT или CENTER, и означает выравнивание содержимого соответствующей ячейки (или ячеек) таблицы по горизонтали.

Альтернатива табличному представлению

Поддержка таблиц стала широко распространенным свойством Web-браузеров, так что практически не осталось причин, из-за которых следовало бы избегать их использования. Тем не менее рассмотрим возможные варианты альтернативного представления данных, которые можно использовать вместо таблиц или в добавление к ним.

Некоторые иные способы, не использующие понятие таблиц:

- *Использование преформатированного текста.* Этот способ традиционно использовался в ранних версиях языка HTML, когда поддержки таблиц еще не существовало. Его употребление и до настоящего времени не потеряло актуальности, так как такие тексты будут правильно отображаться любыми браузерами, в том числе и чисто текстовыми.
- *Использование изображения, содержащего таблицу.* Таблица может быть создана любым текстовым редактором или даже отображена Web-браузером и затем сохранена как картинка в одном из графических форматов. Это не лучший вариант, так как при этом теряется вся гибкость динамической настройки отображения таблиц. Кроме того, возникает необходимость хранения дополнительного файла с изображением, размер которого к тому же, как правило, будет значительно больше, чем размер текста, описывающего HTML-таблицу. Возможная область применения — таблицы строго определенных размеров, для которых недопустима зависимость ее отображения от каких-либо внешних факторов (шрифтов, режимов работы браузера и т. п.).
- *Использование списков вместо таблиц.* Для простейших случаев вместо организации таблиц вполне возможно обойтись одним из видов списков, имеющихся в языке HTML.

Подготовка таблиц

Для подготовки HTML-таблиц могут быть использованы любые редакторы, большинство из которых имеют средства для визуального создания таблиц. Приведем пример подготовки таблицы в редакторе HotDog Professional. Для создания таблицы достаточно выбрать пункт **Tables** из меню **Insert**, после чего будет выдано диалоговое окно, показанное на рис. 4.19. Создание таблицы заключается в заполнении соответствующих полей в окне. После определения количества строк и столбцов в таблице можно перейти к непосредственному заполнению отдельных ячеек таблицы, которые будут показаны в этом же диалоговом окне. Диалоговое окно имеет кнопку **Preview**, нажатие которой позволяет просмотреть получаемую таблицу при помощи встроенного браузера (рис. 4.20).

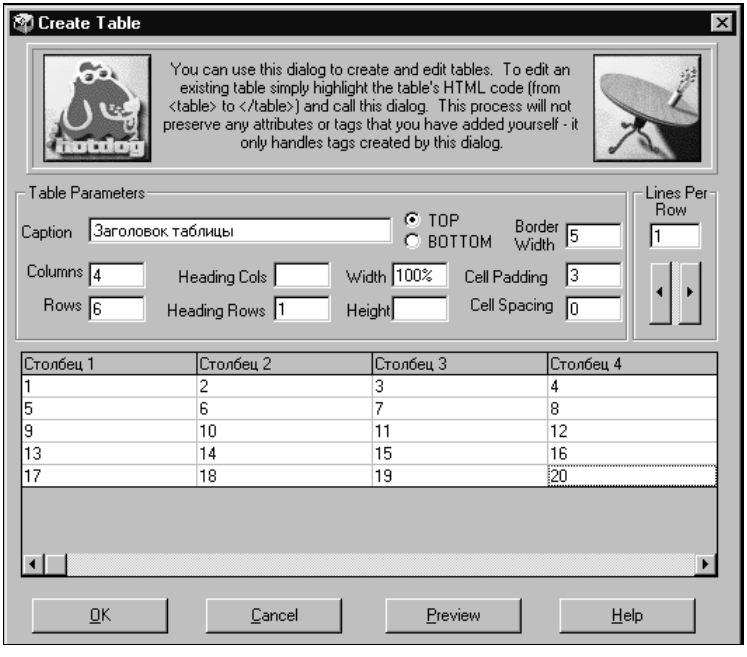


Рис. 4.19. Диалоговое окно для создания таблиц

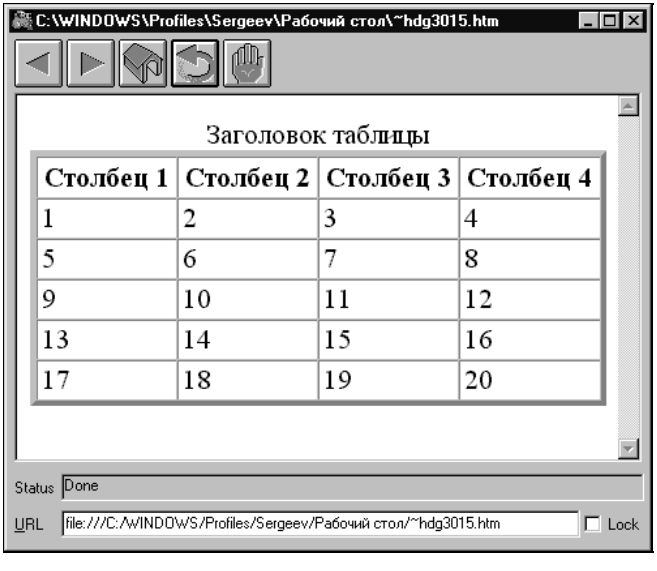


Рис. 4.20. Таблица, показанная при помощи встроенного браузера

После завершения подготовки данных для таблицы следует нажать кнопку **OK**. Тогда сгенерированный код описания таблицы будет вставлен в редактируемый HTML-документ. Для примера, приведенного на рис. 4.19, будет сгенерирован следующий код:

```
<table border=5 width=100% cellpadding=3 cellspacing=0>
<caption align=top>Заголовок таблицы</caption>
<tr>
       <th>Столбец 1</th>
       <th>Столбец 2</th>
       <th>Столбец 3</th>
       <th>Столбец 4</th>
</tr>
<tr>
       <td>1</td>
       <td>2</td>
       <td>3</td>
       <td>4</td>
</tr>
  (часть кода опущена)
</table>
```

Аналогичным образом данная задача решается при использовании компоненты Netscape Composer программы Netscape Communicator. На рис. 4.21 показано диалоговое окно, в котором необходимо заполнить нужные поля. Для ввода дополнительных параметров тэга `<TABLE>` предусмотрена кнопка **Extra HTML**. После заполнения полей диалогового окна следует нажать кнопку **Apply** и тогда будет предоставлена возможность заполнения ячеек таблицы (рис. 4.22).

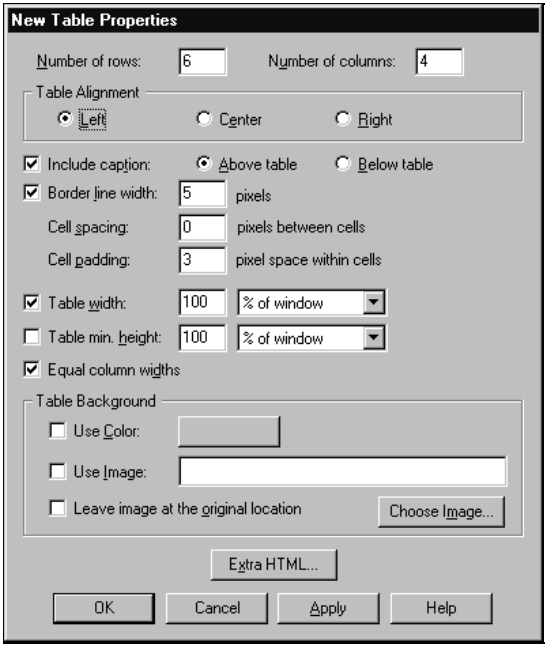


Рис. 4.21. Диалоговое окно для задания параметров таблицы программы Netscape Composer

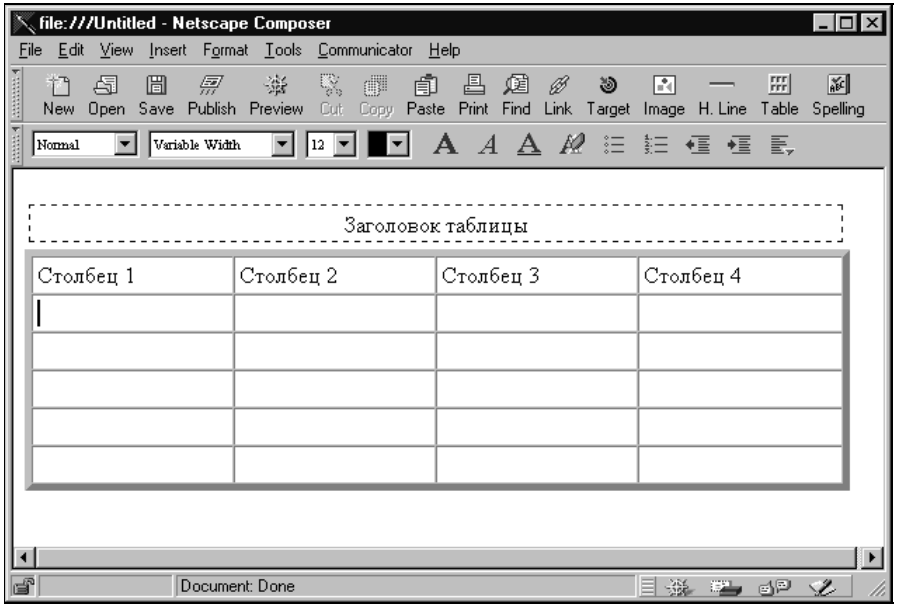


Рис. 4.22. Исходная позиция курсора ввода в пустой таблице

ГЛАВА 5

Фреймы

Фреймы позволяют разбить окно просмотра браузера на несколько прямоугольных подобластей, располагающихся рядом друг с другом. В каждую из подобластей можно загрузить отдельный HTML-документ, просмотр которого осуществляется независимо от других. Между фреймами, также как и между отдельными окнами браузера, при необходимости можно организовать взаимодействие, которое заключается в том, что выбор ссылки в одном из фреймов может привести к загрузке нужного документа в другой фрейм или окно браузера.

Возможность работы с фреймами впервые была реализована в браузере Netscape 2.0. Следующая версия браузера Netscape 3.0 обогатила возможности фреймов, добавив несколько дополнительных параметров к основным тэгам описания структуры фреймов. Браузер Microsoft Internet Explorer поддерживает фреймы, начиная с версии 3.0, а также предоставляет уникальную возможность создания плавающих фреймов.

В данной главе приводятся основные правила и примеры построения фреймов, даются рекомендации по их использованию.

Сферы применения фреймов

Разработчикам HTML-документов предоставляется довольно богатый выбор форм отображения информации на страницах. Текстовая и графическая информация может быть упорядочена и организована при помощи списков, таблиц или просто с помощью параметров выравнивания, задания горизонтальных линий, разделения на абзацы. Иногда этих возможностей оказывается недостаточно и тогда приходится разбивать окно просмотра браузера на отдельные области или *фреймы* (frames). В ряде русскоязычных описаний языка HTML вместо термина *фреймы* используется термин *кадры*. Частота использования обоих терминов примерно одинакова.

Выбор фреймовой структуры отображения информации на WWW оправдан в следующих случаях:

- при необходимости организовать управление загрузкой документов в одну из подобластей окна просмотра браузера при работе в другой подобласти;
- для расположения в определенном месте окна просмотра информации, которая должна постоянно находиться на экране вне зависимости от содержания других подобластей экрана;
- для представления информации, которую удобно расположить в нескольких смежных подобластях окна, каждая из которых может просматриваться независимо.

Приведенный список не исчерпывает все возможные случаи, где можно применить фреймы, а носит рекомендательный характер.

Рассмотрим сначала типичные варианты использования фреймов на примерах реально существующих HTML-документов, а затем обратимся к правилам разработки документов, содержащих фреймы.

На рис. 5.1 показана одна из HTML-страниц агентства "Финмаркет", специализирующегося на предоставлении информации с финансовых и фондовых рынков России.

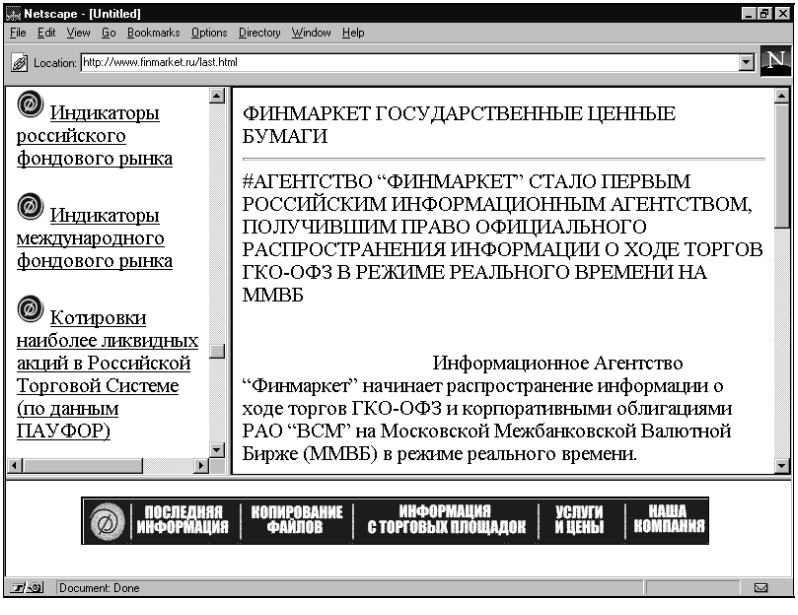


Рис. 5.1. Типичный Web-документ с фреймовой структурой

На этой странице окно браузера разбивается на три фрейма. Нижняя часть окна занимает 20% высоты всего окна и содержит постоянную информацию, которая в данном случае представляет собой графическое меню, позволяющее в любой момент обратиться к наиболее важным разделам. Этот фрейм не может изменять своих размеров по командам пользователя и не

имеет полос прокрутки. Верхняя часть окна (составляющая 80% высоты) разделена по горизонтали на два фрейма. Левый фрейм содержит оглавление документов, которые могут быть просмотрены пользователем. Правый фрейм, занимающий большую часть окна просмотра, предназначен для отображения самих документов. При первоначальной загрузке эти два фрейма делят окно браузера по горизонтали в соотношении 15% на 85%. Это соотношение может изменяться пользователем при просмотре, что позволяет выбрать оптимальные размеры фреймов с учетом содержимого загруженных документов. Каждый из этих фреймов имеет свою полосу прокрутки, обеспечивающую возможность просмотра всего содержимого фрейма вне зависимости от размера самого фрейма, всего окна браузера и используемых шрифтов. При выборе любой ссылки в левом фрейме соответствующий документ будет загружен в правый фрейм. Такая структура позволяет одновременно видеть на экране и оглавление документов, и содержимое выбранного документа.

Приведем без пояснений фрагмент HTML-кода, по которому построен документ с данной структурой:

```
<FRAMESET ROWS="80%,20%">
       <FRAMESET COLS="15%, 85%">
       <FRAME SRC="LIST.htm">
       <FRAME scrolling=auto SRC="empty.htm" name="pages">
       </FRAMESET>
<FRAME SRC = "toolbar.html" scrolling=noresize>
</FRAMESET>
```

Данный пример показывает наиболее типичное использование фреймовых структур, когда один фрейм служит оглавлением документов, а другой используется для загрузки их содержимого. Решение такой задачи без применения фреймов обычно выполняется следующим образом. На одной из страниц располагают оглавление, состоящее из ссылок на другие документы или их отдельные фрагменты. При переходе по такой ссылке оглавление исчезает, а на его место загружается нужный документ, после прочтения которого обычно необходимо вновь вернуться к оглавлению. При использовании фреймов такой возврат становится ненужным, так как оглавление постоянно располагается на части экрана.

На рис. 5.2 показан фрагмент начальной страницы электронного издания популярного в Санкт-Петербурге адресно-телефонного справочника "Весь Петербург".

Электронная версия справочника доступна по адресу **http://www.allpetersburg.ru** и позволяет находить нужные сведения по запросам пользователя. Данная страница документа также имеет фреймовую структуру и состоит из двух фреймов, причем первый из них имеет ширину 100 пикселов, а второй занимает всю оставшуюся ширину окна просмотра. Фрейм, расположенный

с левой стороны, используется для графического меню, постоянно присутствующего на экране, а также содержит логотип компании "Nevalink". Второй фрейм содержит документ, который в данном случае представляет собой форму для запроса пользователя. Структура этой страницы определена следующим HTML-кодом:

```
<FRAMESET COLS="100,*" frameborder="0" framespacing="0" border="0">
       <FRAME NAME="Menu window" SRC="menu.htm" noresize>
       <FRAME NAME=content SRC="sql.idc" noresize>
</FRAMESET>
```

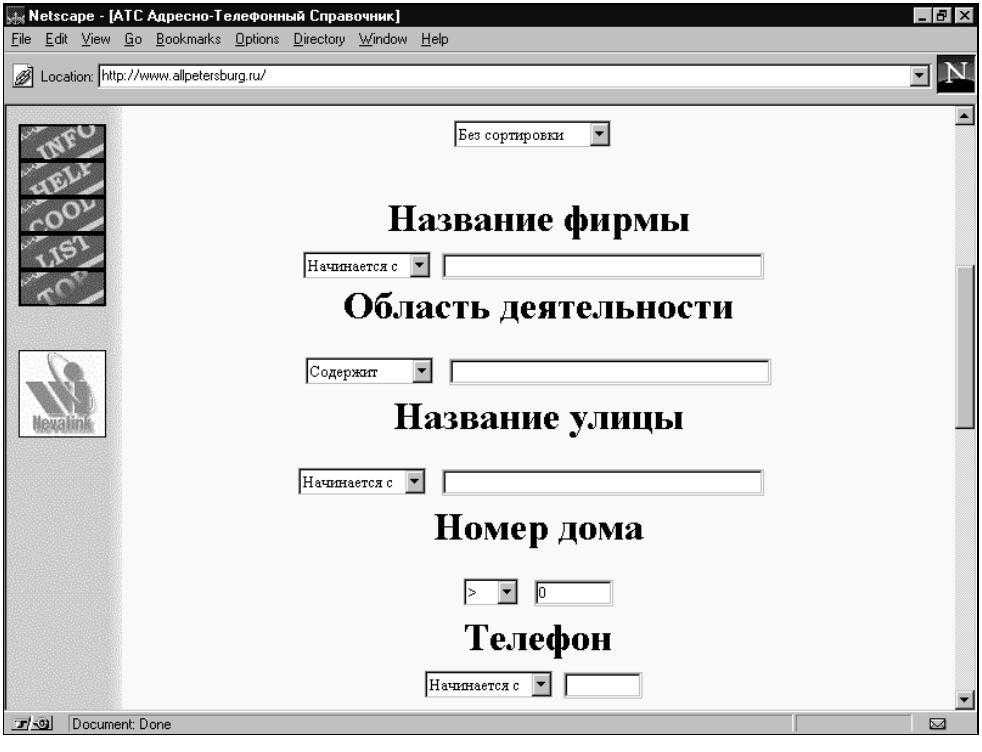


Рис. 5.2. Пример реального документа с простейшей структурой фреймов

Фреймы очень похожи на таблицы — и те и другие осуществляют разбиение окна просмотра браузера на прямоугольные области, в которых располагается некоторая информация. Однако при помощи фреймов можно решить не только задачу форматирования страниц документа, но организовать взаимодействие между ними. Принципиальная разница между фреймами и таблицами состоит в том, что каждому фрейму должен соответствовать отдельный HTML-документ, а содержимое всех ячеек таблицы всегда является частью одного документа. Кроме того, отображаемая во фрейме страница может

прокручиваться при просмотре независимо от других. Каждый фрейм по существу является отдельным "мини-браузером". В отличие от фреймов, вся структура которых всегда представлена на экране, таблицы могут полностью не помещаться в окне и быть просмотрены только по частям. Отсюда следует вывод, что если в HTML-таблицах общее число ячеек практически не ограничено и может достигать нескольких сотен, то число фреймов в документе обычно не превосходит нескольких единиц.

Совет

Если требуется только отформатировать документ, то для этого достаточно ограничиться применением таблиц. Если же необходимо решить более сложные задачи, например, организовать взаимодействие между подобластями окна или создать подобласти, постоянно расположенные на экране, то здесь удобно применить фреймы.

В конечном итоге, выбор структуры документа — табличной или фреймовой — зависит от многих факторов и не может быть однозначно предопределен.

Приведем пример еще одной страницы, которая с виду построена аналогично предыдущим. На рис. 5.3 показана страница очень популярного во всем мире сборника программных продуктов, предназначенных в основном для работы с Интернетом.

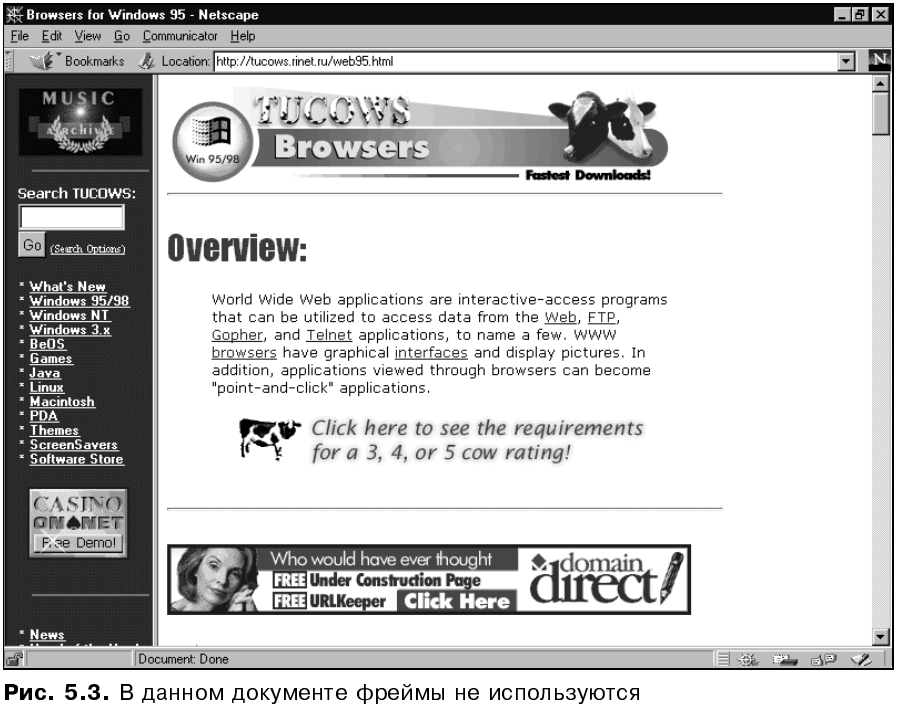


Рис. 5.3. В данном документе фреймы не используются

Адрес сервера **http://www.tucows.com**. Заметим, что имя сервера определила аббревиатура, полученная от сокращения полного названия сборника — The Ultimate Collection of Winsock Software. Поскольку сокращение tucows оказалось созвучным словосочетанию two cows (две коровы), то на страницах сервера сплошь и рядом встречается изображение коров, а рейтинг программных продуктов оценивается в количестве мычаний ("Moo") и графически изображается в виде ряда из соответствующего числа коров. Большинство страниц сервера построено однотипно — в левой части окна приводится список доступных разделов, а в правой части — перечень программных продуктов выбранного раздела. На первый взгляд структура документа должна иметь примерно такой же вид, как и в предыдущих примерах. Однако в этом документе фреймы не используются! Данная страница построена при помощи таблицы, которая состоит всего лишь из одной строки с двумя ячейками. Таблица не имеет обрамления и лишь преследует цель форматирования страницы. Впечатление разделения экрана на две части по вертикали создано путем использования фонового графического изображения, содержащего вертикальную линию, а вовсе не сеткой таблицы. В этом можно убедиться, если выполнять просмотр страницы без загрузки изображений. Использование таблицы здесь, видимо, обусловлено соображениями большей доступности документов, поскольку фреймы позволяют отображать не все браузеры.

Недостатком такого подхода в данном случае является необходимость повторения в каждом документе всего списка разделов (левая часть страницы), что незначительно увеличивает размер файлов.

Сравнение приведенных примеров показывает, что использование таблиц и фреймов может иногда быть взаимозаменяемым и определяется пожеланиями разработчиков. Заметим, что часто при взгляде на страницу с отображенным на ней документом невозможно определить, как она построена. Конечному пользователю знание внутренней структуры документа не требуется, однако при разработке собственных Web-страниц ознакомление с исходным кодом существующих документов было бы крайне полезно. В первом примере (см. рис. 5.1) фреймовая структура документа сразу же видна — наличие двух вертикальных полос прокрутки уже определяет присутствие отдельных фреймов. Последующие два примера внешне очень схожи, причем при взгляде на рисунки (рис. 5.2 и 5.3) невозможно определить, что первый из них построен с помощью фреймов, а второй — с помощью таблиц. Отличия будут проявляться только при работе с ними. В примере на рис. 5.2 при прокрутке документа левая часть окна будет оставаться на месте, что возможно только при наличии фреймовой структуры. В следующем примере (рис. 5.3) при прокрутке будет смещаться все содержимое окна.

Посмотреть структуру документа при работе с браузером Netscape можно воспользовавшись пунктом **Page Info** (в версиях 3.x браузера Netscape этот пункт меню назывался **Document Info**) меню **View** (рис. 5.4).

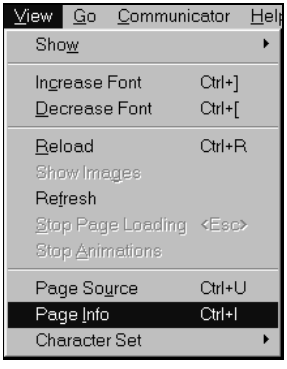


Рис. 5.4. Меню **View** браузера Netscape

Например, документ одного из описанных выше примеров (см. рис. 5.2) имеет структуру, приведенную на рис. 5.5.

Кроме того, всегда можно ознакомиться с исходным HTML-кодом всего документа, воспользовавшись пунктом **Page Source** меню **View** (или пунктом **View Frame Source** контекстного меню, вызываемого правой кнопкой мыши, для просмотра HTML-кода документа, загруженного в выбранный фрейм).

Рис. 5.5. Структура загруженного документа

Совет

Не следует без необходимости злоупотреблять использованием фреймов, причем их количество не должно превосходить трех-четырех.

Еще один пример реального использования фреймов представлен на рис. 5.6.

Здесь два смежных фрейма использованы для загрузки документов, которые удобно просматривать одновременно и сопоставлять друг с другом. В каж-

дом из двух документов, загружаемых во фреймы, использована табличная форма представления информации. В результате такой организации данных каждая из двух таблиц может быть просмотрена (или распечатана) отдельно, а может изучаться в сравнении с другой.

Наименование ЦБ			Цена, USD	Лот
"Башкирэнерго"	ао		0,500	100.000
"ЕЭС России" РАО	ао	#	0,3045	500.000
"ЕЭС России" РАО	ап		0,200	500.000
"Газпром"	ао		0,640	100.000
"Иркутскэнерго"	ао	#	0,280	500.000
"Колэнерго"	ао		0,295	100.000
"Коминефть"	ао	#	4,0	30.000
"Кубаньэлектросвязь"	ао		23,0	1.000
"Кубаньэлектросвязь"	ап		13,0	1.000
"ЛУКойл Нефтяная компания"	ао	#	14,85	5.000
"ЛУКойл Нефтяная компания"	ап	#	10,55	5.000
"Мегионнефтегаз"	ао	#	5,90	10.000

Наименование ЦБ			Цена, USD	Лот
"Башкирэнерго"	ао		0,530	100.000
"ЕЭС России" РАО	ао	#	0,3150	500.000
"ЕЭС России" РАО	ап	#	0,2120	500.000
"Газпром"	ао		0,650	100.000
"Иркутскэнерго"	ао	#	0,30	500.000
"Колэнерго"	ао		0,305	100.000
"Коминефть"	ао	#	5,50	30.000
"Кубаньэлектросвязь"	ао		24,50	1.000
"Кубаньэлектросвязь"	ап		14,0	1.000
"ЛУКойл Нефтяная компания"	ао	#	14,90	5.000
"ЛУКойл Нефтяная компания"	ап	#	11,10	5.000
"Мегионнефтегаз"	ао		6,10	10.000

Рис. 5.6. Пример использования фреймов для загрузки двух документов, которые удобно просматривать в сопоставлении друг с другом

Все приведенные примеры данного раздела взяты со страниц популярных WWW-серверов и, пожалуй, могут служить образцами применения фреймов в HTML-документах.

Последующие разделы данной главы посвящены правилам записи документов, содержащих фреймы.

Правила описания фреймов

Перейдем теперь к рассмотрению правил записи тэгов, используемых для документов с фреймовыми структурами.

Давайте для начала рассмотрим полный HTML-код, создающий документ с фреймами средней сложности:

```
<HTML>
<HEAD>
<TITLE> </TITLE>
</HEAD>
<FRAMESET ROWS="25%,50%,25%">
<FRAME SRC="header.htm">
       <FRAMESET COLS="25%,75%">
       <FRAME SRC="list.htm">
       <FRAME SRC="info.htm">
       </FRAMESET>
       <FRAME SRC="footer.htm">
</FRAMESET>
<NOFRAMES>
Ваш браузер не может отображать фреймы
</NOFRAMES>
</HTML>
```

Этот пример создает страницу с фреймами, показанную на рис. 5.7. Как вы видите, этот HTML-код определяет четыре фрейма. Верхний фрейм занимает всю ширину страницы и содержит заголовок. Далее идут два центральных фрейма, один из которых расположен с левой стороны и занимает 25 процентов от ширины экрана, а второй занимает оставшееся место. Последний, четвертый фрейм занимает нижнюю четверть экрана. В каждый из фреймов загружается отдельный HTML-документ, имя которого определяется параметром `SRC`.

Как видно из примера, для описания структуры фреймов применяются тэги `<FRAMESET>`, `<FRAME>` и `<NOFRAMES>`. Рассмотрим назначение этих тэгов.

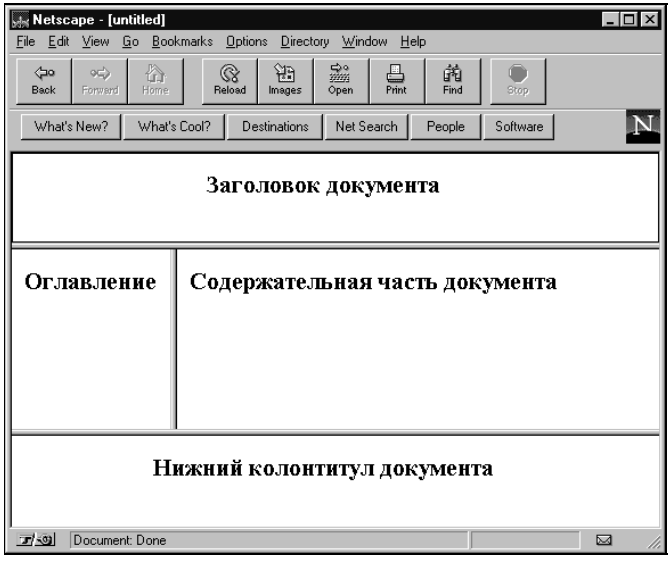


Рис. 5.7. Результат отображения браузером Netscape HTML-документа с фреймами, приведенного в примере

Тэг <FRAMESET>

Фреймы определяются в структуре, называемой FRAMESET, которая используется для страниц, содержащих фреймы, вместо раздела BODY обычного документа. Web-страницы, составленные из фреймов, не могут содержать раздел BODY в своем HTML-коде. В свою очередь, страницы с разделом BODY не могут использовать фреймы.

Совет

Так как для страниц с фреймами не применяется раздел BODY, то нет возможности задать фоновое изображение и цвет фона для всей страницы в целом. Напомним, что эти установки определяются параметрами BACKGROUND и BGCOLOR, записываемыми в тэге BODY. Однако это не мешает в каждый фрейм загружать документы, имеющие свои параметры фона.

Контейнер из тэгов <FRAMESET> и </FRAMESET> обрамляет каждый блок определений фрейма. Внутри контейнера <FRAMESET> могут содержаться только тэги <FRAME> и вложенные тэги <FRAMESET>.

Тэг <FRAMESET> имеет два параметра: ROWS (строки) и COLS (столбцы) и записывается в следующем виде:

```
<FRAMESET ROWS="список_значений" COLS="список_значений">.
```

Примечание

Некоторые браузеры разрешают использовать дополнительные параметры тэга <FRAMESET>. Обзор возможностей браузеров Netscape и Microsoft Internet Explorer дается в конце главы.

Можно определить значения для ROWS или COLS, или обоих вместе. Необходимо определить, по меньшей мере, два значения хотя бы одного из этих параметров. Если другой параметр опущен, то его значение принимается равным 100%.

Совет

Если в тэге <FRAMESET> определено только одно значение для ROWS и COLS, то этот тэг будет считаться ошибочным и браузер проигнорирует его. Другими словами, нельзя определить <FRAMESET>, состоящий только из одного фрейма.

Список значений параметров ROWS и COLS тэга <FRAMESET> представляет собой разделенный запятыми список значений, которые могут задаваться в пикселах, в процентах или в относительных единицах. Число строк или столбцов определяется числом значений в соответствующем списке. Например, запись

```
<FRAMESET ROWS="100,240,140">
```

определяет набор трех фреймов. Эти значения представляют собой абсолютные значения в пикселах. Другими словами, первый фрейм (первая строка) имеет высоту 100 пикселов, второй — 240 и последний — 140 пикселов.

Задание значений размеров фреймов в пикселах не очень удобно. Здесь не учитывается тот факт, что браузеры запускаются в различных операционных системах и с различными разрешениями дисплеев. В то же время, можно определить абсолютные значения размеров для некоторых случаев, например, для отображения небольшого изображения с известными размерами. Лучшим вариантом будет задание значений в процентах или в относительных единицах, например:

```
<FRAMESET ROWS="25%,50%,25%">.
```

В этом примере создаются три фрейма, размещаемые как строки во всю ширину экрана. Верхняя строка займет 25 процентов от доступной высоты экрана, средняя строка — 50 процентов и нижняя — 25 процентов. Если сумма заданных процентов не равна 100%, то значения будут пропорционально отмасштабированы, чтобы в итоге получилось ровно 100%.

Значения в относительных единицах выглядят следующим образом:

```
<FRAMESET COLS="*,2*,3*">.
```

Звездочка (*) используется для пропорционального деления пространства. Каждая звездочка представляет собой одну часть целого. Складывая все значения чисел, стоящих у звездочек (если число опущено, то подразумевается единица), получим знаменатель дроби. В этом примере первый столбец займет 1/6 часть от общей ширины окна, второй столбец — 2/6 (или 1/3), а последний — 3/6 (или 1/2).

Помните, что числовое значение без каких-либо символов определяет абсолютное число пикселов для строки или колонки. Значение со знаком процента (%) определяет долю от общей ширины (для `COLS`) или высоты (для `ROWS`) от окна просмотра, а значение со звездочкой (*) задает пропорциональное распределение оставшегося пространства.

Приведем пример, использующий все три варианта задания значений:

```
<FRAMESET COLS="100,25%,*,2*">.
```

В этом примере первый столбец будет иметь ширину 100 пикселов. Второй столбец займет 25 процентов от всей ширины окна просмотра, третий столбец — 1/3 оставшегося пространства и, наконец, последний столбец — 2/3. Абсолютные значения рекомендуется назначать первыми по порядку слева направо. За ними следуют процентные значения от общего размера пространства. В заключение записываются значения, определяющие пропорциональное разбиение оставшегося пространства.

Совет

Если вы используете абсолютные значения параметров COLS или ROWS, то задавайте их небольшими, чтобы они могли поместиться в любом окне браузера, и дополняйте их, по крайней мере, одним значением, заданным в процентной или относительной форме, для заполнения оставшегося пространства.

Если используется тэг <FRAMESET>, в котором заданы значения и COLS, и ROWS, то будет создана сетка из фреймов. Например:

```
<FRAMESET ROWS="*,2*,*" COLS="2*,*">
```

Эта строка HTML-кода создает сетку фреймов с тремя строками и двумя столбцами. Первая и последняя строки занимают 1/4 высоты каждая, а средняя строка — половину. Первый столбец занимает 2/3 ширины, а второй — 1/3.

Контейнер <FRAMESET> </FRAMESET> может быть вложен внутрь другого такого же контейнера, как это было показано в начальном примере. Рассмотрим далее использование тэга <FRAME>.

Примечание

В некоторых источниках по языку HTML указывается, что параметры COLS и ROWS тэга <FRAMESET> являются взаимоисключающими. Однако и Netscape, и Microsoft Internet Explorer допускают их совместное использование.

Тэг *<FRAME>*

Тэг <FRAME> определяет одиночный фрейм. Он должен располагаться внутри пары тэгов <FRAMESET> и </FRAMESET>. Например:

```
<FRAMESET ROWS="*,2*">
<FRAME>
<FRAME>
</FRAMESET>
```

Обратите внимание, что тэг <FRAME> не является контейнером и в отличие от <FRAMESET> не имеет завершающего тэга. Все определение одиночного фрейма выполняется одной строчкой HTML-кода.

Необходимо записать столько тэгов <FRAME>, сколько отдельных фреймов определено при задании тэга <FRAMESET>. В предыдущем примере тэгом <FRAMESET> задано две строки, поэтому потребовалось записать два тэга <FRAME>. Однако этот пример, по существу, бесполезен, так как ни один из фреймов не имеет какого-либо содержания!

Тэг <FRAME> имеет шесть параметров: SRC, NAME, MARGINWIDTH, MARGINHEIGHT, SCROLLING и NORESIZE.

Примечание

Некоторые браузеры разрешают использовать ряд дополнительных параметров тэга <FRAME>. Обзор возможностей браузеров Netscape и Microsoft Internet Explorer дается в конце главы.

Приведем запись тэга <FRAME> со всеми параметрами:

```
<FRAME SRC="url" NAME="window_name" SCROLLING=YES|NO|AUTO
MARGINWIDTH="value" MARGINHEIGHT="value" NORESIZE>
```

На практике в тэге <FRAME> редко используются одновременно все параметры.

Наиболее важный параметр — SRC (сокращение от слова source). Довольно часто в тэге <FRAME> задается единственный параметр SRC. Например:

```
<FRAME SRC="url">.
```

Значение параметра SRC определяет URL-адрес документа, который будет загружен изначально в данный фрейм. Обычно в качестве такого адреса записывается имя HTML-файла, расположенного в том же самом каталоге, что и основной документ. Тогда строка определения фрейма будет выглядеть, например, так:

```
<FRAME SRC="sample.htm">.
```

Обратите внимание, что любой HTML-файл, заданный в описании фрейма, должен быть полным HTML-документом, а не фрагментом. Это означает, что документ должен иметь тэги HTML, HEAD, BODY и т. д.

Конечно, в качестве значения SRC может быть задан любой допустимый URL-адрес. Если, например, фрейм используется для отображения изображения в формате GIF, которое располагается на сервере издательства данной книги, то следует записать:

```
<FRAME SRC="http://www.bhv.ru/example.gif">.
```

Совет

Не задавайте в документе, описывающем структуру фреймов, никакого содержания.

Обычный текст, заголовки, графические изображения и другие элементы не могут прямо использоваться в документе, который описывает структуру фреймов. Все содержание фреймов должно быть определено в отдельных HTML-файлах, имена которых задаются параметром SRC тэга <FRAME>.

Параметр NAME определяет имя фрейма, которое может использоваться для ссылки к данному фрейму. Обычно ссылка задается из другого фрейма, располагающегося на той же самой странице. Например:

```
<FRAME SRC="sample.htm" NAME="Frame_1">.
```

Такая запись создает фрейм с именем "Frame_1", на который может быть выполнена ссылка. Например:

```
<A HREF="other.htm" TARGET="Frame_1">Щелкните здесь для загрузки
документа other.htm во фрейм с именем Frame_1</A>.
```

Обратите внимание на параметр TARGET, который ссылается на имя фрейма. Если для фрейма не задано имя, то будет создан фрейм без имени, и не будет возможности использовать ссылки на него из другого фрейма. Имена фреймов должны начинаться с алфавитно-цифрового символа.

Параметры MARGINWIDTH и MARGINHEIGHT дают возможность устанавливать ширину полей фрейма. Записывается это следующим образом:

```
MARGINWIDTH="value",
```

где "value" — абсолютное значение в пикселах. Например:

```
<FRAME MARGINHEIGHT= "5" MARGINWIDTH= "7">.
```

Данный фрейм имеет поля сверху и снизу по 5 пикселов, а слева и справа — по 7 пикселов. Не забудьте, что здесь идет речь о полях, а не о рамках. Параметры MARGINWIDTH и MARGINHEIGHT определяют пространство внутри фрейма, в пределах которого не будет располагаться никакая информация. Минимально допустимое значение этих параметров равно единице.

Для фреймов будут автоматически создаваться и отображаться полосы прокрутки, если содержимое фрейма не помещается полностью в отведенном пространстве. Иногда это нарушает дизайн страницы, поэтому было бы удобно иметь возможность управлять отображением полос прокрутки. Для этих целей используется параметр SCROLLING. Формат записи:

```
<FRAME SCROLLING="YES|NO|AUTO">.
```

Параметр SCROLLING может принимать три значения: YES, NO или AUTO. Значение AUTO действует так же, как и в случае отсутствия параметра SCROLLING. Значение YES вызывает появление полос прокрутки вне зависимости от необходимости этого, а NO — запрещает их появление. Например:

```
<FRAME SCROLLING=YES>.
```

Обычно пользователь может изменять размер фреймов при просмотре страницы. Если установить курсор мыши на рамке фрейма, то курсор примет форму, указывающую на возможность изменения размеров, и позволит выполнить перемещение рамки в нужное место. Это иногда нарушает структуру красиво спроектированных фреймов. Для предотвращения возможности изменения пользователем размера фреймов следует воспользоваться параметром NORESIZE:

```
<FRAME NORESIZE>.
```

Этот параметр не требует никаких значений. Естественно, когда задан параметр NORESIZE для одного из фреймов, то размер любого из смежных фрей-

мов также не может быть изменен. Иногда, в зависимости от расположения фреймов, использования параметра NORESIZE в одном из фреймов будет достаточно, чтобы предотвратить возможность изменения размеров любого из них на экране.

Тэг <NOFRAMES>

Возможность работы с фреймами не предполагалась ни в стандарте HTML 3.0, ни в HTML 3.2. Здесь до последнего времени складывалась достаточно типичная ситуация, когда реально используемые возможности активно применяются на многих WWW-страницах, однако не являются частью стандарта. Это означало, что браузеры вполне законно могли игнорировать фреймы. С появлением стандарта HTML 4.0 ситуация изменилась — теперь поддержка фреймовых структур закреплена стандартом. Заметим, что большинство современных браузеров распознавали фреймы и до появления HTML 4.0. Тем не менее, необходимо предоставлять информацию пользователям, применяющим браузеры без поддержки фреймов. Для таких браузеров можно предусмотреть альтернативную информацию, которая записывается между парой тэгов <NOFRAMES> и </NOFRAMES>. Это выглядит следующим образом:

```
<NOFRAMES>
весь HTML-документ
</NOFRAMES>
```

Все, что размещено между тэгами <NOFRAMES> и </NOFRAMES>, будет отображаться браузерами, не имеющими возможностей поддержки фреймов. Браузеры с поддержкой фреймов проигнорируют всю информацию между этими тэгами.

Заметим, что в реальной жизни разработчики HTML-страниц часто не используют возможности тэга <NOFRAMES> для создания страниц без фреймовых структур, а просто создают две версии своих HTML-документов. Для такого варианта на стартовой странице обычно предлагается выбор загрузки документа с фреймовой структурой или без нее. Далее в зависимости от выбора пользователя загружается только один вариант документа.

Особенности описания фреймовых структур

Одним из важнейших тэгов, применяемых при описании фреймовых структур, является тэг <FRAME>. Тэг имеет ряд параметров, ни один из которых не является обязательным и не зависит от других, однако при их записи следует учитывать ряд моментов.

Оказывается, что при необходимости создания фрейма, в который в дальнейшем может быть загружен какой-либо документ, например, по команде из другого фрейма, следует в тэге <FRAME> записать параметр SRC. Если этот

параметр опущен, то фрейм не будет создан, хотя место под него будет оставлено. Например, запись типа `<FRAME NAME="B">` вполне логична и могла бы определять фрейм с именем "B", в который исходно не загружается никакой документ. Однако из-за отсутствия параметра `SRC` фрейм с таким именем не будет существовать, поэтому дальнейшие попытки загрузить в него какой-либо документ останутся безрезультатными, а место в окне, отведенное под данный фрейм, будет пустовать. Более того, некоторые браузеры (например, Microsoft Internet Explorer версии 3 для Windows 3.xx) при попытке загрузки документа в такой фрейм выдадут сообщение об ошибке и завершат работу.

Обязательность задания параметра `SRC` не поддается логическому объяснению, поэтому лучше всего просто принять к сведению этот факт. Тогда даже при отсутствии документа, который необходимо загружать в данный фрейм с самого начала, следует в параметре `SRC` задать имя какого-либо файла. Например, такой файл можно назвать empty.htm (empty — пустой), содержимым которого будет являться минимально возможный корректный HTML-документ, а именно:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Можно сократить данный документ до двух тэгов: `<HTML></HTML>`, что также будет верным HTML-документом. Идя по пути максимального сокращения размеров "пустого" документа, можно ограничиться файлом, размер которого равен одному байту, в котором хранится символ пробела (или любой другой неотображаемый символ). Этот файл не будет корректным HTML-документом, но не вызовет нареканий со стороны большинства браузеров. Дальнейшее сокращение размера такого файла до нуля не оправдано, так как при его загрузке браузером Netscape будет выдаваться предупреждающее сообщение (рис. 5.8) о том, что документ не содержит данных.

Рис. 5.8. Предупреждающее сообщение при загрузке файла нулевой длины

При этом на данное сообщение необходимо отреагировать, нажав клавишу <Enter> или кнопку мыши. При любой перезагрузке документа или изменении размеров окна просмотра браузера сообщение будет возникать вновь.

Можно также задать имя несуществующего файла, однако при этом браузер Netscape будет выдавать предупреждающее сообщение (рис. 5.9), что не помешает дальнейшей работе, но приведет к аналогичным неудобствам.

Рис. 5.9. Предупреждающее сообщение при попытке загрузки несуществующего файла

Совет

Создайте файл с именем empty.htm, размер которого равен одному байту, содержащий символ пробела. Примите за правило при записи тэга `<FRAME>` всегда указывать `SRC=empty.htm`, если сразу невозможно указать имя конкретного файла.

Примеры фреймов

В этом разделе представлены некоторые типичные примеры определений фреймов.

Возвратимся к примеру, который приведен в начале данного раздела (рис. 5.7). Этот пример использует вложенную структуру `<FRAMESET>`. Внешний тэг `<FRAMESET>` создает три строки высотой, соответственно, в 25, 50 и 25 процентов от общей высоты окна просмотра:

```
<FRAMESET ROWS="25%,50%,25%">.
```

Внутри этой области определения первая и последняя строки представляют собой простые фреймы:

```
<FRAME SRC="header.htm">
<FRAME SRC="footer.htm">
```

Каждая из этих строк заполняет всю ширину экрана. Первая строка в верхней части экрана занимает 25 процентов высоты, и третья строка в нижней части также занимает 25 процентов высоты.

Между ними, однако, располагается вложенный тэг `<FRAMESET>`:

```
<FRAMESET COLS="25%,75%">
<FRAME SRC="list.htm">
<FRAME SRC="info.htm">
</FRAMESET>
```

Этот тэг определяет два столбца, на которые разбивается средняя строка экрана. Строка, в которой располагаются эти два столбца, занимает 50 про-

центов высоты экрана, что определено во внешнем тэге <FRAMESET>. Левый столбец использует 25 процентов от ширины экрана, в то время как правый столбец занимает оставшиеся 75 процентов ширины.

Фреймы для этих столбцов определены внутри вложенной пары тэгов <FRAMESET> и </FRAMESET>, в то время как определение фреймов для первой и последней строки записывается вне этой пары, но внутри внешнего <FRAMESET> в соответствующем порядке.

Структуру записи легко понять, если воспринимать вложенный блок <FRAMESET> как отдельный элемент <FRAME>. В нашем примере внешний тэг <FRAMESET> определяет три строки. Каждая из них должна быть заполнена. В данном случае они заполняются сначала отдельным элементом <FRAME>, далее — вложенным блоком <FRAMESET> шириной в два столбца, а затем еще одним элементом <FRAME>.

Теперь может возникнуть вопрос, можно ли в качестве значения параметра SRC тэга <FRAME> задать имя файла, который, в свою очередь, содержит описание структуры фреймов. Да, это допустимо. В данном случае тэг <FRAME> будет использован для указания на HTML-документ, который является фреймовой структурой и используется в качестве отдельного фрейма.

Вернемся к примеру и заменим вложенный <FRAMESET> на отдельный <FRAME>. Естественно, потребуется два HTML-файла вместо одного, так как вложенный <FRAMESET> теперь будет располагаться в отдельном документе. Приведем содержимое первого (внешнего) файла:

```
<HTML>
<HEAD>
</HEAD>
<FRAMESET ROWS="25%,50%,25%">
<FRAME SRC="header.htm">
<FRAME SRC="frameset.htm">
<FRAME SRC="footer.htm">
</FRAMESET>
<NOFRAMES>
Ваш браузер не может отображать фреймы
</NOFRAMES>
</HTML>
```

Второй файл с именем frameset.htm содержит следующий код:

```
<HTML>
<HEAD>
</HEAD>
<FRAMESET COLS="25%,75%">
<FRAME SRC="list.htm">
<FRAME SRC="info.htm">
```

```
</FRAMESET>
</HTML>
```

В этом случае верхняя и нижняя строки ведут себя по-прежнему. Но вторая строка теперь является простым фреймом, как и другие. Однако файл frameset.htm, на который указывает параметр SRC, определяет собственную структуру фреймов. В результате на экране будет отображено в точности то же самое, что и в первоначальном примере.

Примечание

Принципиально возможно создать вложенные структуры <FRAMESET>, использующие тэги <FRAME>, которые ссылаются на тот же самый файл с описанием структуры фреймов, однако этого делать не следует. Такая ситуация приведет к бесконечной рекурсии и не даст возможности дальнейшей работы. Некоторые браузеры контролируют подобную ситуацию и предотвращают возможность сбоя. Если адрес, записанный в SRC, совпадает с одним из предыдущих адресов в иерархии фреймов, то он игнорируется, как если бы параметр SRC отсутствовал вообще.

Совет

Используя вложенные структуры <FRAMESET> в различных комбинациях, возможно создать практически любую сетку фреймов, которую можно себе представить. Однако помните, что следует создавать удобный для пользователя интерфейс, а не просто демонстрировать свое умение работать с фреймами.

Приведем пример создания регулярной прямоугольной сетки фреймов:

```
<HTML>
<HEAD>
</HEAD>
<FRAMESET ROWS="*,2*" COLS="20%,30%,40%">
<FRAME SRC="docum1.htm">
<FRAME SRC="docum2.htm">
<FRAME SRC="docum3.htm">
<FRAME SRC="docum4.htm">
<FRAME SRC="docum5.htm">
<FRAME SRC="docum6.htm">
</FRAMESET>
</HTML>
```

Этот пример создает сетку фреймов с двумя строками и тремя столбцами (рис. 5.10). Так как определен набор из шести фреймов, необходимо также дать шесть определений отдельных фреймов <FRAME>. Обратите внимание, что определения фреймов даются построчно. То есть первый тэг <FRAME> определяет содержимое первой колонки в первой строке, второй — второй колонки, а третий заканчивает определение данных для последней колонки первой строки. Последние три фрейма затем заполняют столбцы второй строки.

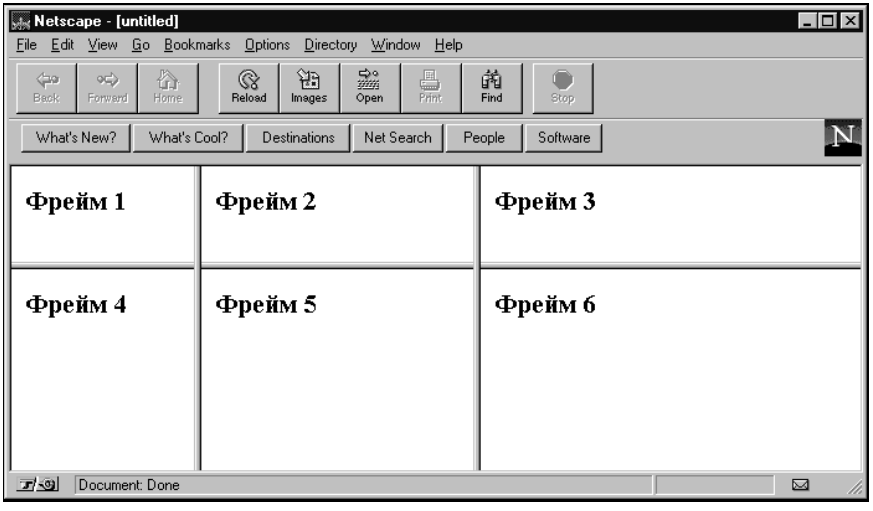


Рис. 5.10. Сетка фреймов 2 на 3

Заметим также, что сумма значений процентов в параметре `COLS` равна не 100, а только 90 процентов. В этом нет ничего страшного, так как браузер автоматически пропорционально изменит ширину колонок, чтобы ликвидировать это противоречие.

Особенности навигации при использовании фреймов

Работа с документами, имеющими фреймовую структуру, имеет некоторые особенности, которые необходимо знать. Эти особенности в основном проявляются в навигации при загрузке документов. Значительные различия в навигации свойственны не только разным браузерам, но и разным версиям одного и того же браузера.

Браузер Netscape версий 3.x и 4.x при нажатии кнопки **Back** возвращает обратно документ в тот фрейм, действия с которым производились последним. Те же действия будут произведены в случае, если будет выбран пункт **Back** при вызове контекстного меню в любом из фреймов. Напомним, что контекстное меню вызывается нажатием правой кнопки мыши. Таким образом, независимо от того, в каком из фреймов было вызвано контекстное меню, при нажатии кнопки **Back** будет выполнена отмена последней операции, даже если она была произведена в другом фрейме.

Браузер Netscape версии 2.x работал совершенно по-другому. Контекстное меню содержит команду **Back in Frame**, выполнение которой возвращает документ в текущий фрейм, а не выполняет отмену последней операции.

В любой версии Netscape можно сделать закладку на документ, содержащийся в выбранном фрейме. Для этого необходимо выбрать режим **Add Bookmark** из контекстного меню, о котором говорилось выше. Если же просто будет выбран режим **Add Bookmark** из главного меню браузера, то будет сделана закладка на документ с описанием структуры фреймов `<FRAMESET>`, которая не будет точно указывать на конкретный фрейм. Возможность создания закладки на документ отдельного фрейма вовсе не означает, что при дальнейшем использовании этой закладки возникнет та же самая структура фреймов. Документ, на который указывает закладка, будет загружен в полное окно вне фреймовой структуры.

Взаимодействие между фреймами

Простейшая форма просмотра информации на WWW состоит в чтении страниц и переходах по ссылкам, при которых текущий документ в окне браузера замещается другим документом. При работе с фреймами можно организовать более удобную для пользователя схему загрузки документов.

Взаимодействие между фреймами заключается в возможности загрузки документов в выбранный фрейм по командам из другого фрейма. Для этой цели используется параметр `TARGET` тэга `<A>`. Данный параметр определяет имя фрейма или окна браузера, в которое будет загружаться документ, на который указывает данная ссылка. По умолчанию при отсутствии параметра `TARGET` документ загружается в текущий фрейм (или окно). Это умолчание может быть изменено заданием тэга `<BASE>` с нужным значением параметра `TARGET`. Задание имени фрейма, в который осуществляется загрузка по умолчанию, очень удобно для тех случаев, когда большое количество ссылок должно направлять документы в определенный фрейм. Типичная ситуация с оглавлением в одном фрейме, ссылки из которого загружают соответствующие документы в смежный фрейм, была показана в начале данной главы (рис. 5.1). Для этого примера в разделе `<HEAD>` файла с именем LIST.htm целесообразно записать следующую строчку: `<BASE TARGET="pages">`. В противном случае для каждой ссылки пришлось бы указывать параметр `TARGET`.

Имена фреймов должны начинаться с латинской буквы или цифры. В качестве имени может задаваться имя существующего окна или фрейма, а может указываться новое имя, под которым будет открыто новое окно. Имеется четыре зарезервированных имени, при задании которых выполняются специальные действия. Эти имена начинаются с символа подчеркивания (_): `"_blank"`, `"_self"`, `"_parent"` и `"_top"`. Любое другое имя, начинающееся с символа "подчеркивание", недопустимо.

`TARGET="_blank"` — обеспечивает загрузку документа в новое окно. Это окно не будет иметь имени, а следовательно, в него невозможно будет загрузить другой документ.

TARGET="_self" — загрузка документа будет произведена в текущий фрейм (или окно). Такую запись следует использовать для обхода умолчания, заданного тэгом <BASE>.

TARGET="_top" — вызывает загрузку документа в полное окно. Если документ уже располагается в полном окне, то данное значение действует так же, как "_self".

TARGET="_parent" — вызывает загрузку документа в область, занимаемую фреймом-родителем текущего фрейма. При отсутствии фрейма-родителя данное значение параметра действует так же, как "_top".

Примечание

В ряде источников по языку HTML ошибочно утверждается, что при отсутствии родителя у фрейма значение "_parent" эквивалентно "_self". Такое утверждение не всегда корректно.

Предупреждение

Зарезервированные имена фреймов "_blank", "_self", "_parent" и "_top" должны записываться строчными латинскими буквами. Заметим, что такая требовательность присуща только Netscape. Браузер Microsoft Internet Explorer правильно распознает зарезервированные имена, записанные на любом регистре.

Приведем примеры взаимодействия между фреймами и отдельными окнами браузера. Рассмотрим следующий HTML-код:

```
<HTML>
<HEAD>
<TITLE>Использование фреймов</TITLE>
</HEAD>
<FRAMESET COLS=2*,*,*>
<FRAME SRC=frame_a.htm NAME="A">
<FRAME SRC=empty.htm NAME="B">
<FRAME SRC=empty.htm NAME="C">
</FRAMESET>
</HTML>
```

В этом HTML-документе дается описание структуры, состоящей из трех фреймов с именами "A", "B" и "C". Имена фреймов потребуются в дальнейшем для организации ссылок между фреймами. Заметим, что на фрейм с именем "A" в данном примере ссылок не будет, поэтому он мог быть оставлен без имени вообще. При загрузке приведенного выше документа в браузер во фреймах будет отображена информация, содержащаяся в файлах, определяемых параметром SRC. Во фрейм "A" попадет содержимое файла frame_a.htm, а остальные два фрейма получат данные из файла empty.htm, который не имеет отображаемых данных. Еще раз напомним, что HTML-документ, описывающий структуру фреймов, не имеет раздела <BODY>.

Приведем текст файла с именем frame_a.htm:

```
<HTML>
<HEAD>
<TITLE>Документ для фрейма A</TITLE>
</HEAD>
<BODY>
<A HREF="test.htm" TARGET="B">1. Загрузка документа во фрейм B</A><P>
<A HREF="test.htm" TARGET="C">2. Загрузка документа во фрейм C</A><P>
<A HREF="test.htm" TARGET="D">3. Загрузка документа в окно с именем
D</A><P>
<A HREF="test.htm" TARGET="_blank">4. Загрузка документа в новое ок-
но</A><P>
<A HREF="test.htm" TARGET="_top">5. Загрузка документа в полное ок-
но</A><P>
<A HREF="test.htm" TARGET="_self">6. Загрузка документа в текущий
фрейм</A>
</BODY>
</HTML>
```

Этот документ является полным HTML-документом, имеющий разделы `<HEAD>` и `<BODY>` и, в свою очередь, имеет ссылки на файл с именем test.htm, располагающийся в том же самом каталоге, что и файл frame_a.htm.

Текст файла test.htm крайне прост:

```
<HTML>
<HEAD>
<TITLE>Тестовый документ</TITLE>
</HEAD>
<BODY>
Текст тестового документа
</BODY>
</HTML>
```

Файл frame_a.htm, содержимое которого загрузилось во фрейм "A", имеет шесть ссылок на один и тот же файл test.htm с различным значением параметра `TARGET`.

Рассмотрим действия, которые будут происходить при реализации этих ссылок. Первая ссылка со значением `TARGET="B"` будет загружать файл test.htm во фрейм с именем "B". Заметим, что после реализации любой из шести ссылок браузер Netscape автоматически окрасит в другой цвет все шесть, так как они указывают на один и тот же файл. Microsoft Internet Explorer отмечает только действительно реализованные ссылки.

Вторая ссылка выполнит те же действия для фрейма "C". Изначально во фреймах "B" и "C" ничего нет (точнее загружено содержимое пустого файла empty.htm). Реализация первой и второй ссылок заполнит эти фреймы.

Третья ссылка со значением `TARGET="D"` приведет к образованию нового окна браузера с именем `"D"` и загрузке в него файла test.htm. Заметим, что форма записи этой ссылки ничем не отличается от первых двух. Различие состоит в том, что в первых двух случаях ссылки были даны на существующие фреймы, имена которых были определены в файле со структурой фреймов, а в данном случае ссылка дана на несуществующий объект. Если данная ссылка будет выполнена хотя бы один раз, то окно с именем `"D"` будет образовано и повторный переход по ссылке лишь перезагрузит данные в существующее теперь окно `"D"`. Конечно, пользователь может в любой момент его закрыть и вновь образовать выбором данной ссылки. На рис. 5.11 показана ситуация после того, как первые три ссылки были реализованы. Напомним, что расположение и размеры окон на экране определяются пользователем.

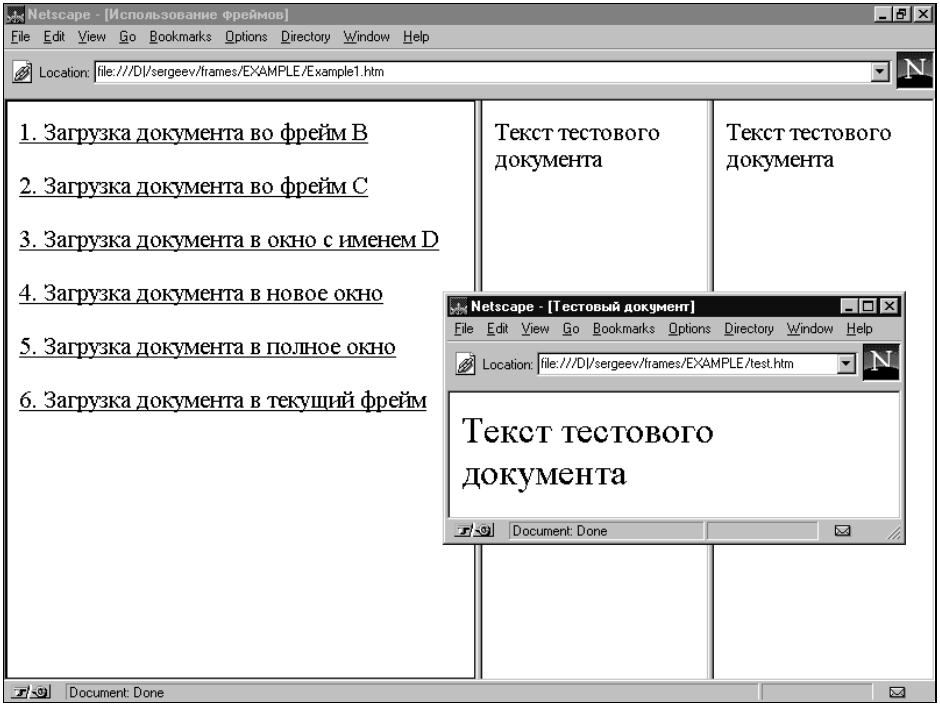


Рис. 5.11. Ситуация, полученная после последовательной реализации первых трех ссылок, имеющихся в левом фрейме

Четвертая ссылка со значением `TARGET="_blank"` создаст новое окно без имени и загрузит туда требуемый документ. Любое повторение данной ссылки будет открывать еще одно окно браузера.

Пятая ссылка со значением `TARGET="_top"` загрузит документ в полное окно вместо всей фреймовой структуры. При таком значении параметра `TARGET`

новое окно не образуется. Возврат к фреймовой структуре возможен нажатием кнопки **Back**.

Последняя ссылка со значением `TARGET="_self"` загрузит документ во фрейм `"A"` на место документа со ссылками. В данном случае результат эквивалентен выполнению ссылки без параметра `TARGET`.

Примечание

Имена фреймов или окон браузера не следует путать с названиями загружаемых документов. Имена фреймов при просмотре нигде не видны, они требуются только для организации взаимодействия и поэтому скрыты от пользователя. Увидеть их можно только при просмотре исходного текста HTML-файлов.

Совет

Напомним, что названия загружаемых документов задаются тэгом `<TITLE>`. Если документ загружен в полное окно, то его название выдается в самой верхней части окна браузера. Если же документ загружается во фрейм, то его название нигде не отображается, а в верхней части окна по-прежнему будет располагаться название документа, содержащего описание фреймовой структуры документа. Поэтому названия документов, предназначенных для просмотра во фреймах, не имеют большого значения. Например, на рис. 5.11 один и тот же документ загружен во фреймы `"B"` и `"C"`, а также в отдельное окно с именем `"D"`, при этом название документа видно только в окне `"D"`. Тем не менее, вряд ли стоит рекомендовать опускать названия документов, загружаемых во фреймы, так они могут появиться, например, в списке закладок (**Bookmarks**) при создании закладки на документ, расположенный во фрейме или списке просмотренных документов.

Рассмотрим еще один интересный пример организации взаимодействия между фреймами и окнами браузера. Пусть имеется текст основного загружаемого HTML-документа:

```
<HTML>
<HEAD>
<TITLE>Использование имен окон</TITLE>
</HEAD>
<BODY>
<A HREF="test.htm" TARGET="D">Загрузка документа в окно с именем D</A><P>
<A HREF="frame.htm" TARGET="_blank">Загрузка документа, имеющего
фреймовую структуру, в новое окно</A><P>
</BODY>
</HTML>
```

В данном документе имеются ссылки на файлы test.htm и frame.htm. Пусть первый из них содержит ту же информацию, что и в предыдущем примере. Текст файла frame.htm имеет следующий вид:

```
<HTML>
<HEAD>
<TITLE>Использование фреймов</TITLE>
</HEAD>
<FRAMESET COLS=*,*>
<FRAME SRC=empty.htm NAME="C">
<FRAME SRC=empty.htm NAME="D">
</FRAMESET>
</HTML>
```

Обратите внимание, что если основной документ является стандартным HTML-документом, то загружаемый по ссылке из основного документа файл frame.htm содержит структуру фреймов и в свою очередь ссылается на файл empty.htm.

После загрузки основного документа окно браузера будет иметь вид, показанный на рис. 5.12 (левое окно). Весь документ состоит из двух ссылок. Выполним переход по первой ссылке. Будет образовано новое окно с именем "D", в котором появится текст файла test.htm (рис. 5.12, правое окно). Повторение этой ссылки будет лишь перезагружать данные в окне "D".

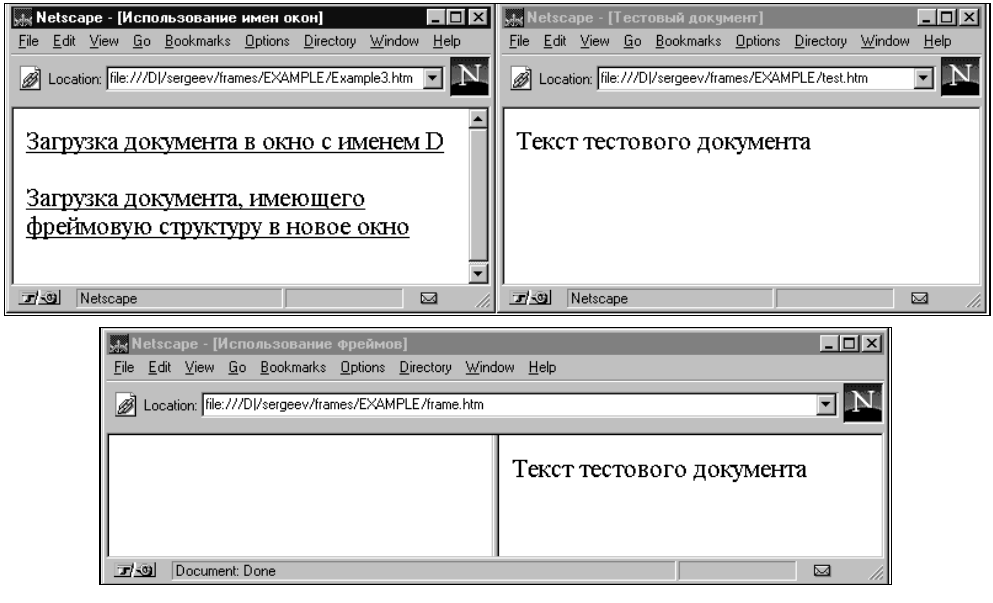


Рис. 5.12. Пример окон взаимодействующих фреймов

Выполним переход по второй ссылке. Образуется новое окно без имени, в которое загрузится файл frame.htm, определяющий два фрейма с именами "C" и "D" (рис. 5.12, нижнее окно). В обоих фреймах ничего нет (точнее загружен пустой документ empty.htm). Обратим внимание, что теперь имеется

открытое окно с именем "D" и окно с фреймами, один из которых также имеет имя "D". Выполним снова переход по первой ссылке. В отличие от первого случая загрузка данных будет осуществляться не в окно "D", а во фрейм с именем "D". Результат всех описанных действий показан на рис. 5.12.

Примечание

Появление открывающихся окон и их содержимое может иногда зависеть даже от порядка действий пользователя. Работа с документами, поведение которых трудно предсказуемо, обычно вызывает справедливое раздражение пользователя и говорит о недостаточной продуманности структуры данных разработчиками.

Если же изменить порядок действий, т. е. сначала выполнить вторую ссылку, а затем первую, то окно с именем "D" вообще не появится! Это произойдет потому, что после реализации второй ссылки образуется фрейм с именем "D" и для первой ссылки не будет нужды открывать новое окно.

Этот пример вовсе не является образцом для подражания, а лишь показывает возможную сложность организации взаимодействия. Напротив, нужно стараться без особой нужды не усложнять организацию данных, тем более не создавать ситуаций, в которых результат меняется в зависимости от порядка действий пользователя.

Совет

Избегайте коллизий в именах фреймов и окон. Хотя формально не запрещено иметь фреймы с одинаковыми именами, однако это может привести к путанице.

Предупреждение

Имена фреймов и окон сравниваются с учетом регистра символов. Так, например, фреймы с именами "frame_1" и "Frame_1" будут различны.

Примеры более сложного взаимодействия между фреймами

Выше были рассмотрены достаточно простые типовые примеры взаимодействия между фреймами. Были рассмотрены задачи создания новых окон, замены содержимого отдельных фреймов, а также выдачи документа в полное окно с разрушением всей структуры фреймов. Даны примеры использования задаваемых имен фреймов, а также зарезервированных имен "_blank", "_self" и "_top". Использование последнего зарезервированного имени "_parent" более сложно и будет описано ниже.

В данном разделе будут рассмотрены более сложные варианты взаимодействия между фреймами. В частности, будет реализована замена содержимого нескольких смежных фреймов.

Одним из наиболее часто встречающихся вариантов применения фреймов, который уже упоминался в данной главе, является случай двух фреймов, один из которых содержит список ссылок, а в другой загружаются сами документы (рис. 5.1).

Попробуем расширить постановку задачи. Пусть необходимо отображать на экране содержимое достаточно большого документа, состоящего из глав, разделенных на разделы. Типичным примером служит техническая литература по какой-либо тематике. Опишем желаемое представление такого документа на экране. Разобьем экран на три фрейма, в одном из которых будет располагаться список глав книги, во втором — перечень разделов выбранной главы, а в третьем — текст выбранного раздела. При выборе ссылки во втором фрейме должно меняться содержимое третьего фрейма. Реализация этого требования тривиальна. При выборе ссылки в первом фрейме должно одновременно изменяться содержимое как второго, так и третьего фрейма. На первый взгляд реализация этой задачи на языке HTML невозможна (без применения программирования на языке JavaScript или др.), так как при выполнении ссылки загружается только один документ, а не два или более. Тем не менее, решение данной задачи вполне возможно.

Покажем возможную схему решения такой задачи на простом примере. Пусть требуется отобразить на экране три фрейма и загрузить в них некоторые документы. Поставим задачу создать в каждом из этих фреймов ссылки, реализация которых, например, меняла местами содержимое двух фреймов. Пусть первый фрейм занимает 50% ширины окна и 100% высоты и располагается с левой стороны окна. Правая половина окна делится по горизонтали также пополам и содержит два других фрейма. Такая структура описывается следующим кодом:

```
<HTML>
<HEAD>
<TITLE>Пример взаимодействия между фреймами</TITLE>
</HEAD>
<FRAMESET COLS="*,*">
<FRAME SRC="left.htm">
       <FRAMESET ROWS="*,*">
       <FRAME SRC="1.htm">
       <FRAME SRC="2.htm">
       </FRAMESET>
</FRAMESET>
</HTML>
```

С помощью данного HTML-кода будет создана требуемая структура, однако решение поставленной задачи невозможно. Необходимо вынести вложенную структуру `<FRAMESET>` в отдельный файл, а в данном HTML-коде описать фрейм, ссылающийся на созданный файл. Тогда текст исходного документа будет иметь вид:

```
<HTML>
<HEAD>
<TITLE>Пример взаимодействия между фреймами</TITLE>
</HEAD>
<FRAMESET COLS="*,*">
<FRAME SRC="left.htm">
<FRAME SRC="1_2.htm" NAME="Two_Frames">
</FRAMESET>
</HTML>
```

Созданный файл с вложенной структурой `<FRAMESET>` имеет имя 1_2.htm и содержит следующий код:

```
<HTML>
<HEAD>
<TITLE>1-2</TITLE>
</HEAD>
<FRAMESET ROWS="*,*">
<FRAME SRC="1.htm">
<FRAME SRC="2.htm">
</FRAMESET>
</HTML>
```

На первый взгляд совершенно ничего не изменилось. В обоих случаях имеется три фрейма, в которые загружаются документы left.htm, 1.htm и 2.htm соответственно. Однако при взаимодействии фреймов различие проявится. Если в первом случае ни у одного из фреймов нет фрейма-родителя, то во втором случае для двух фреймов родительским будет фрейм с именем "Two_Frames". Поэтому если в любом из двух фреймов применить ссылку со значением параметра TARGET, равным "_parent", то результат будет различным для первого и второго случая. Для первого случая реализация такой ссылки приведет к загрузке документа в полное окно с замещением существующей структуры фреймов. Здесь проявляется свойство значения "_parent", которое при отсутствии фрейма-родителя действует как "_top". Во втором случае будет замещен фрейм с именем "Two_Frames", который занимает правую половину экрана и по существу состоит из двух фреймов.

Второй случай формально отличается от первого также наличием фрейма с именем "Two_Frames", к которому могут быть обращены ссылки. Как раз эта особенность и позволит нам решить поставленную задачу.

Приведем содержимое файла left.htm, который изначально загружается в первый из рассматриваемых фреймов:

```
<HTML>
<HEAD>
<TITLE>Левый фрейм</TITLE>
```

```
</HEAD>
<BODY>
Реализация любой ссылки во всех трех фреймах приводит к перезагрузке
документов в двух фреймах, расположенных в правой части окна.
<P>
Выберите вариант расположения документов:
<P>
<A HREF="1_2.htm" TARGET="Two_Frames">Вариант 1-2</A>
<P>
<A HREF="2_1.htm" TARGET="Two_Frames">Вариант 2-1</A>
</BODY>
</HTML>
```

В этом документе имеются ссылки на файлы 1_2.htm и 2_1.htm. Текст первого был дан выше, а второго имеет следующий вид:

```
<HTML>
<HEAD>
<TITLE>2-1</TITLE>
</HEAD>
<FRAMESET ROWS="*,*">
<FRAME SRC="2.htm">
<FRAME SRC="1.htm">
</FRAMESET>
</HTML>
```

Заметим, что текст файлов 1_2.htm и 2_1.htm отличаются только порядком ссылок на файлы 1.htm и 2.htm.

Рассмотрим теперь построение документа, загруженного в левый фрейм. В нем имеется две ссылки с параметром `TARGET="Two_Frames"`. Реализация любой из этих ссылок создает на месте расположения фрейма `"Two_Frames"` (это правая половина экрана) два фрейма с загрузкой документов 1.htm и 2.htm в том или ином порядке. Таким образом при выборе варианта 1-2 в верхний правый фрейм загружается документ 1.htm, а в нижний правый — 2.htm. При выборе варианта 2-1 порядок документов меняется. В итоге поочередный выбор вариантов создает впечатление того, что документы в двух фреймах меняются местами. Именно такого эффекта мы и стремились достичь (рис. 5.13).

Содержимое документов 1.htm и 2.htm для описанного примера не имеет значения. Тем не менее, для примера, вместо тривиальных документов создадим документы со ссылками, реализующими те же действия.

Текст файла 1.htm:

```
<HTML>
<HEAD>
```

```
<TITLE>Документ 1</TITLE>
</HEAD>
<BODY>
<H2>Документ 1</H2>
<A HREF="1_2.htm" TARGET="_parent">Вариант 1-2</A>
<P>
<A HREF="2_1.htm" TARGET="_parent">Вариант 2-1</A>
</BODY>
</HTML>
```

Файл 2.htm отличается от 1.htm только заголовком.

Здесь имеются две ссылки со значением `TARGET="_parent"`, которые обращены к родительскому фрейму. Эти ссылки могли бы быть записаны и с явным указанием имени фрейма-родителя, т. е. `TARGET="Two_Frames"`, однако использование неявного указания имени обычно более удобно. Например, если из левого фрейма (документ left.htm) исключить ссылки, то можно было бы опустить имя фрейма `"Two_Frames"`, заданное при описании основной фреймовой структуры. При этом был бы создан фрейм без имени, но ссылки из документов 1.htm и 2.htm со значением `TARGET="_parent"` по-прежнему работали бы правильно.

Совет

По возможности используйте неявное указание имен фреймов. Например, `"_parent"`, `"_top"`, `"_self"` вместо задания конкретных имен.

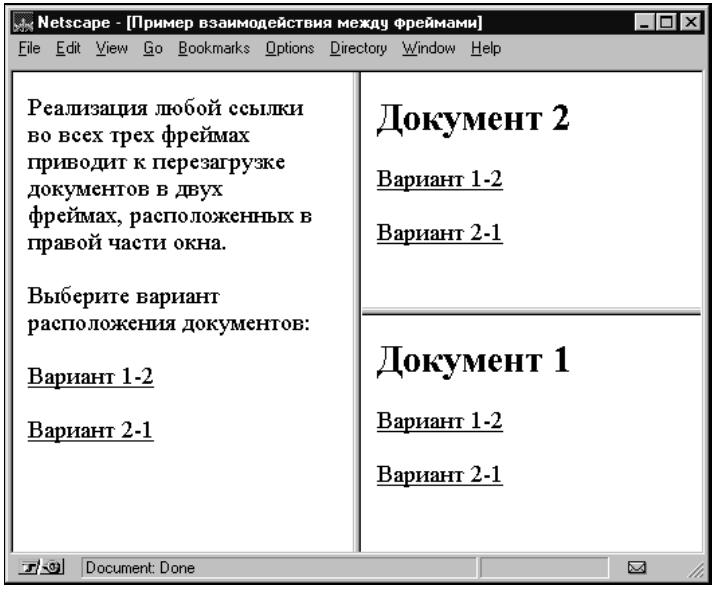


Рис. 5.13. Окна взаимодействующих фреймов с эффектом смены загружаемых документов

Различие между фреймами и окнами браузера

При работе с фреймами возникает вопрос о принципиальной разнице между организацией фреймовой структуры окна браузера и созданием нескольких окон. На первый взгляд может показаться, что вполне можно было бы обойтись возможностями создания нескольких окон, поскольку работа с окнами и фреймами очень похожа. Каждый фрейм требует загрузки отдельного документа, имеет возможность независимой прокрутки содержимого и может изменяться по командам из других фреймов. Эти свойства фреймов аналогичны свойствам окон браузера. При табличной организации данных добиться такой свободы действий невозможно.

Однако между фреймами и окнами есть существенная разница. При фреймовой организации деление области просмотра на фреймы выполняет сам HTML-документ, указывая размеры и их расположение. Пользователь при просмотре может изменить размеры фреймов, если это не запрещено в описании их структуры. Расположение окон определено общими правилами работы с системой Windows — пользователь может распахнуть любое окно на весь экран, свернуть его в пиктограмму или произвольным образом задать размеры и расположение. Окна, в отличие от фреймов, могут перекрываться. Такое богатство выбора имеет свою оборотную сторону — необходимо каждый раз вручную располагать окна на экране и изменять их размеры для достижения оптимального варианта просмотра. В случае фреймов оптимальный вариант соотношения размеров обычно задается разработчиком в описании фреймовой структуры и часто не нуждается в изменении.

Совет

Хотя фреймы и не могут предоставить всех возможностей работы с отдельными окнами, их рациональная организация создаст для пользователя максимум удобств.

Работе с окнами свойственны и другие недостатки. Создание каждого окна требует немало памяти. По существу в Netscape каждое окно представляет собой еще одну копию браузера и снабжено полным набором кнопок и меню. Та же ситуация характерна и для Microsoft Internet Explorer.

Заметим, что организация отдельных окон в браузерах выполнена по-разному. Создание нового окна с документом приводит к появлению отдельной задачи в системе Windows, в чем можно убедиться при просмотре списка запущенных задач. Поэтому переключение между окнами может выполняться так же, как и переключение между разными задачами, например, нажатием сочетания клавиш <Alt>+<Tab>.

Во многих популярных Windows-приложениях существует понятие окна с документом (document window). В качестве примера можно привести тексто-

вый процессор Microsoft Word или программу работы с графическими изображениями Paint Shop Pro и многие другие. В каждом из таких приложений допустимо одновременное использование нескольких окон с данными и, как правило, существует меню **Window**, в котором приводится список окон и дается возможность переключения между ними. Образование нового окна в таких приложениях обычно происходит при открытии существующего файла или создании нового. Однако в этих программах при создании нового окна не образуется новая работающая задача.

В Netscape также имеется меню **Window**, в котором приводится список существующих окон. (В версиях Netscape 4.x эта возможность предоставляется пунктом **Window** меню **Communicator**.) Вернемся к рис. 5.12. В этом примере одновременно открыто три окна, каждое из которых по существу является отдельным браузером. Однако для пользователя все они являются окнами одного браузера, которые могут взаимодействовать друг с другом. В любом из этих окон можно открыть меню **Window** и увидеть список трех окон. На рис. 5.14 показана ситуация, когда это сделано для нижнего окна.

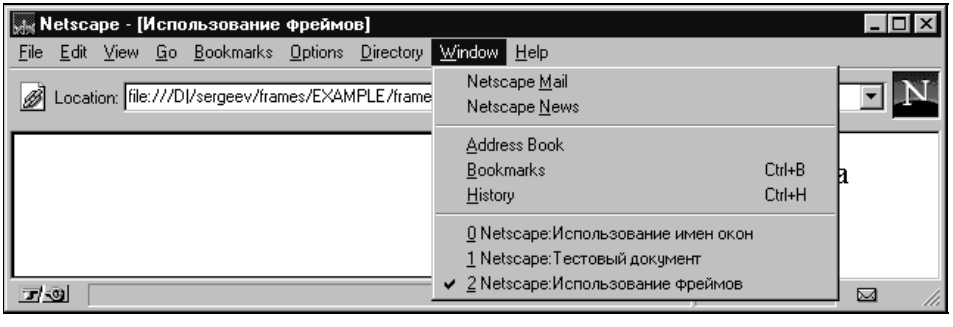


Рис. 5.14. Открытие окон фреймов в браузере Netscape

Каждое из окон может быть закрыто отдельно (командой **Close** из меню **File**). Для завершения работы с браузером в любом из окон можно открыть меню **File** и выполнить пункт **Exit** (рис. 5.15).

Если при этом было открыто несколько окон, то все они будут закрыты, но перед этим появится предупреждающее сообщение (рис. 5.16).

Каждое окно браузера может иметь свои установки (правда не все). Посмотрите на рис. 5.11. Открыто два окна, одно из которых разбито на три фрейма. В два из трех фреймов, а также в отдельное окно загружен один и тот же документ. Возможность независимой настройки параметров каждого окна позволяет по-разному отобразить один и тот же документ. В приведенном примере размер шрифта документа в одном окне больше, чем в другом. Такой эффект достигнут установкой различных кодировок каждого из окон (пункт **Document Encoding** меню **Options** или пункт **Character Set** меню **View** для версии 4.x), причем обе кодировки используют один и тот же шрифт, но

разного размера. Изменение же любого пункта меню **General Preferences** воздействует на все окна.

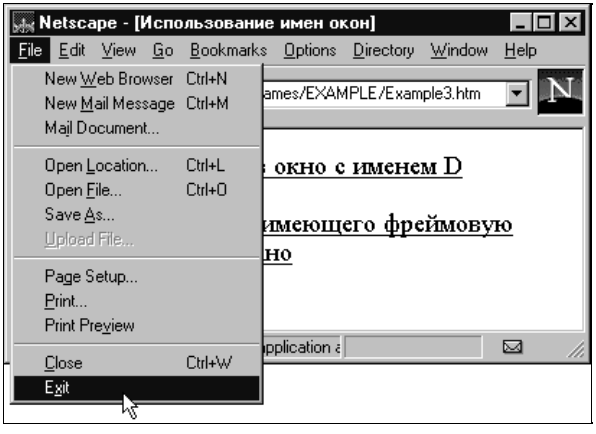


Рис. 5.15. Завершение работы с браузером Netscape

Рис. 5.16. Предупреждение о закрытии окон в браузере Netscape

Дополнительные возможности браузеров

Все перечисленные выше тэги описания фреймов с соответствующими параметрами практически одинаково реализованы в браузерах Netscape и Microsoft Internet Explorer, однако каждый из этих браузеров позволяет дополнительно использовать свои уникальные тэги или параметры.

Возможности браузера Netscape

Браузер Netscape, начиная с версии 3.0, позволяет использовать три дополнительных параметра: `BORDER`, `FRAMEBORDER` и `BORDERCOLOR`. Параметр `BORDER` применяется только в тэге `<FRAMESET>`. Значение параметра `BORDER` определяет толщину рамок между фреймами в пикселах.

Параметр `FRAMEBORDER` может применяться как в тэге `<FRAMESET>`, так и в тэге `<FRAME>` и определяет наличие рамки между фреймов. Этот параметр может принимать значение `Yes` или `No`. Если параметр записан в тэге `<FRAMESET>`, то его действие распространяется на все фреймы этой группы. Для отдельного фрейма значение может быть переопределено. По умолчанию принимается значение `Yes`.

Заметим, что параметры `BORDER` и `FRAMEBORDER` работают независимо друг от друга. Например, если в качестве значения `FRAMEBORDER` задано `No`, а для `BORDER` задано значение, отличное от нуля, то рамка между фреймами прорисовываться не будет, но место под нее, определенное значением параметра `BORDER`, все равно будет отведено.

Параметр BORDERCOLOR может применяться как в тэге <FRAMESET>, так и в тэге <FRAME> и определяет цвет рамки, который может задаваться названием цвета или его шестнадцатеричным представлением.

Приведем пример:

```
<FRAMESET COLS=2*,*,* BORDER=10 FRAMEBORDER=No BORDERCOLOR=red>
<FRAME SRC=frame_a.htm NAME="A">
<FRAME SRC=empty.htm NAME="B">
<FRAME SRC=empty.htm NAME="C" FRAMEBORDER=Yes>
</FRAMESET>
```

Первая строка этого HTML-кода задает три фрейма, между которыми оставлено место под рамку толщиной 10 пикселов (рис. 5.17).

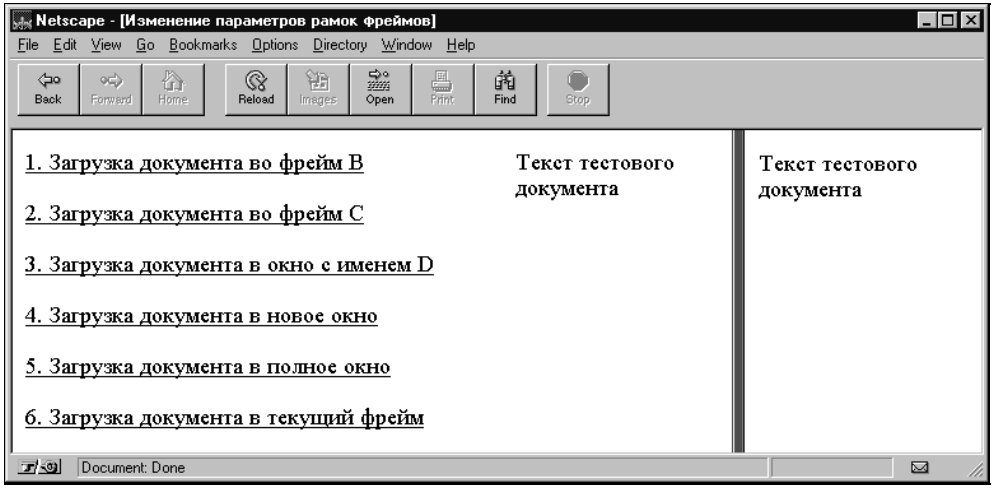


Рис. 5.17. Прорисовка рамок между фреймами в браузере Netscape

Между окнами фреймов "A" и "B" рамка не прорисовывается благодаря значению No параметра FRAMEBORDER, тем не менее для рамки определен красный цвет (red). Для последнего фрейма "C" значение FRAMEBORDER задано равным Yes и переопределяет значение, установленное в первой строке. Поэтому между фреймами с именами "B" и "C" все-таки будет нарисована рамка красного цвета и толщиной 10 пикселов.

Примечание

Если рамки между фреймами не прорисовываются, то браузер Netscape не позволит изменять размеры фреймов путем перетаскивания рамок мышью даже при отсутствии параметра NORESIZE. Для Microsoft Internet Explorer ситуация иная.

Заметим, что фреймы без рамок используются не так уж и редко (например, см. рис. 5.2). Следует помнить, что отсутствие рамок не запрещает появление полос прокрутки (рис. 5.18).

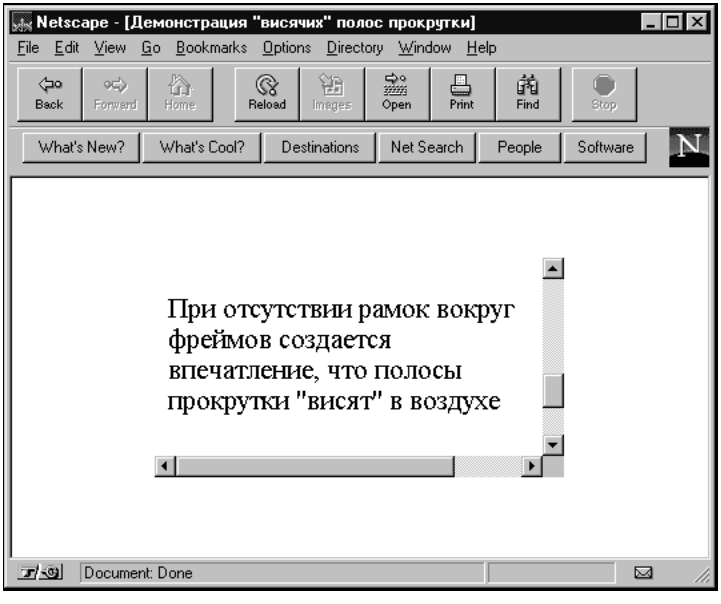


Рис. 5.18. Полосы прокрутки во фрейме без рамок

Возможности браузера Microsoft Internet Explorer

Браузер Microsoft Internet Explorer разрешает использовать параметр `FRAMEBORDER` для тех же целей, как было описано выше, но не позволяет задавать цвет и толщину рамок. Однако в качестве значения параметра `FRAMEBORDER` допустимо использовать только числовое значение "0" для отмены прорисовки рамки или отличное от нуля числовое значение для прорисовки рамки.

Различие в правилах задания значений параметра `FRAMEBORDER` для разных браузеров весьма неприятно. Попробуйте, например, задать `FRAMEBORDER=Yes`. Такая запись верна для Netscape, а для Microsoft Internet Explorer приведет к отсутствию рамки. Предыдущий пример (рис. 5.17) при просмотре в Microsoft Internet Explorer будет представлен без рамки.

Совет

Можно рекомендовать всегда записывать значение параметра `FRAMEBORDER` в числовом виде, например, `FRAMEBORDER=0`. Это соответствует правилам записи параметра для Microsoft Internet Explorer, но нарушает правила для Netscape (хотя и верно воспринимается последним).

Примечание

Если рамки между фреймами не прорисовываются, то браузер Microsoft Internet Explorer (в отличие от Netscape) при отсутствии параметра NORESIZE позволит "наощупь" изменять размеры фреймов путем перетаскивания рамок мышью. Поймать место, где должна находиться рамка, можно по изменению формы указателя мыши.

Браузер Microsoft Internet Explorer разрешает использовать дополнительный параметр FRAMESPACING, записываемый в тэге <FRAMESET>, значение которого определяет количество пикселов между фреймами, оставляемое пустыми.

Приведем пример, результат отображения которого приводится на рис. 5.19.

```
<HTML>
<HEAD>
<TITLE>Изменение расстояния между фреймами</TITLE>
</HEAD>
<FRAMESET COLS=2*,*,* FRAMESPACING=30>
<FRAME SRC=frame_a.htm NAME="A">
<FRAME SRC=empty.htm NAME="B">
<FRAME SRC=empty.htm NAME="C">
</FRAMESET>
</HTML>
```

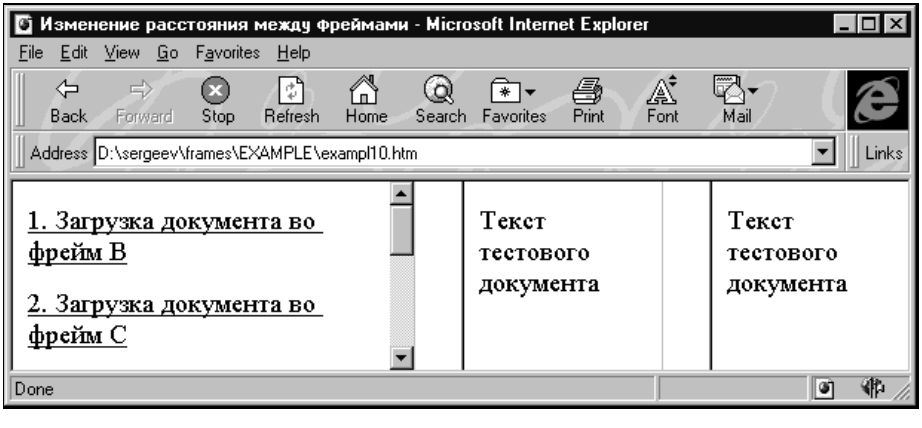


Рис. 5.19. Пустое пространство между фреймами в Microsoft Internet Explorer

Примечание

К сожалению, во многих описаниях языка HTML ошибочно указывается, что параметр FRAMESPACING должен использоваться в тэге <FRAME>. Microsoft Internet Explorer допускает использование этого параметра только в тэге <FRAMESET>.

Плавающие фреймы

Браузер Microsoft Internet Explorer разрешает использовать уникальный тэг <IFRAME>, который реализует концепцию плавающих фреймов. В отличие от обычных фреймов описание плавающих фреймов может встречаться в тексте обычного HTML-документа. Браузеры, не поддерживающие тэг <IFRAME>, вместо фрейма будут отображать любую информацию, записанную между <IFRAME> и </IFRAME>. В тэге <IFRAME> применяются точно такие же параметры, как и в тэге описания обычных фреймов <FRAME>. Единственным исключением является параметр NORESIZE, применение которого бессмысленно, так как размер плавающих фреймов в любом случае не может быть изменен пользователем при просмотре документа.

Кроме того, для задания расположения и размеров плавающего фрейма в документе можно использовать следующие дополнительные параметры: WIDTH, HEIGHT, HSPACE, VSPACE, ALIGN. Их назначение и порядок использования совпадает с соответствующими параметрами для встроенных изображений, которые задаются тэгом <IMG>.

Приведем пример использования плавающих фреймов:

```
<HTML>
<HEAD>
<TITLE>Использование плавающих фреймов</TITLE>
</HEAD>
<BODY>
<CENTER><H2>Пример использования концепции плавающих фрей-
мов</H2></CENTER>
<IFRAME SRC=float.htm NAME="A" HEIGHT=300 WIDTH=40% HSPACE=10
SCROLLING=YES ALIGN=RIGHT>
Ваш браузер не позволяет отображать плавающие фреймы
</IFRAME>
Браузер Microsoft Internet Explorer — первый из браузеров (и пока
единственный), который поддерживает так называемые "плавающие" фреймы.
Такие фреймы могут размещаться в любом месте экрана так же, как
графические изображения и таблицы.
Фрейм справа от данного текста размещен на странице с помощью
специального тэга <IFRAME>. При создании фрейма было указано
выравнивание вправо.
</BODY>
</HTML>
```

Результат отображения данного примера браузером Microsoft Internet Explorer показан на рис. 5.20. Браузеры, не поддерживающие концепцию плавающих фреймов, для данного примера вместо отображения содержимого документа float.htm выдадут текст "Ваш браузер не позволяет отображать плавающие фреймы".

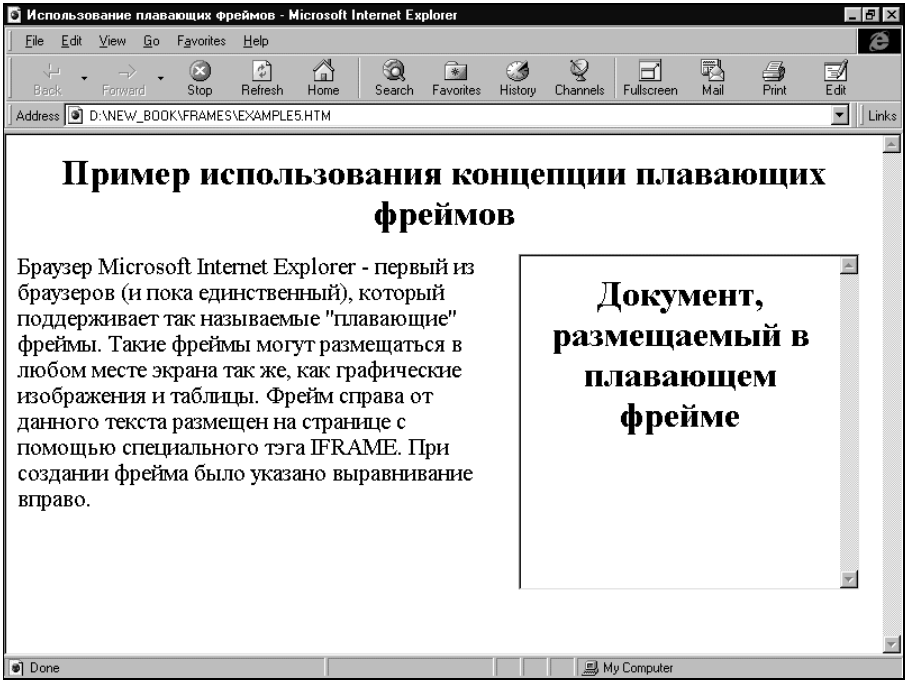


Рис. 5.20. Плавающий фрейм в Microsoft Internet Explorer

Отметим, что концепция плавающих фреймов близка по идеологии к встроенным изображениям или таблицам. Здесь в нужное место HTML-документа целиком встраивается другой HTML-документ.

Совет

В настоящий момент применение плавающих фреймов ограничивается единственным браузером — Microsoft Internet Explorer версии 3.0 и выше. Следует помнить, что пользователи других браузеров (в частности, Netscape) не смогут увидеть содержимого плавающих фреймов.

Средства создания документов, содержащих фреймы

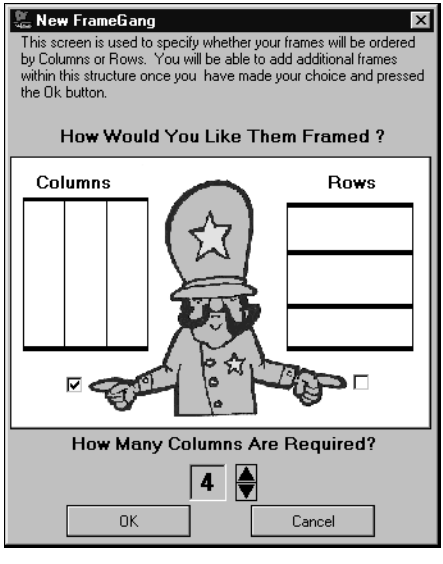
HTML-документ, содержащий фреймы, как и любой другой документ, может быть создан или отредактирован вручную при помощи любого доступного текстового редактора. Большинство специализированных HTML-редакторов либо не имеют возможностей визуального создания фреймов, либо обладают весьма ограниченными возможностями. Существует несколько специальных редакторов, которые ориентированы на создание фреймов. Кратко опишем возможности некоторых из них.

Редактор фреймов FrameGang

Одним из таких редакторов является утилита FrameGang, разработанная австралийской фирмой Sausage Software, которая более известна своим популярным HTML-редактором HotDog.

Информацию об этом программном продукте можно получить по адресу **http://www.sausage.com**, а также из сборника программных продуктов для Интернета (по адресу **http://www.tucows.com** или любому другому из нескольких десятков серверов-зеркал, разбросанных по всему миру).

Утилита FrameGang является дополнением к любому HTML-редактору или обычному текстовому редактору, работающему в среде Windows, которая позволяет визуально сконструировать необходимую структуру фреймов и затем сгенерировать соответствующий HTML-код. Получаемый HTML-код через буфер обмена Windows (Clipboard) может быть передан в HTML-редактор. Программа FrameGang позволит быстро построить нужную фреймовую структуру.



Рассмотрим возможности программы. После установки данной программы в Windows и ее запуска появляется картинка, как на рис. 5.21.

Рис. 5.21. Первое окно программы FrameGang

Здесь предлагается выбрать одну из двух возможных структур фреймов первого (верхнего) уровня — поколонное расположение фреймов (Columns) или построчное (Rows), а также определить их количество. В дальнейшем каждый из фреймов первого уровня может при необходимости быть разбит на несколько фреймов второго уровня. Большее количество уровней фреймов не предусмотрено, однако это не является существенным ограничением, так как на практике редко используется более двух уровней.

Пусть для примера выбрано четыре фрейма, расположенных по колонкам. Далее следует создать структуру фреймов второго уровня, что осуществляет-

ся разбиением существующих фреймов в противоположном направлении. Для данного примера фреймы могут разбиваться только по горизонтали. Создание фреймов второго уровня выполняется в пункте меню **Add**, реализация которого добавляет очередной фрейм к текущему. На рис. 5.22 показана ситуация после выполнения пункта меню **Add** для третьего фрейма.

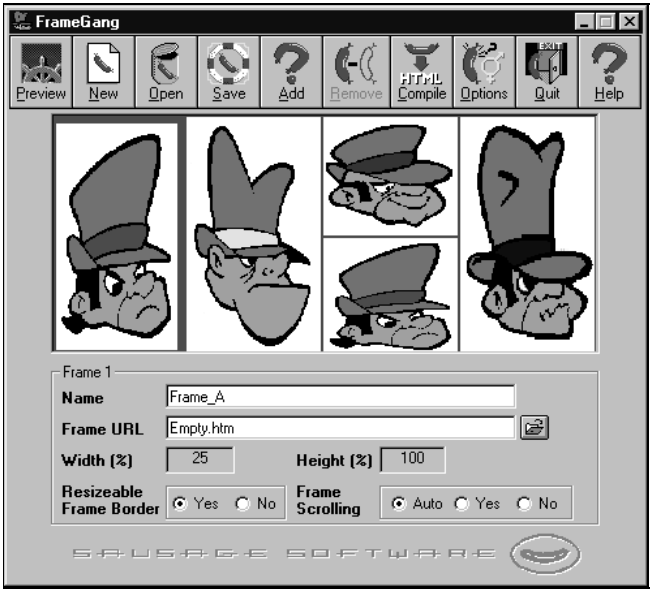


Рис. 5.22. Двухуровневая структура фреймов

Заметим, что смешные картинки во фреймах не несут смысловой нагрузки и служат лишь для заполнения пустого пространства. Их прорисовка может быть отменена в меню **Options**. Также отметим, что само название фирмы-производителя (sausage — сосиски) и название некоторых ее программных продуктов (HotDog — в переводе не нуждается) определило стиль интерфейса данного пакета — во многих пунктах меню встречается изображение сосисок.

После определения структуры фреймов следует задать их размеры. Это выполняется простым перемещением границ фреймов мышью так же, как это делается для изменения размеров окон системы Windows. Установленные размеры для текущего фрейма отображаются в процентах в окнах с названиями **Width** и **Height**. При необходимости задать размеры фреймов не в процентах, а в пикселах или относительных единицах, следует изменить получаемый HTML-код вручную вне программы FrameGang.

Далее для каждого фрейма нужно задать его имя (**Name**), которое может быть опущено, URL-адрес документа, загружаемого в данный фрейм изначально (**Frame URL**), а также выбрать значение параметра изменяемости размеров фрейма (**Resizeable Frame Border**) и параметра прокрутки содержимого фрейма (**Frame Scrolling**).

В любой момент можно посмотреть создаваемую фреймовую структуры в выбранном внешнем браузере, не выходя из программы FrameGang. Для этого служит кнопка **Preview**. Задание имени браузера производится в меню **Options**. После задания всех параметров следует сгенерировать HTML-код, соответствующий выбранной структуре фреймов, который будет записан в буфер обмена Windows (кнопка **HTML Compile**). Полученный код может быть вставлен из буфера обмена в нужное место HTML-файла при работе в любом редакторе.

Кнопки сохранения (**Save**) и открытия (**Open**) файла позволяют сохранять и считывать файл только в специальной кодировке, свойственной программе FrameGang и не предназначенной для иного использования.

Ниже представлен HTML-код, сгенерированный программой FrameGang для описанного примера:

```
<FRAMESET COLS="25%,25%,25%,25%">
       <FRAME SCROLLING=AUTO SRC="Empty.htm" NAME="Frame_A">
       <FRAME SCROLLING=AUTO SRC="Empty.htm" NAME="Frame_B">
       <FRAMESET ROWS="50%,50%">
              <FRAME SCROLLING=AUTO SRC="Empty.htm" NAME="Frame_C1">
              <FRAME SCROLLING=AUTO SRC="Empty.htm" NAME="Frame_C2">
       </FRAMESET>
       <FRAME SCROLLING=AUTO SRC="Empty.htm" NAME="Frame_D">
</FRAMESET>
```

Для получения корректного HTML-документа полученный код достаточно заключить между тэгами `<HTML>` и `</HTML>`. Можно добавить раздел заголовка документа `<HEAD>`, в котором определить нужные данные, например, название документа (напомним, что название документа записывается между тэгами `<TITLE>` и `</TITLE>`). Заметим, что современные браузеры могут правильно работать даже при отсутствии некоторых необходимых тэгов. Если приведенный выше HTML-код сохранить в виде файла с соответствующим расширением даже без добавления каких-либо тэгов, то и Netscape Navigator, и Microsoft Internet Explorer смогут правильно отобразить данный документ.

Программа FrameGang позволяет визуально проектировать фреймы, достаточна удобна и проста в эксплуатации, однако не лишена отдельных недостатков. В частности, нет возможности уточнить размеры фреймов, записывая нужные числа в окнах **Width** и **Height**, так как в них лишь отображаются размеры, устанавливаемые при перемещении границ фреймов мышью. Не задаются общепринятые параметры фреймов `MARGINWIDTH` и `MARGINHEIGHT`. Нет возможности записи файла в текстовом формате на диск, что не дает возможности воспользоваться данной программой автономно.

Редактор фреймов Frame-It

Еще одним специализированным редактором фреймов является программа Frame-It, информацию о которой (рис. 5.23) можно получить по адресу:

http://www.iinet.net.au/~bwh/frame-it.html.

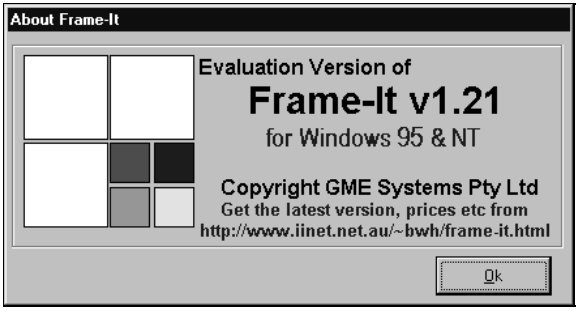


Рис. 5.23. Информация о программе Frame-It

Работа с этой программой во многом аналогична предыдущей. Сначала требуется определить основную структуру фреймов и их количество (рис. 5.24). Далее каждый фрейм при необходимости разбивается на несколько, образуя структуры второго уровня. Так же, как и предыдущей программе, количество уровней вложенности фреймов ограничивается двумя.

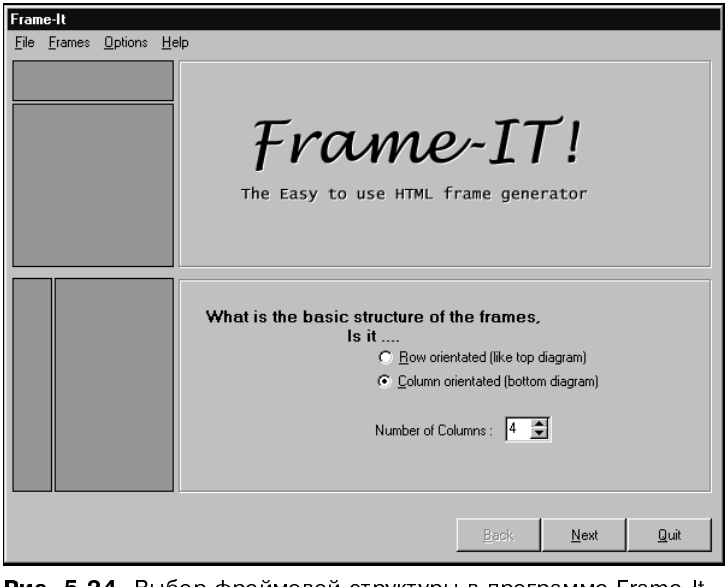


Рис. 5.24. Выбор фреймовой структуры в программе Frame-It

Для каждого фрейма задаются все необходимые параметры путем заполнения соответствующих полей (рис. 5.25). В отличие от предыдущей про-

граммы здесь предусмотрено задание значений параметров MARGINWIDTH и MARGINHEIGHT. Кроме того, введен флажок **Invisible Frame Borders**, установка которого обеспечивает генерацию следующего фрагмента кода: FRAMEBORDER="NO" BORDER="0" FRAMESPACING="0".

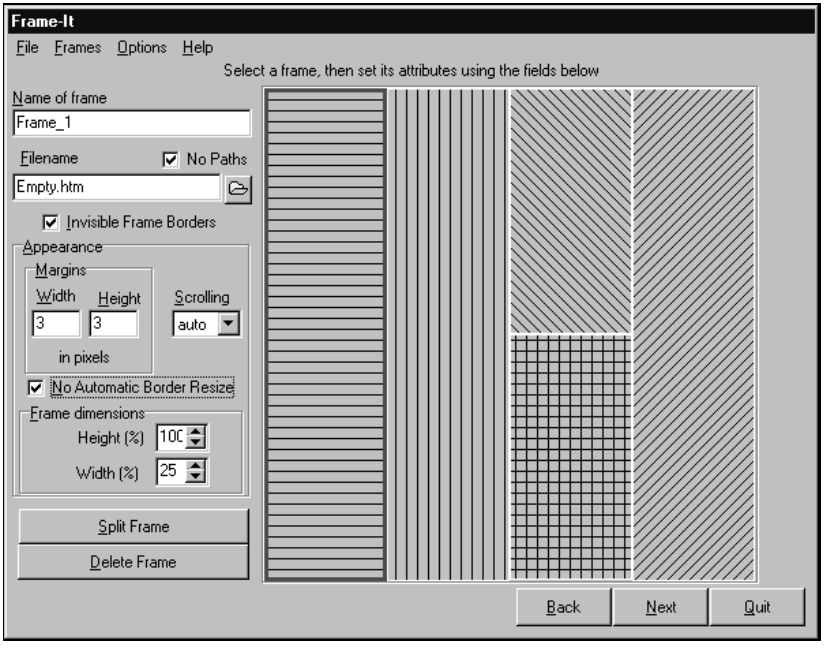


Рис. 5.25. Настройка параметров отдельных фреймов в программе Frame-It

После задания необходимых параметров фреймов можно сохранить сгенерированный HTML-код в файле или записать его в буфер обмена Windows. Сгенерированный код будет содержать не только описание структуры фреймов, но и начальный тэг <HTML>, а также пару тэгов <NOFRAMES> и </NOFRAMES>, между которыми записывается информация, предназначенная для браузеров, не отображающих фреймы (рис. 5.26).

Для примера, показанного на рис. 5.25, будет сгенерирован следующий код:

```
<html>
<! --- Generated using Frame-it v1.21 --- >
<! --- http://www.iinet.net.au/~bwh/frame-it.html --->
<frameset cols="25%,25%,25%,25%" FRAMEBORDER="NO" BORDER="0"
FRAMESPACING="0">
<frame name="Frame_1" src="Empty.htm">
<frame name="Frame_2" src="Empty.htm">
<frameset rows="50%,50%">
<frame name="Frame_3" src="Empty.htm">
```

```
<frame name="Frame_5" src="Empty.htm">
</frameset>
<frame name="Frame_4" src="Empty.htm">
</frameset>
<noframes>
<body>
Ваш браузер не может показывать документы
с фреймовой структурой
</noframes>
```

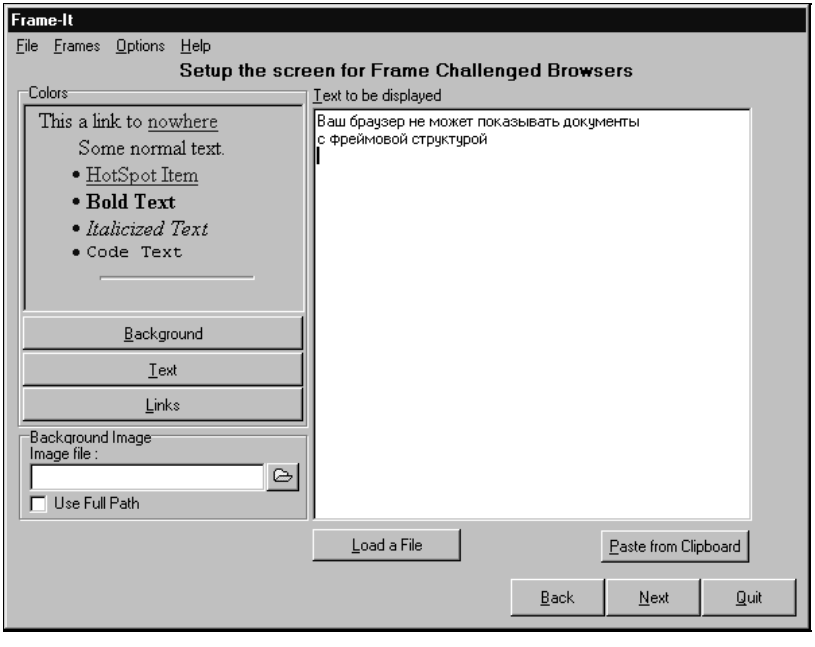


Рис. 5.26. Окно для задания альтернативного содержания для браузеров, не поддерживающих концепцию фреймов в программе Frame-It

Если в данном примере сбросить флажок **Invisible Frame Borders**, то генерируемый код значительно изменится:

```
<html>
<! --- Generated using Frame-it v1.21 --- >
<! --- http://www.iinet.net.au/~bwh/frame-it.html --->
<frameset cols="25%,25%,25%,25%">
<frame name="Frame_1" src="Empty.htm" marginheight=3 marginwidth=3
scrolling=auto noresize>
<frame name="Frame_2" src="Empty.htm" marginheight=3 marginwidth=3
scrolling=auto noresize>
<frameset rows="50%,50%">
```

```
<frame name="Frame_3" src="Empty.htm" marginheight=3 marginwidth=3
scrolling=auto noresize>
<frame name="Frame_5" src="Empty.htm" marginheight=3 marginwidth=3
scrolling=auto noresize>
</frameset>
<frame name="Frame_4" src="Empty.htm" marginheight=10 marginwidth=10
scrolling=auto noresize>
</frameset>
<noframes>
<body>
Ваш браузер не может показывать документы
с фреймовой структурой
</noframes>
```

Информация об использовании фреймов на WWW

Для получения информации о фреймах можно обратиться к следующим адресам на WWW:

http://www.spunwebs.com/sites2c/frmtutor.html

http://union.ncsa.uiuc.edu/HyperNews/get/www/html/guides.html

http://cox.asu.edu/Trial/faq/webfaqs/frame/

http://www.netlingo.com/more/framestarget.html

http://www.aubg.bg/beast/students/raduluc/teach/fr/

http://www.as.net/frame/

http://www.cqi.com/~pmurphey/instruction/

http://www.iwaynet.net/~rtyler/htmltutorial/frames.html

http://edbo.com/frames/

http://bela.fei.tuke.sk/netscape/frames/

http://www.newbie.net/frames/2ed/menu.htm#contents

http://infoserver.etl.vt.edu/coe/COE_Students/laughon/frame.html

http://www.htmlhelp.com/frames/syntax/

http://www.woodhill.co.uk/html/

http://www.htmlhelp.com/design/frames/

ГЛАВА 6

Карты-изображения

В последнее время многие Web-страницы для организации ссылок используют так называемые *карты-изображения*. Реализация этой возможности предусмотрена языком HTML и позволяет привязывать гипертекстовые ссылки к различным областям изображения. Такой подход нагляднее, чем применение обыкновенных текстовых связей, поскольку пользователь может не читать словесное описание связи, а сразу понять ее смысл по графическому образу.

Даже начинающий пользователь, побродив по просторам Интернета, вскоре столкнется с картой-изображением. На рис. 6.1 показана Web-страница одной из крупнейших компьютерных фирм Северо-Запада России. Основное меню на этой странице представляет собой карту-изображение с соответствующими ссылками.

Запуск известного поискового сервера Yahoo! также приводит к появлению страницы, содержащей карту-изображение (рис. 6.2). Самая верхняя часть изображения, приведенного на рисунке, содержит четыре кнопки, между которыми написано слово "Yahoo!". Курсор на рисунке показывает на первую из этих кнопок, причем форма курсора дает понять, что последний указывает на ссылку, адрес которой виден в строке статуса браузера. Ссылки, реализующиеся по этим кнопкам, и сделаны по технологии карт-изображений.

Однако не следует считать, что карты-изображения должны использоваться всюду, где требуется организовать переходы по ссылкам. Нужно обдумать, имеет ли смысл применение карт-изображений в том или ином случае, взвесив все "за" и "против". В данной главе содержатся все необходимые сведения, касающиеся использования карт-изображений.

В этой главе вы узнаете:

- Что такое карта-изображение, и как она работает
- Как создать файл конфигурации карты-изображения
- Как создать ссылки в HTML-документе, используя карту-изображение

Рис. 6.1. Пример Web-страницы, в которой основное меню сделано с помощью карты-изображения

Рис. 6.2. Начальная страница поискового сервера Yahoo! содержит карту-изображение

- Какими принципами следует руководствоваться при использовании карт-изображений
- В чем состоят особенности различных форматов файлов конфигурации карт-изображений
- Какими программными средствами следует воспользоваться для создания карт-изображений

Основы использования карт-изображений

Карты-изображения предоставляют пользователям дружественный интерфейс для перехода на другие Web-страницы. Чтобы выполнить переход по такой ссылке, следует просто выбрать нужное место на изображении и щелкнуть мышью. Наличие такого развитого графического интерфейса является одним из значительных преимуществ Web-страниц по сравнению с другими ресурсами Интернета. Вместо текстовых меню, подобных интерфейсу клиентов системы Gopher, пользователи получают наглядное графическое представление информации (рис. 6.3).

Рис. 6.3. Наглядное представление ссылок (на страницы, рассказывающие о семи чудесах света) с помощью карты-изображения

Карта-изображение внешне выглядит как обычное встроенное изображение, но при выборе с помощью курсора мыши той или иной области на этом изображении выполняется переход на другие страницы. Обычно на изображении указывается, где следует сделать щелчок, чтобы перейти на ту или иную страницу. Существует несколько путей указания границ областей, реализующих различные ссылки. Часто используется рамка или какой-либо иной разделитель.

Рассмотрим основные понятия, связанные с использованием карт-изображений.

Терминология

Imagemap, Image Map, Area Map, Clickable Map, Sensitive Map — все эти англоязычные термины используются в справочной литературе для обозначения одной и той же возможности — использование встроенного в HTML-документ изображения, для которого определены "горячие" (или активные) точки или области, имеющие ссылки на различные URL-адреса. Будем описывать эту возможность словосочетанием "карта-изображение", подразумевая под этим совокупность нескольких компонентов, обеспечивающих реализацию данной концепции. Основными компонентами являются: само изображение, которое будем называть опорным для данной карты-изображения; описание конфигурации активных областей; а также соответствующее программное обеспечение.

Графическое представление карты-изображения

Карта-изображение фактически представляет собой обычное встроенное графическое изображение на Web-странице. Эти изображения могут иметь любой допустимый формат (GIF или JPG). При этом в формате GIF может использоваться прозрачный цвет, а также режим чередования строк. Для того чтобы изображение могло использоваться в качестве опорного для карты-изображения, формально не накладывается никаких дополнительных ограничений.

Описание конфигурации карты-изображения

Конфигурация карты-изображения записывается в виде обычного текста, который в зависимости от используемого формата может быть сохранен в отдельном файле или являться частью HTML-документа. Описание конфигурации содержит координаты для каждой из активных областей изображения, а также URL-адреса, связанные с каждой из этих областей. Активные

области могут иметь форму прямоугольников, кругов и многоугольников. Допускается любая комбинация этих фигур. Также может задаваться одно значение URL-адреса, определенное для случая, когда пользователь выполняет щелчок в пределах изображения, но вне любой из заданных активных областей. Конкретные правила записи конфигурации области зависят от выбранного варианта реализации и будут представлены далее.

Варианты реализации карт-изображений

Концепция карты-изображения на Web-страницах может быть реализована в двух различных вариантах — серверный вариант (server-side imagemap) и клиентский вариант (client-side imagemap). Последнее название часто используют в виде аббревиатуры CSIM. Исторически первым появился и получил распространение *серверный вариант* карт-изображений, который впервые был реализован в браузере Mosaic. Серверный вариант позволяли использовать первые версии всех трех ведущих браузеров. Серверный вариант может быть реализован в двух различных форматах, которые получили свое наименование по названиям организаций-разработчиков — NCSA и CERN.

В последнее время все большее развитие получает *клиентский вариант*, который впервые был реализован в браузере Microsoft Internet Explorer. Начиная с версии 2.0, этот вариант также поддерживает браузер Netscape. Данный вариант имеет свои неоспоримые преимущества и становится все более популярным.

Преимущества и недостатки карт-изображений

В использовании карт-изображений есть как положительные, так и отрицательные моменты. Большинство из них носит эстетический характер, но некоторые имеют и технические аспекты. Для создания хороших Web-страниц важно понимать преимущества и недостатки карт-изображений.

Карты-изображения наиболее удобно использовать в следующих ситуациях:

- Для представления пространственных связей, например географических координат, которые было бы трудно задать отдельными кнопками или текстом На рис. 6.4 в качестве примера приведена карта Северной Америки, на которой выбор каждого из штатов ведет к переходу на соответствующую страницу.
- В качестве меню верхнего уровня, появляющегося на каждой странице. Наличие такого меню предоставляет возможность перехода в интересующий раздел сервера с любой страницы и в любой момент (рис. 6.5). Создание общего графического меню позволит сократить время разработки

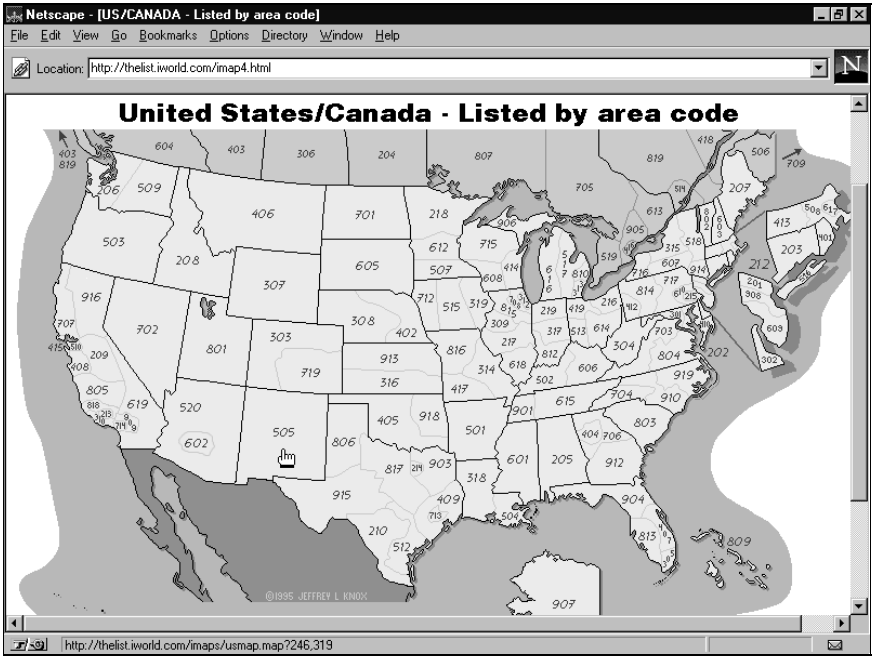


Рис. 6.4. Карта-изображение для представления пространственных связей

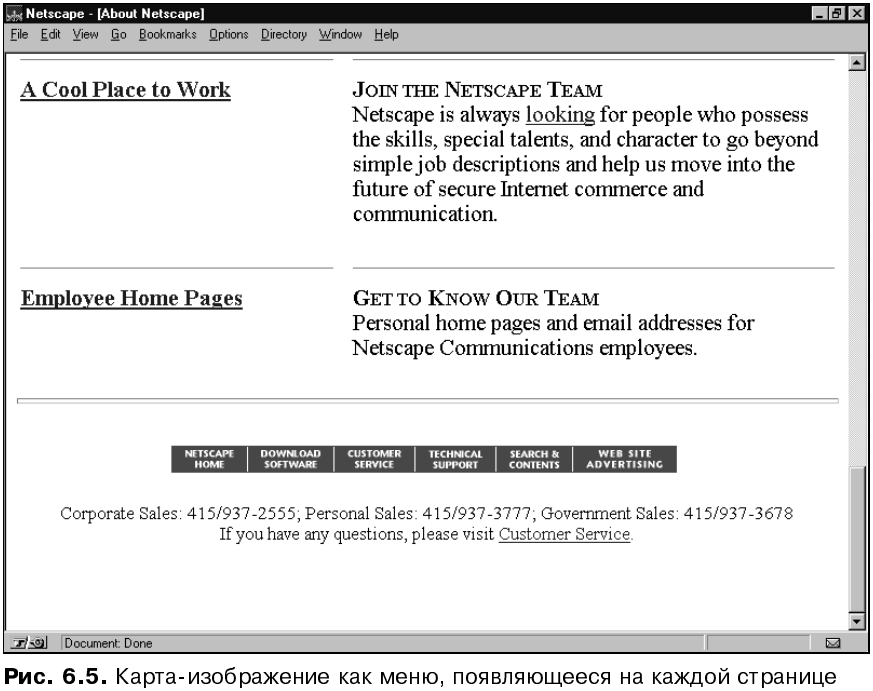


Рис. 6.5. Карта-изображение как меню, появляющееся на каждой странице

HTML-документов, поскольку будет использоваться один и тот же файл описания ссылок. Вместо того, чтобы на каждой странице устанавливать связи с различными частями начальной страницы, достаточно сослаться на общее меню. Такое меню также облегчит навигацию для пользователя.

Несмотря на то, что карты-изображения стали необычайно популярны, очевидно, что они не являются неотъемлемым атрибутом Web-страниц и используются далеко не на всех страницах. Есть ситуации, когда не следует использовать карты-изображения.

К недостаткам карт-изображений можно отнести следующие:

- Если не предусмотрено альтернативное текстовое меню, то не остается никаких средств навигации для пользователей, которые не могут загрузить графику или отключили ее загрузку.
- Картам-изображениям свойственны общие недостатки, присущие использованию изображений на Web-страницах, а именно, значительное увеличение времени загрузки по сравнению с чисто текстовыми документами.
- Неудачно спроектированные изображения могут внести путаницу. Иногда бывает трудно определить области, являющиеся активными на изображении. Особенно это трудно сделать в серверном варианте. При реализации клиентского варианта ситуация упрощается, так как есть возможность перемещать мышь в пределах изображения и следить за появляющимися адресами ссылок в нижней части окна браузера.
- При использовании карт-изображений браузер не имеет возможности отмечать другим цветом уже пройденные ссылки так, как это делается для текстовых ссылок.

Серверный вариант реализации карт-изображений

Использование на Web-страницах карт-изображений оказывается несколько более сложным делом, чем простое встраивание интересных графических изображений и связывание с ними ссылок. Для реализации серверного варианта карты-изображения необходимо, чтобы HTML-документ был размещен на сервере. Также требуется, чтобы на сервере была сконфигурирована поддержка CGI-сценариев (Common Gateway Interface, Общий интерфейс шлюза), которые выполняют обработку запросов, поступающих от браузера при работе с картой-изображением. Для каждой карты-изображения на сервере должен быть размещен файл, описывающий конфигурацию активных областей. При щелчке мышью в пределах изображения координаты места щелчка передаются браузером серверу, который обращается к конфигурационному файлу, являющемуся, по существу, поисковой таблицей активных областей. Результат поиска возвращается браузеру в виде URL-адреса или

сообщения об отсутствии найденных активных областей, соответствующих указанному месту изображения.

Для обеспечения функционирования карты-изображения необходимо указать, что данное изображение является опорным для карты. Это выполняется заданием параметра ISMAP в тэге <IMG>. Кроме того, карту-изображение необходимо сделать ссылкой на Web-странице, во многом подобно тому, как это делалось при использовании всего изображения в качестве отдельной ссылки.

Напомним, что встроенные изображения могут использоваться как гипертекстовые ссылки, если они включены в тэг <A>. Например, чтобы сделать изображение с именем MyImage.gif графическим указателем ссылки на документ в том же самом каталоге с именем example.html, следует записать:

```
<A HREF=example.html ><IMG SRC=MyImage.gif></A>
```

Этот HTML-код сообщает серверу, что при щелчке на изображении MyImage.gif браузеру должен быть возвращен документ с именем example.html.

Параметр ISMAP, добавленный к тэгу <IMG> для приведенного примера, активизирует карту-изображение. Ссылка в этом случае выполняется не к определенному документу, а к файлу конфигурации карты-изображения, содержащему координаты всех активных областей изображения. Файл конфигурации, обычно имеющий расширение MAP, анализируется CGI-программой на сервере вкупе с координатами точки щелчка на изображении. Тогда вместо приведенной строки следует записать:

```
<A HREF=MyImage.map><IMG SRC=MyImage.gif ISMAP></A>
```

Ссылка в этом примере представляет собой не адрес другого HTML-документа, а файл конфигурации карты-изображения, который содержит координаты для каждой активной области изображения с именем MyImage.gif.

Примечание

Порядок расположения параметров тэга <IMG> произволен, однако, параметр ISMAP обычно помещают последним.

Файл конфигурации изображения-карты — это обычный текстовый файл, который содержит информацию об активных областях данного изображения. Для каждого изображения, которое будет использоваться в режиме карты, требуется отдельный конфигурационный файл.

Совет

Для каждой карты-изображения необходим отдельный файл конфигурации. Примите за правило сохранять файл конфигурации в том же самом каталоге и с тем же именем, что и связанное с ним изображение. Например: main_menu.gif и main_menu.map.

Существуют два формата файлов конфигурации карт-изображений, разработанные в CERN и NCSA и носящие названия этих организаций. Оба этих формата содержат одинаковую информацию, но по-разному ее представляют. В обоих случаях используются одни и те же типы областей, о которых речь пойдет ниже. При разработке карт-изображений для использования на конкретном сервере необходимо получить у администратора системы сведения о принятом на данном сервере способе поддержки карт-изображений.

В обоих форматах используются активные области в виде прямоугольников, кругов и многоугольников, а также может задаваться так называемая область по умолчанию (default), которая характеризует все точки области, не принадлежащие ни к одной из активных областей. Если пользователь выполняет щелчок внутри изображения, но вне любой из заданных активных областей, то в качестве URL-адреса будет взято значение, определяемое типом `default`.

Для каждой области в файле конфигурации записывается URL-адрес, который будет возвращен пользователю, когда произойдет щелчок внутри данной области. Этот адрес может быть написан как в относительной, так и в абсолютной форме. Нужно учесть, что относительный URL-адрес должен быть определен по отношению к расположению файла конфигурации, а не файла изображения. Список активных областей в файле конфигурации читается, начиная с первой строки. Если две области перекрываются, то реализуется та ссылка, описание области действия которой встретится первым в файле конфигурации.

Совет

В файле конфигурации рекомендуется всегда задавать ссылку по умолчанию. Заданная по умолчанию связь будет реализовываться для областей изображения, не относящихся к активным. В простейшем случае URL-адрес, являющийся ссылкой по умолчанию, может просто указывать на страницу с полезной информацией об использовании данной карты.

Далее рассмотрим особенности записи и использования различных форматов серверного варианта карт-изображений.

Формат CERN

CERN — это европейский научный центр, тематика исследований которого весьма широка. Именно здесь была разработана концепция системы World Wide Web, которая явилась толчком всех разработок в области WWW. По праву CERN можно считать родиной Web. Когда возникла необходимость разработки структуры файлов конфигурации карт-изображений, в CERN был предложен следующий формат:

```
тип_области координаты URL-адрес
```

Значения пар координат X и Y разделяются запятой и заключаются в круглые скобки. Формат CERN не допускает использования комментариев для пояснения ссылок, связанных с той или иной областью. Могут использоваться следующие типы областей: `rect`, `circle`, `poly` и `default`. Этот формат допускает двоякое написание названий типов областей — как в сокращенной, так и в полной форме. Наряду с приведенными названиями типов могут использоваться названия `rectangle`, `circ` и `polygon`.

Приведем пример записи информации об участках на карте-изображении в формате CERN:

```
rect (56,47) (357,265) http://www.anywhere.com/
circ (366,147) 109 http://www.anywhere.com/
polygon (534,62) (699,62) (698,236) (626,261) (534,235) (534,62)
http://www.anywhere.com/
```

Формат NCSA

Национальный центр суперкомпьютерных приложений NCSA (National Center for Supercomputing Applications) университета штата Иллинойс также внес значительный вклад в развитие Web. Здесь был создан первый популярный графический браузер — программа Mosaic. В NCSA был предложен формат конфигурационного файла, отличающийся по форме записи от формата CERN. Этот формат имеет следующий вид:

```
тип_области URL-адрес координаты
```

Могут использоваться следующие типы областей: `rect`, `circle`, `poly`, `default` и `point`.

Координаты X и Y отделяются запятыми, но не заключаются в круглые скобки. В этом формате допускается использование строк комментариев. Любая строка, начинающаяся с символа #, будет рассматриваться как комментарий, и ее содержимое будет игнорироваться программой интерпретации.

Формат NCSA предлагает несколько отличный способ для задания круговых областей (по сравнению с форматом CERN и рассматриваемым ниже клиентским вариантом). Круговая область задается координатами двух точек — центра и любой точки, лежащей на окружности.

Примечание

Формат NCSA допускает использование типа области `point` (точечная область). Этот тип области не применяется ни в формате CERN, ни при использовании клиентского варианта карт-изображений. Замысел создателей формата заключался в том, что при наличии ряда точечных областей по щелчку мыши активизировалась ссылка, находящаяся ближе других. Однако, наличие такого типа области, по существу, входит в противоречие с типом области `default`, так как при одновременном использовании областей `point` и `default` реализа-

ция ссылки, определенной типом point, возможна только при точном попадании в данную точку мышью. Это довольно трудно и вряд ли создаст комфортные условия при работе с таким документом. В настоящее время тип point практически не применяется, причем все большее развитие получает клиентский вариант карт-изображений.

Приведем пример записи конфигурационного файла формата NCSA:

```
# Пример записи конфигурационного файла
rect http://www.anywhere.com/ 56,47 357,265
circle http://www.anywhere.com/ 366,147 366,256
poly http://www.anywhere.com/ 534,62 699,62 698,236 626,261 534,235 534,62
```

Клиентский вариант карты-изображения

Клиентский вариант карты-изображения позволяет разместить всю информацию о конфигурации карты в HTML-файле, в который встроено изображение. В случае же применения серверного варианта браузер посылает запрос на сервер для получения адреса выбранной ссылки и ждет ответа с требуемой информацией. Это может потребовать дополнительных затрат времени на ожидание. При клиентском варианте число обращений к серверу уменьшается, и увеличивается скорость доступа к информации. В этом варианте также для редактирования конфигурации карты нет необходимости обращения к серверу, поэтому вся работа по созданию карты-изображения может быть выполнена локально, одновременно с редактированием HTML-файла. В отличие от серверного варианта, в котором для каждой карты-изображения требовался отдельный конфигурационный файл, в этом варианте конфигурация карты может располагаться непосредственно в том же HTML-документе, в котором задана ссылка на опорное изображение. Чаще всего именно так и поступают, хотя допустимо сохранять конфигурацию карты в отдельном файле и давать на него ссылку.

Для указания того, что встроенное изображение является опорным для карты, используется параметр USEMAP тэга <IMG>. Значением параметра USEMAP является ссылка на описание конфигурации карты.

Примечание

Браузер Netscape не допускает использование отдельного файла для описания конфигурации карты.

Например:

```
<IMG SRC=logo.gif USEMAP=#logo>
```

В этом примере изображение, хранящееся в файле с именем logo.gif, является опорным для карты-изображения, реализуемой в клиентском варианте.

Описание конфигурации активных областей должно располагаться в том же файле, что и данная строка HTML-кода, и иметь для данного примера имя logo.

Тэг *<MAP>*

Для описания конфигурации областей карты-изображения используется специальный тэг <MAP>, единственным параметром которого является NAME. Значение параметра NAME определяет имя, которое должно соответствовать имени в USEMAP. Тэг <MAP> требует закрывающего тэга </MAP>. Внутри этой пары тэгов должны располагаться описания активных областей карты, для чего используется специальный тэг <AREA>.

Тэг *<AREA>*

Каждый отдельный тэг <AREA> задает одну активную область. Завершающий тэг не требуется. Активные области могут перекрываться. В случае если некоторая точка относится одновременно к нескольким активным областям, то будет реализована та ссылка, описание которой располагается первым в списке областей.

Параметрами тэга <AREA> являются SHAPE, COORDS, HREF, NOHREF, TARGET, и ALT. Рассмотрим назначение этих параметров.

Параметр *SHAPE*

Параметр SHAPE определяет форму активной области. Допустимыми значениями являются rect, circle, poly, default. Эти значения задают области в виде прямоугольника, круга, многоугольника. Последнее значение — default — определяет все точки области. Если параметр SHAPE опущен, то по умолчанию предполагается значение rect, т. е. область в виде прямоугольника.

Предупреждение

Не следует путать область типа default, которая описывает все точки изображения, и значение параметра SHAPE по умолчанию, которым является rect.

Примечание

В отличие от серверного варианта, где область типа default определяла все точки на изображении, которые не относятся к какой-либо активной области, для клиентского варианта область типа default определяет вообще все точки изображения. Поэтому в данном случае описание области default должно располагаться последним в списке активных областей. Если, например, описание области default поставить первым, то всегда для клиентского варианта будет реализовываться ссылка, определяемая данной областью, а все остальные ссылки будут игнорироваться (именно так реализован данный тип области

в Netscape). Для серверного варианта расположение описания области `default` не имеет значения. Это различие учитывается в примерах, приводимых в конце главы.

Совет

Отметим также, что не все браузеры поддерживают тип области `default`. В частности, Microsoft Internet Explorer вообще не разрешает использовать данный тип области. Поэтому вместо области типа `default` можно рекомендовать задание прямоугольной области с размерами, равными размерам всего изображения. Естественно, что такая область должна описываться последней. Именно так поступают некоторые программы редактирования карт-изображений, которые будут рассматриваться ниже.

Параметр *COORDS*

Параметр `COORDS` задает координаты отдельной активной области. Значением параметра является список координат точек, определяющих активную область, разделенных запятыми. Координаты записываются в виде целых неотрицательных чисел. Начало координат располагается в верхнем левом углу изображения, которому соответствует значение 0,0. Первое число определяет координату по горизонтали, второе — по вертикали. Список координат зависит от типа области.

Для области типа `rect` задаются координаты верхнего левого и правого нижнего углов прямоугольника.

Для области типа `circle` задаются три числа — координаты центра круга и радиус.

Для области типа `poly` задаются координаты вершин многоугольника в нужном порядке. Заметим, что последняя точка в списке координат не обязательно должна совпадать с первой. Если они не совпадают, то при интерпретации данных для этой формы области браузер автоматически соединит последнюю точку с первой. Различные редакторы карт-изображений в этом отношении работают по-разному — одни добавляют первую точку в конец списка, а другие — нет. Количественные ограничения на число вершин довольно велики и покрывают практически все мыслимые потребности. По крайней мере многоугольник, имеющий100 вершин, уверенно обрабатывается всеми ведущими браузерами. Есть ограничение, связанное с самим языком HTML, согласно которому список не может содержать более 1024 значений. Многоугольник вполне может быть невыпуклым.

Область типа `default` не требует задания координат.

Параметры *HREF* и *NOHREF*

Параметры `HREF` и `NOHREF` являются взаимоисключающими. Если не задан ни один из этих параметров, то считается, что для данной области не имеется

ссылки. То же самое явно определяет параметр NOHREF, не требующий значения. Параметр HREF определяет адрес ссылки, который может записываться в абсолютной или относительной форме. Правила записи полностью совпадают с правилами записи ссылок в тэге <A>.

Параметр NOHREF полезно использовать для исключения части активной области. Пусть, например, необходимо создать активную область в виде кольца. Такой тип области не предусмотрен в списке возможных областей, однако он может быть реализован путем задания двух круговых областей. Для этого сначала следует задать область меньшего радиуса и указать в качестве параметра NOHREF. Далее нужно задать область большего радиуса с центром в той же точке и указать нужную ссылку. Тогда область внутри кольца, определенная двумя окружностями различного радиуса, будет иметь необходимую ссылку. Использование подхода, основанного на взаимном перекрытии областей, позволит строить области весьма разнообразной формы.

Параметр *TARGET*

Параметр TARGET употребляется при работе с фреймами. Его назначение — указать имя фрейма, в который будет размещен документ, загружаемый по данной ссылке. Более подробную информацию об использовании этого параметра можно получить из главы, посвященной работе с фреймами.

Параметр *ALT*

Параметр ALT позволяет записать альтернативный текст для каждой из активных областей изображения. По существу этот текст будет играть лишь роль комментария для создателя документа. Если альтернативный текст, записанный для всего изображения (в тэге <IMG>), служит для выдачи его на экран при работе с отключенной загрузкой изображений, то альтернативный текст для активных областей никогда на экране не появится.

Приведем пример задания областей различных типов:

```
<MAP NAME="logo">
<AREA SHAPE=rect COORDS="33,60,191,246" HREF="r.htm" ALT="Прямоугольная
область">
<AREA SHAPE=circle COORDS="366,147,109" HREF="c.htm" ALT="Круговая область">
<AREA SHAPE=poly COORDS="534,62,699,62,698,236,626,261,534,235"
HREF="p.htm" ALT="Многоугольник">
<AREA SHAPE=default HREF="default.htm">
</MAP>
```

Этот фрагмент кода размещается в HTML-файле. Часто все описания карт-изображений одного документа сводятся вместе и размещаются в начале раздела <BODY> документа. Такой подход близок программистам, которые

обычно при написании программ разносят описательную часть программы и исполняемую, что упрощает понимание записанного кода, а иногда и определяется требованиями компилятора.

Комбинация клиентского и серверного вариантов

Допустимо использование комбинированного варианта, при котором для одного и того же изображения определены оба параметра — USEMAP и ISMAP, что предполагает использование данного изображения в качестве опорного как для клиентского, так и для серверного варианта. Параметр USEMAP является доминирующим. Это означает, что браузер, который поддерживает клиентский вариант, будет использовать USEMAP, игнорируя указания параметра ISMAP. Те браузеры, которые не поддерживают клиентский вариант и не поймут назначение параметра USEMAP, согласно общим правилам HTML проигнорируют его наличие и будут реализовывать серверный вариант, обнаружив присутствие параметра ISMAP. Комбинированный вариант является более надежным, однако требует наличия данных для конфигурации областей для обоих вариантов. В настоящее время необходимость использования комбинированного варианта все более снижается, так как все ведущие браузеры осуществляют поддержку клиентского варианта. Тем не менее, начальные странички компании Netscape, с которыми наверняка сталкивался любой пользователь браузера Netscape, сделаны именно в комбинированном варианте.

Приведем пример комбинированного варианта:

```
<A HREF="http://www.anywhere.com/testmap/logo.map">
<IMG SRC="logo.gif" USEMAP="#logo" ISMAP></A>
```

Примечание

Параметр USEMAP также является доминирующим по отношению к ссылке, определяемой тэгом <A>. Так, если изображение, используемое для реализации концепции карты-изображения в клиентском варианте, записано внутри области действия тэга <A HREF>, то ссылка, определяемая последним тэгом, будет проигнорирована браузерами, поддерживающими клиентский вариант. Пусть, например, имеется следующий фрагмент:

```
<A HREF=NoMaps.htm> <IMG SRC="example.gif" USEMAP="#map" </A>.
```

С одной стороны, все изображение является ссылкой на документ с именем NoMaps.htm. С другой стороны, наличие параметра USEMAP определяет данное изображение как опорное для соответствующей карты-изображения. Ссылка на документ NoMaps.htm будет игнорироваться уже благодаря присутствию параметра USEMAP и вне зависимости от других факторов.

Особенности использования карт-изображений

Отметим некоторые особенности в использовании карт-изображений в клиентском варианте. Когда пользователь перемещает мышь в пределах изображения, реализующего клиентский вариант, то в строке состояния в нижней части браузера Netscape отображается соответствующий URL-адрес. В серверном варианте URL-адрес не отображается, потому что эта информация размещена на сервере, которая недоступна до тех пор, пока пользователь не щелкнет мышью на изображении. Первый вариант более информативен, поскольку пользователь видит адреса ссылок, а также может определить места изображения без ссылок. В серверном варианте отображаются цифры, являющиеся относительными координатами мыши на изображении и не дающие никакой информации о ссылках и об их наличии (см. рис. 6.4).

Альтернативные средства навигации

Применение карт-изображений стало общепринятым, однако не следует забывать о том, что еще не все пользователи Web могут использовать графику или же хотят работать в режиме отключения загрузки изображений для уменьшения времени передачи файлов. Поэтому нужно предусмотреть для них какие-либо другие, альтернативные средства навигации на странице. В противном случае пользователи вообще не смогут обнаружить на странице и, соответственно, реализовать те ссылки, которые определены только картой-изображением.

В качестве альтернативного варианта можно создать отдельный раздел с текстовым описанием ссылок и соответствующих URL-адресов. Можно также создать ссылку на текстовое меню, которое имеет те же самые связи, что и карта-изображение. При любом подходе, какой бы ни был избран, нужно убедиться, что для текстового режима браузера доступны все ссылки.

Приведем пример реально существующего документа, в котором решены эти вопросы. На рис. 6.6 показан фрагмент одной из страниц известной фирмы Hewlett-Packard, в котором есть карта-изображение. В середине страницы располагается перечень десяти различных типов оборудования, для каждого из которых задана ссылка на соответствующий документ. На самом деле на странице располагается картинка, которая является опорным изображением для карты. На этом изображении выделены десять прямоугольных зон, являющимися активными областями.

Если загрузить данную страницу при отключенном режиме загрузки изображений, то вы увидите картинку, показанную на рис. 6.7. Обратите внимание, что вместо карты-изображения присутствует лишь маленькая пиктограмма, показывающая местоположение изображения и текст, который

Рис. 6.6. Фрагмент HTML-страницы, содержащей карту-изображение и дублирующее текстовое меню

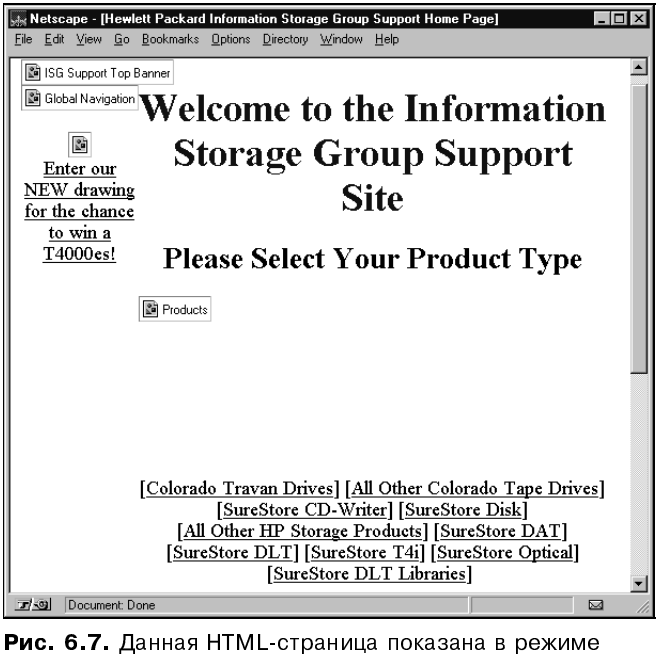


Рис. 6.7. Данная HTML-страница показана в режиме отключения загрузки изображений

был задан в качестве альтернативного для всего данного изображения. Очевидно, воспользоваться ссылками здесь не удастся. Для решения этого вопроса ниже изображения располагается обычное текстовое меню, которое полностью повторяет список, приведенный на изображении, с соответствующими ссылками. Поэтому при отключенной загрузке изображений пользователь все же сможет реализовать необходимые переходы по ссылкам, воспользовавшись дублирующим текстовым меню. В случае загрузки изображения текстовое меню лишь дублирует возможности выбора ссылок.

Средства создания карт-изображений

Создание карт-изображений требует двух шагов: подготовки опорного изображения, на котором впоследствии будут заданы активные области, и разработки файла конфигурации, описывающего геометрические параметры активных областей. Подготовка изображения, которое будет являться основой для карты-изображения, ничем не отличается от работы по подготовке обычных изображений, встраиваемых на Web-страницах. Для этого можно воспользоваться любым графическим редактором или использовать готовое изображение.

На втором шаге необходимо отметить активные области на изображении и сопоставить им соответствующие адреса ссылок. Подготовка файла конфигурации является наиболее трудным шагом в создании карт-изображений. В принципе возможен ручной способ задания границ активных областей на изображении. Например, при работе в графическом редакторе можно отмечать отдельные точки, записывать их координаты и затем создавать файл описания геометрических параметров выбранных областей. Однако такой подход крайне неудобен и громоздок.

Для автоматизации процесса разметки областей на изображении существует ряд программ, большинство из которых очень похожи друг на друга. Они позволяют создавать и изменять файлы конфигурации, работая непосредственно с изображением на экране. Большинство программ представляют собой отдельные утилиты, работающие автономно и, по существу, являющиеся дополнением к HTML-редакторам. Эти программы позволяют сохранять создаваемый конфигурационный файл либо в буфере обмена Windows, либо в файле на диске. В первом случае типичным вариантом является совместная работа программы редактирования карты-изображения и какого-либо HTML-редактора или обычного текстового редактора. Если же программа позволяет сохранять конфигурационный файл на диске, то она может использоваться полностью автономно. Все программы позволяют размечать на изображении области трех основных типов — `rect`, `circle` и `poly`. Некоторые редакторы поддерживают тип `default`. Пожалуй, единственным критерием выбора программы редактирования карт-изображений является удобство ее использования, так как по функциональным свойствам все програм-

мы очень близки. Если интерфейс программы покажется вам неудобным, можно отказаться от ее использования и выбрать другую.

Рассмотрим некоторые из существующих программ.

Программа MapEdit

Одной из наиболее простых и известных программ редактирования конфигурационных файлов является утилита MapEdit, разработанная Томасом Бутеллом (Thomas Boutell). Эта программа существует уже на протяжении нескольких лет и реализована для различных платформ. В частности, имеются версии для Windows 3.x и Windows 95/98/NT. Как и для большинства программ, существовал ряд версий данной утилиты. На текущий момент последней доступной версией для Windows 95/98/NT является версия 2.6 (сентябрь 1999 г.). Информацию о программе можно получить по адресу:

`http://www.boutell.com/mapedit/`

Программа MapEdit является условно-бесплатной (shareware) и имеет 30-дневный оценочный период, по истечении которого необходима ее регистрация. Программа невелика по размеру — дистрибутив занимает около 300 Кб, и при этом обладает практически всеми необходимыми возможностями.

Программа позволяет редактировать конфигурационные файлы как для серверного варианта (в форматах NCSA и CERN), так и для клиентского. Есть возможность визуального создания активных областей в форме прямоугольников, кругов и многоугольников, а также определения адреса ссылки для области по умолчанию.

Рассмотрим вкратце основные возможности данной программы. После запуска MapEdit выдается основное окно, содержащее заставку (рис. 6.8) и меню. Имеется возможность редактирования существующих файлов как для серверного, так и для клиентского вариантов карт-изображений. Есть возможность также создания нового файла конфигурации, однако это касается только серверного варианта. Для клиентского варианта необходимо наличие исходного HTML-файла со ссылками на встроенные изображения, которые будут использоваться в качестве опорных для карт-изображений.

Примечание

Невозможность создания нового HTML-файла с помощью программы MapEdit можно легко обойти. Для этого следует запустить программу в режиме создания файла в одном из форматов серверного варианта (NCSA и CERN), выполнить все необходимые действия, а затем сохранить полученные результаты в режиме **Save as**, указав при этом формат **Client Side Map**. Будет создан HTML-файл, который в дальнейшем можно использовать в качестве готового фрагмента HTML-документа.

Пусть нам необходимо создать новый конфигурационный файл для серверного варианта карт-изображений. Выберем пункт **Open/Create Map** из меню

File. Появится диалоговое окно (рис. 6.9), в котором следует задать имя создаваемого файла конфигурации (например, Blazons.map), указать существующий файл с изображением и формат создаваемого файла (NCSA или CERN). Файл изображения может иметь формат GIF, JPG или PNG.

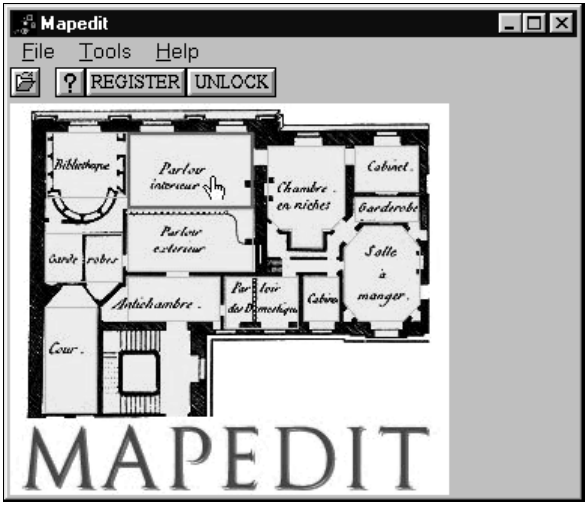


Рис. 6.8. Заставка MapEdit

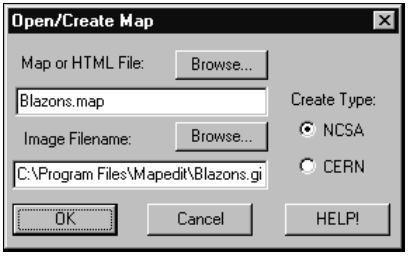


Рис. 6.9. Диалоговое окно **Open/Create Map** для создания конфигурационного файла

Примечание

Многие из программ интерпретации файлов конфигурации для серверного варианта требуют, чтобы файл имел расширение MAP. Можно рекомендовать всегда придерживаться этого правила.

Программа загрузит выбранный файл с изображением, на котором можно будет произвести разметку активных областей (рис. 6.10).

Для этого нужно выбрать форму активной области — прямоугольник, круг или прямоугольник путем нажатия соответствующей пиктограммы или выбора нужного пункта из меню **Tools** (рис. 6.11).

Дальнейшие действия производятся непосредственно на изображении путем отметки точек мышью. Для прямоугольной области отмечают левый верхний и правый нижний углы, для круговой области — центр и одну из точек

на окружности, для многоугольника задаются его вершины. Для примера на рис. 6.10 показан случай, когда на изображении уже размечены три активных области различной формы. Заметим, что линии, ограничивающие активные области, служат лишь для их визуализации при работе в редакторе и никак не изменяют файл с изображением. Изображение в данном примере по существу содержит три отдельных картинки (изображены гербы городов Томск, Якутск и Санкт-Петербург), что обычно не характерно для реалистических изображений. Однако для изображений, содержащих, например, набор кнопок управления, такая ситуация довольно типична.

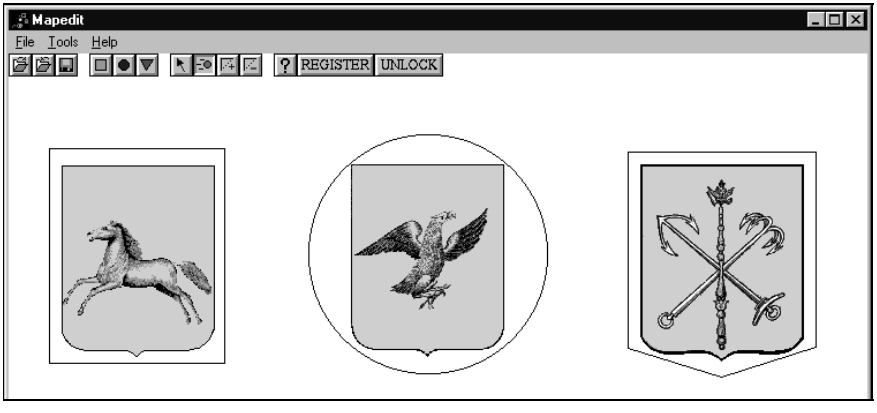


Рис. 6.10. Изображение с размеченными активными областями различного типа

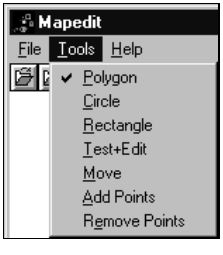


Рис. 6.11. Меню **Tools**

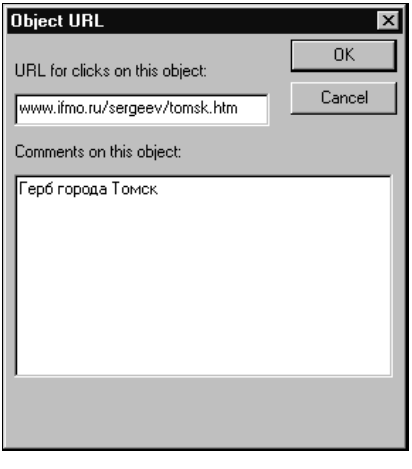


Рис. 6.12. Диалоговое окно **Object URL** для задания URL-адреса и необязательного комментария

После разметки любой из областей следует задать адрес ссылки, соответствующий данной области, а также комментирующую информацию (рис. 6.12). Можно задать адрес ссылки для области по умолчанию, который будет реа-

лизовываться для части области изображения, не входящей ни в одну из активных областей (рис. 6.13).

После разметки областей можно визуально проконтролировать или изменить созданные активные области, воспользовавшись пунктом **Test** меню **Edit**. Последним шагом работы является сохранение результатов в виде файла конфигурации (пункт **Save** меню **File**). Можно также использовать пункт **Save As**, в котором задать требуемый формат сохранения файла (рис. 6.14).

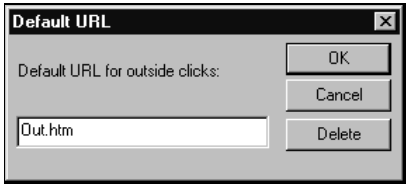


Рис. 6.13. Диалоговое окно **Default URL** для задания URL-адреса для области по умолчанию

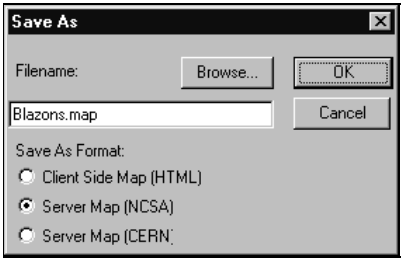


Рис. 6.14. Диалоговое окно команды **Save As**

Примечание

Старые версии редактора MapEdit содержали небольшую ошибку, связанную с заданием формата сохраняемого файла конфигурации. Если при создании файла был указан формат CERN, то при сохранении данных в режиме **Save** файл все равно будет сохранен в формате NCSA. Создать файл формата CERN удается только при использовании режима **Save as** с указанием требуемого формата.

Для рассматриваемого примера будет создан файл с именем Blazons.map, содержащий следующую информацию (формат NCSA):

```
#Герб города Томск
rect www.ifmo.ru/sergeev/tomsk.htm 35,58 187,244
#Герб города Якутск
circle www.ifmo.ru/sergeev/jakutsk.htm 364,150 468,150
#Герб города Санкт-Петербург
poly www.ifmo.ru/sergeev/Spb.htm 537,61 700,61 700,230 618,256 537,231
```

Те же данные, сохраненные редактором в формате CERN, будут выглядеть следующим образом:

```
rect (35,58) (187,244) www.ifmo.ru/sergeev/tomsk.htm
circle (364,150) 104 www.ifmo.ru/sergeev/jakutsk.htm
poly (537,61) (700,61) (700,230) (618,256) (537,231)
www.ifmo.ru/sergeev/Spb.htm
```

Заметим, что комментарии в данном формате не допускаются, поэтому при сохранении файла эта информация будет утрачена.

Рассмотрим задачу создания клиентского варианта карты-изображения. Для решения этой задачи необходимо наличие исходного HTML-файла, содержащего хотя бы одно встроенное изображение. Данный исходный файл может быть создан заранее любым текстовым редактором или специальным HTML-редактором. Пусть имеется файл с именем CSIM.HTM, содержащий следующий код:

```
<HTML>
<HEAD>
</HEAD>
 <BODY>
<IMG SRC=Blazons.gif>
</BODY>
</HTML>
```

Этот файл следует открыть в редакторе MapEdit (рис. 6.15). В отличие от варианта, в котором выполнялось создание конфигурационного файла, здесь не требуется указание имени файла с изображением в пункте меню **Open/Create Map**.

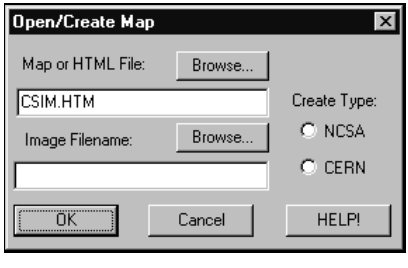


Рис. 6.15. Диалоговое окно **Open/Create Map** для открытия существующего HTML-файла

Рис. 6.16. Диалоговое окно **Select Inline Image**

Редактор после открытия исходного HTML-файла выдаст диалоговое окно с перечнем всех встроенных изображений, из которого необходимо выбрать нужное (рис. 6.16). Конечно, файл с выбранным изображением должен существовать.

Дальнейшая работа по разметке активных областей полностью идентична предыдущему случаю. После сохранения результатов разметки исходный файл будет изменен, и для приведенного примера будет иметь следующий вид:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<IMG SRC="Blazons.gif" usemap="#Blazons">
<map name="Blazons">
<area shape="rect" alt="Герб города Томск" coords="35,58,187,244"
href="tomsk.htm">
<area shape="circle" alt="Герб города Якутск" coords="364,150,104"
href="jakutsk.htm">
<area shape="poly" alt="Герб города Санкт-Петербург" co-
ords="537,61,700,61,700,230,618,256,537,231" href="Spb.htm">
<area shape="default" nohref>
</map>
</BODY>
</HTML>
```

Обратите внимание, что редактор автоматически присваивает имя для описания карты-изображения, которое совпадает с именем файла опорного изображения. Для данного примера файл с изображением имел имя Blazons.gif, поэтому параметру `name` тэга `<map>` было присвоено значение "Blazons".

Примечание

Редактор MapEdit не совсем корректно работает с символами русского алфавита. Часть русских букв при сохранении файла исчезают и на их месте оказываются пробелы. Самым простым выходом из этой ситуации является добавление русского текста после завершения работы в редакторе.

Программа Map THIS!

Еще одной утилитой создания и редактирования конфигурационных файлов карт-изображений является программа Map THIS!, информацию о которой можно получить по адресу:

`http://galadriel.ecaetc.ohio-state.edu/tc/mt/.`

Работа с данной программой по идеологии схожа с программой MapEdit. Основой работы с программой является визуальное конструирование активных областей с дальнейшим сохранением результатов в файле в одном из выбранных форматов. Редактор поддерживает оба формата серверного варианта карт-изображений (NCSA и CERN) и клиентский вариант. Изображения могут загружаться из файлов в форматах GIF и JPG.

Приведем примеры файлов конфигурации, созданных данной программой. Для примера, приведенного в предыдущем разделе, файл, сохраненный в формате NCSA, будет иметь следующий вид:

```
#$MTIMFH
#$-:Image Map file created by Map THIS!
#$-:Map THIS! free image map editor by Todd C. Wilson
#$-:Please do not edit lines starting with "#$"
#$VERSION:1.30
#$TITLE: Blazons
#$DESCRIPTION:Серверный вариант карты-изображения
#$AUTHOR:Сергеев
#$DATE:Tue Sep 14 12:10:42 1999
#$PATH:C:\Program Files\Mapthis\
#$GIF:Blazons.gif
#$FORMAT:ncsa
#$EOH
default default.htm
# Герб города Томск
rect Tomsk.htm 33,60 191,246
# Герб города Якутск
circle Jakutsk.htm 366,147 366,256
# Герб города Санкт-Петербург
poly Spb.htm 534,62 699,62 698,236 626,261 534,235 534,62
```

В отличие от программы MapEdit, данный редактор записывает в выходной файл довольно много комментирующей информации, включающей краткие сведения о самой программе, дате создания файла и др. При этом после символа #, означающего начало строки комментария, редактор добавляет символ $ для строк-комментариев, создаваемых самим редактором. Обратите внимание на четвертую строчку приведенного кода, в которой содержится просьба не редактировать комментарии, вставленные редактором.

Тот же пример, сохраненный в формате CERN, будет иметь вид:

```
rect (4096,4096) (4096,4096) mt:#$MTIMFH
rect (4096,4096) (4096,4096) mt:#$-:Image Map file created by Map THIS!
rect (4096,4096) (4096,4096) mt:#$-:Map THIS! free image map editor
by Todd C. Wilson
rect (4096,4096) (4096,4096) mt:#$-:Please do not edit lines starting
with "#$"
rect (4096,4096) (4096,4096) mt:#$VERSION:1.30
rect (4096,4096) (4096,4096) mt:#$TITLE: Blazons
rect (4096,4096) (4096,4096) mt:#$DESCRIPTION: Серверный вариант
карты-изображения
rect (4096,4096) (4096,4096) mt:#$AUTHOR:Сергеев
rect (4096,4096) (4096,4096) mt:#$DATE:Tue Sep 14 12:10:42 1999
rect (4096,4096) (4096,4096) mt:#$PATH:C:\Program Files\Mapthis\
rect (4096,4096) (4096,4096) mt:#$GIF:Blazons.gif
rect (4096,4096) (4096,4096) mt:#$FORMAT:cern
rect (4096,4096) (4096,4096) mt:#$EOH
default default.htm
```

```
rect (4096,4096) (4096,4096) mt:# Герб города Томск
rectangle (33,60) (191,246) Tomsk.htm
rect (4096,4096) (4096,4096) mt:# Герб города Якутск
circ (366,147) 109 Jakutsk.htm
rect (4096,4096) (4096,4096) mt:# Герб города Санкт-Петербург
polygon (534,62) (699,62) (698,236) (626,261) (534,235) (534,62) Spb.htm
```

Как видно из приведенного выше кода, редактор использует несколько искусственный прием для сохранения комментариев, как введенных пользователем, так и генерируемых самой программой. Напомним, что формат CERN не позволяет задавать строки комментариев, поэтому редактор создает строку вида

```
rect (4096,4096) (4096,4096),
```

в конце которой можно расположить любой текст. В сущности, такая строка описывает прямоугольник, который заведомо располагается за пределами экрана, поэтому его наличие не играет роли. Конечно, при этом текст конфигурационного файла становится гораздо более громоздким и неудобным для чтения, что, правда, не мешает работе программ. Этот прием сохранения комментариев может быть взят на вооружение.

Тот же пример, сохраненный в виде HTML-файла (для клиентского варианта карт-изображений) будет иметь вид:

```
<BODY>
<MAP NAME="Blazons">
<!-- #$-:Image Map file created by Map THIS! -->
<!-- #$-:Map THIS! free image map editor by Todd C. Wilson -->
<!-- #$-:Please do not edit lines starting with "#$" -->
<!-- #$VERSION:1.30 -->
<!-- #$DESCRIPTION:Клиентский вариант карты-изображения -->
<!-- #$AUTHOR:Сергеев -->
<!-- #$DATE:Tue Sep 14 12:10:42 1999 -->
<!-- #$PATH:C:\Program Files\Mapthis\ -->
<!-- #$GIF:Blazons.gif -->
<AREA SHAPE=RECT COORDS="33,60,191,246" HREF="Tomsk.htm"
ALT="Герб города Томск">
<AREA SHAPE=CIRCLE COORDS="366,147,109" HREF="Jakutsk.htm"
ALT="Герб города Якутск">
<AREA SHAPE=POLY COORDS="534,62,699,62,698,236,626,261,534,235,534,62"
HREF="Spb.htm" ALT="Герб города Санкт-Петербург">
<AREA SHAPE=default HREF="default.htm">
</MAP>
</BODY>
```

Здесь, в отличие от программы MapEdit, имя карты-изображения необходимо задавать вручную, поэтому оно может и не совпадать с именем файла с опорным изображением.

Программа CrossEye

Программа-редактор конфигурационных файлов CrossEye, созданная известной австралийской компанией Sausage Software. Эта программа будет встречена с радостью поклонниками популярного HTML-редактора HotDog, поскольку она создана той же компанией и имеет весьма привлекательный, выполненный с юмором, интерфейс (рис. 6.17).

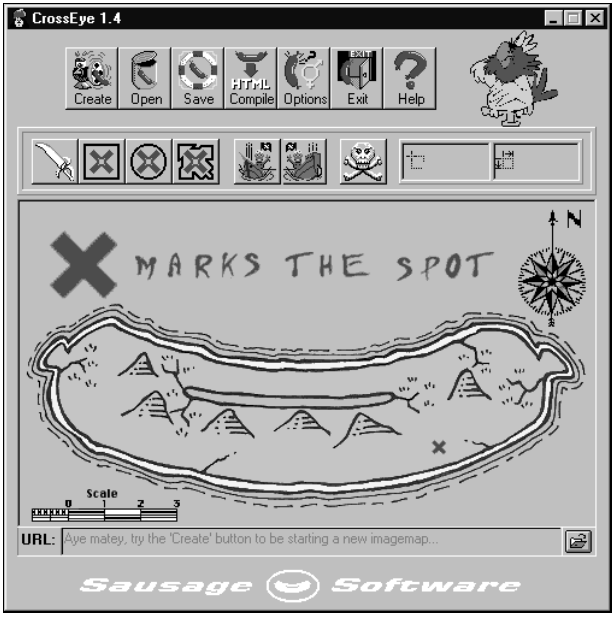


Рис. 6.17. Заставка программы CrossEye 1.4

Информация о пакете CrossEye может быть получена по адресу:

`http://www.sausage.com.au.`

Отличительными особенностями программы является довольно большой размер дистрибутива (около 2,5 Мб), а также небольшой период времени (14 дней), в течение которого можно ее эксплуатировать в режиме оценки. Большой размер программ характерен для всего программного обеспечения, создаваемого компанией Sausage Software, что, видимо, обусловлено выбором инструментария, используемым для разработки (Visual Basic).

К сожалению, программа не обладает рядом необходимых свойств. Например, отсутствует возможность чтения существующего HTML-файла, поэтому невозможно отредактировать уже имеющуюся карту-изображение, которая была создана ранее. Также невозможно выполнить сохранение результатов работы непосредственно в HTML-файле. Сохранение результатов возможно лишь в файле со специальным расширением EYE, который имеет двоичный вид и может использоваться в дальнейшем только в этом редакторе. Сгене-

рированный HTML-код записывается в буфер обмена Windows, откуда он может быть скопирован в любой текстовый редактор.

Как и во всех редакторах, описанных выше, создание и редактирование активных областей осуществляется непосредственно на изображении, однако загрузка изображения осуществляется в окно, размеры которого по непонятным причинам невозможно изменить. Если изображение больше размеров окна, то для просмотра изображения можно использовать прокрутку, но задать активную область, выходящую за пределы окна просмотра, становится невозможно.

Для клиентского варианта карты-изображения редактор вообще не предлагает задать URL-адрес для области по умолчанию. Возможно, это сделано специально, так как не все браузеры поддерживают тип области `default`. Тем не менее, редакторы, описанные выше, эту проблему решают весьма изящно, автоматически заменяя область по умолчанию на прямоугольную область с размерами, равными размерам изображения.

Недостатки в некотором смысле компенсируются отдельными дополнительными свойствами редактора. В частности, можно узнать, что попугая, который виден в правом верхнем углу приведенного рисунка, зовут Полли. Он весьма разговорчив, и пользователи, работающие на компьютере, оснащенном звуковой картой, время от времени будут слышать возгласы попугая, которые, правда, никак не связаны с выполняемыми действиями. А в одном из диалоговых окон настройки редактора есть даже специальный пункт, позволяющий заткнуть рот бедному попугаю. Вот пример сервиса высшего разряда. Видимо, благодаря перечисленным свойствам, рейтинг этого редактора по оценкам http://www.tucows.com, весьма высок, чего нельзя сказать о двух описанных выше программах.

В конечном итоге выбор редактора для создания карт-изображений остается за пользователем.

Заключительный пример

В завершение главы приведем пример довольно удачного изображения, которое используется в одном из реально существующих программных продуктов в качестве графического меню (рис. 6.18). Этот пример не имеет никакого отношения к Web-страницам, однако вполне мог бы играть роль карты-изображения для реализации ссылок. На этой картинке практически все предметы, не говоря уже о непосредственно написанных в различных местах словах, являются областями, реализующими те или иные действия. Так, щелкая мышью на расположенных на столе аптекарских весах, вы получаете доступ к процедуре подведения баланса, наведя мышь на фотоаппарат, получите мгновенный срез экономической ситуации. Скульптура мыслителя помогает получить справку Help, плакаты на стенах реализуют соответст-

вующую функцию и т. д. Реализация выхода из программы видна из рисунка — курсор мыши расположен у выхода. Конечно, не всегда можно догадаться обо всех возможностях, таящихся в этом изображении, однако данное представление является лишь дополнением к меню, выполненном в стандартном виде.

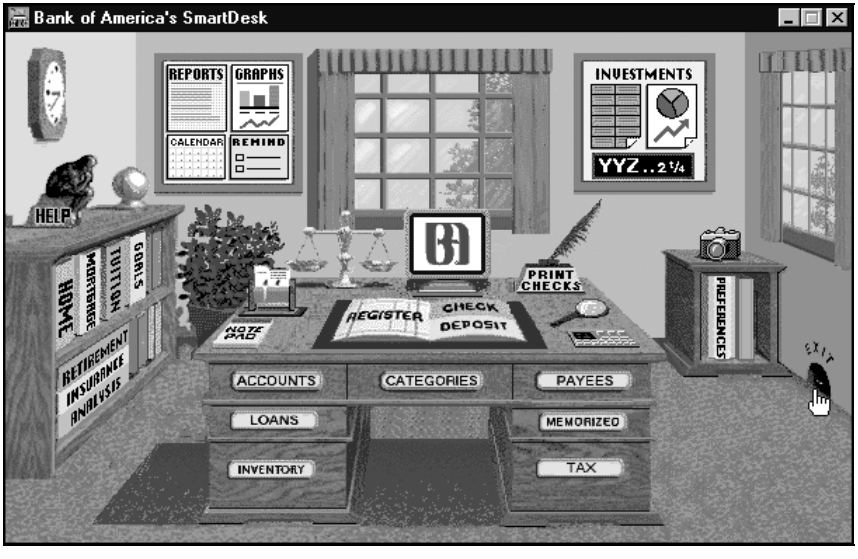


Рис. 6.18. Пример изображения, играющего роль живого графического меню

Реализовать меню такого типа для HTML-документа, пользуясь концепцией карты-изображения, очень просто. Основная сложность будет состоять в подготовке изображения, а назначение активных областей при использовании специальных редакторов становится делом техники.

ГЛАВА 7

Звук

В данной главе вы ознакомитесь с основными принципами использования звука в компьютерах, необходимым программным и аппаратным обеспечением для воспроизведения звука. Узнаете, как настроить браузеры для приема и воспроизведения звука из сети Интернет.

Будут рассмотрены подключаемые программные модули к браузерам, а также отдельные приложения, позволяющие воспроизводить все типы звуковых файлов. Вы узнаете об особенностях встраивания в HTML-документ звуковых файлов и использовании подключаемых модулей.

Средства воспроизведения звука

Еще совсем недавно единственным способом извлечения звуков при помощи компьютера было использование встроенного динамика, традиционно существовавшим на всех моделях персональных компьютеров. Хотя главным предназначением этого динамика была выдача коротких по длительности, чаще всего однотонных звуков, находились программисты, создававшие утилиты для воспроизведения целых музыкальных фрагментов.

В наше время, когда термин "мультимедийный компьютер" стал привычным и уже не вызывает благоговейного трепета, большинство персональных компьютеров оснащаются звуковой картой и приводом для чтения компакт-дисков. Тем не менее, маленький динамик по-прежнему присутствует в каждом компьютере. За прошедшие годы компьютеры очень сильно изменились, в частности, отпала необходимость в некоторых устройствах и элементах конструкции. Так, например, современные компьютеры более не оснащаются кнопкой переключения режимов "Turbo", исчез замок для блокирования клавиатуры. Зато встроенный динамик все так же выполняет возложенные на него задачи.

Не будет преувеличением сказать, что функционирование компьютера начинается со звука. Многие компьютеры при загрузке издают звуковой сигнал, по характеру которого можно определить правильность прохождения тестирования. Опытные пользователи даже при выключенном мониторе на слух могут определить корректность процесса запуска компьютера. Более

того, до сих пор ряд программ используют возможности встроенного динамика для информирования пользователя, например, о завершении очередной фазы какого-либо процесса. Смысл этих рассуждений сводится к тому, что звуковая информация играла и играет весьма значительную роль в работе с компьютером.

Для получения полноценного звука из компьютера нужна звуковая карта и колонки или наушники. За последние годы цена мультимедийных компонентов компьютера по отношению к общей цене значительно снизилась, что говорит о целесообразности их установки на компьютер. На сегодняшний день 8-битовые монофонические звуковые карты ушли в прошлое. Даже самые дешевые из них обеспечивают стереозвучание и работают в 16-битовом режиме. Установка звуковой карты вкупе с приводом чтения компакт-дисков превращает компьютер в музыкальный центр неплохого качества, а также обеспечивает возможность работы с мультимедийными программами. Вся информация, приводимая далее в этой главе, актуальна только для тех пользователей, компьютер которых оснащен звуковой картой.

Как компьютер работает со звуком

Существует два типа звуковых файлов, используемых в компьютерах: цифровые и музыкальные. Иногда выделяют еще один тип файлов — текстово-звуковые, в которых используется метод конвертирования текстового файла в звуки, близкие к человеческой речи, т. е. замена букв фонемами.

Музыкальные файлы похожи на нотные листы, — мелодия в них хранится последовательностью музыкальных нот, и, кроме того, в них имеется информация об используемых для воспроизведения музыкальных инструментах. При проигрывании такой музыки звуковой адаптер может не просто генерировать звуки разной высоты, но и выбирать из памяти фрагменты звучаний реальных музыкальных инструментов и синтезировать мелодию. Использование таких звуков ограничено возможностями синтезатора.

Большинство звуков, которых вы слышите из колонок компьютера, представляют собой оцифрованный звук. По своей сути оцифрованный звук довольно прост. Аналоговые по своей природе звуки, передаваемые по проводам в виде изменяющегося напряжения, преобразуются в цифровые данные, которые можно сохранять в компьютере в виде файлов. Такое преобразование выполняется при помощи аналого-цифрового преобразователя (АЦП). При воспроизведении выполняется обратное действие — преобразование из цифровой формы в аналоговую. Эта работа возлагается на звуковую карту компьютера. Пользователь может создать любое количество звуковых файлов, подключив к компьютеру микрофон или бытовой магнитофон и воспользовавшись стандартными программами записи звука.

На качество получаемого оцифрованного звука влияет несколько факторов. Важнейшим фактором является частота дискретизации звука (sampling rate)

или количество преобразований аналогового звука в цифровой (замеров), выполняемое за одну секунду. Чем больше частота дискретизации, тем выше качество звука и тем больше при воспроизведении он будет похож на оригинал. Обычно этот параметр лежит в диапазоне от 8 до 44,1 кГц. Увеличение частоты дискретизации приводит к пропорциональному росту размеров файла, поэтому здесь важен разумный компромисс.

Другим фактором является разрядность дискретизации (sample rate), определяющая точность замеров. Типичными значениями являются 16 бит, 8 бит и др.

Характеристикой звукового файла является также режим записи — моно или стерео. Выбор того или иного режима сразу же в два раза изменяет размер получаемого файла.

Нетрудно подсчитать, сколько места на диске будут занимать оцифрованные звуки. Так, одна секунда фонограммы качества аудио CD (44100 замеров в секунду, 16 бит на канал, два канала) дает 172 Кб данных. Заметим, что полученная величина близка к скорости передачи данных односкоростных CD-приводов, большинство которых уже умели проигрывать аудиодиски. Сегодня скорости этих устройств значительно возросли, поэтому, тем более, проблем воспроизведения музыки не должно возникать.

Выбор приведенных параметров оцифровки звука не случаен. Оказывается, что определить влияние факторов на качество звука довольно просто. Частота дискретизации определяет максимально возможную частоту воспроизведения звука. В соответствии с законом Найквиста, самая высокая частота воспроизведения равна половине частоты дискретизации. Так, при дискретизации 44,1 кГц звук можно воспроизводить с частотой 22,05 кГц. Такая частота с запасом перекрывает диапазон звуков, воспринимаемых большинством людей, и определяет практически идеальное качество. В то же время для передачи речи такой частоты, как правило, не требуется. Человеческая речь в основном укладывается в диапазон 3 кГц, поэтому в телефонных сетях оцифровка звука ведется на частоте 8 кГц.

Разрядность дискретизации определяет фактор, представляющий собой отношение сигнала к шуму. Любой метод хранения и последующего воспроизведения звука вносит некоторые случайные потери, которые проявляются в виде шумов. Для оцифрованного звука шум определяется точностью замеров. Чем точнее измерение, тем меньше потенциальная возможность внесения шума.

Еще одним фактором является наличие сжатия данных, целью которого является экономия места на диске и выигрыш во времени передачи файлов. Среди схем сжатия/хранения звуковых данных можно выделить схемы, обеспечивающие сжатие без потерь информации, а также схемы, позволяющие значительно (в 10 и более раз) уменьшить размеры файлов без заметного искажения звука. Подробное описание технологий хранения данных выходит за рамки этой книги, однако в конце главы все же будут даны не-

которые практические рекомендации по работе с современными форматами звуковых данных, что вызвано необычайным ростом их популярности.

Модуль LiveAudio

Первые версии браузеров не имели встроенных средств для воспроизведения звука и, соответственно, на Web-страницах практически не встречалось звуковых файлов. Если же такие ссылки попадались, то сначала требовалось переписать файл на локальный диск, а затем воспользоваться внешней программой его воспроизведения. Технология подключаемых модулей изменила ситуацию. Теперь для браузеров имеется ряд подключаемых модулей, позволяющих воспроизводить звуковые файлы в различных форматах. Преимуществом использования подключаемых модулей по сравнению с внешними приложениями является возможность управления проигрыванием звуковых файлов прямо из окна браузера.

Браузер Netscape поставляется вместе с подключаемым модулем LiveAudio, который часто называют "официальным" аудиоплейером Netscape. Модуль LiveAudio может работать со стандартными форматами AIFF, AU, MIDI и WAV. Этот модуль снабжен понятной и удобной консолью с кнопками воспроизведения, паузы, остановки и регулятором громкости.

Модуль LiveAudio заменил вспомогательное приложение NAPlayer, сопровождавшего более ранние версии браузера Netscape Navigator. (Модуль LiveAudio начал поставляться с Netscape Navigator версии 3.0.)

Приложение NAPlayer работало с форматами Sun/NeXT (AU и SND) и Mac/SGI (AIF и AIFF). LiveAudio имеет более широкие возможности и поддерживает следующие типы звуковых форматов:

- AIFF — формат звуковых файлов Mac/SGI
- AU — формат звуковых файлов Sun/NeXT
- MIDI— формат электронных музыкальных инструментов (Musical Instrument Digital Interface)
- WAV — стандарт звуковых файлов операционной системы Windows

Формат AU когда-то был стандартным для Интернета. Формат AIFF — стандарт для платформы Macintosh, формат WAV — наиболее типичен в операционной системе Windows. Таким образом, этот подключаемый модуль может воспроизводить большинство стандартных оцифрованных звуковых файлов, а также работать с музыкальным форматом MIDI.

Управление модулем LiveAudio

Если на Web-странице, загружаемой браузером Netscape, встречается звуковой файл одного из перечисленных выше форматов, модуль LiveAudio выво-

дит в окне браузера консоль управления. Вид консоли управления определяется параметрами, указанными в тэге включения звукового файла в страницу.

Модуль может работать как со встроенными (с помощью тэга <EMBED>), так и отдельными звуковыми файлами (например, можно создать ссылку на звуковой файл). В последнем случае консоль управления выводится в пустом окне браузера, при этом сама консоль будет иметь стандартный вид.

Органы управления модулем понятны и просты в использовании. Кнопки такие же, как и у обычного аудиоплейера — останов, воспроизведение, пауза и регулятор громкости, имеется простое выпадающее меню (рис. 7.1).

Рис. 7.1. Консоль управления модуля LiveAudio

Щелчком мыши слева или справа от ручки регулятора громкости можно уменьшить или увеличить громкость воспроизведения. Линейка "светодиодов" внизу показывает текущий уровень звука. Темно-зеленые "светодиоды" индицируют диапазон 0—40%, светло-зеленые — диапазон 40—100%.

Если на консоли щелкнуть правой кнопкой мыши, будет выведено выпадающее меню. Его пункты дублируют кнопки управления, а также обеспечивают возможность сохранения звукового файла на локальном диске (**Save As...**) и содержат сведения об изготовителе модуля (пункт меню **About**).

Примечание

Пункт меню сохранения файла (**Save As...**) есть только в последней версии модуля LiveAudio (версия 1.1). В предыдущей версии этот пункт отсутствовал, зато отображалась информация о текущей громкости звука (например, появлялась надпись **Current Volume: 60.0%**).

Встраивание звуковых файлов в Web-страницу

Для использования звуковых файлов на Web-странице можно применить любой из двух вариантов: встраивать файл с помощью тэга <EMBED>, либо создать ссылку на звуковой файл. В любом случае фонограмму можно будет прослушать при просмотре страницы в браузере Netscape при помощи подключаемого модуля LiveAudio. Первый вариант дает большую гибкость в компоновке страницы. Как будет описано ниже, тэг <EMBED> обладает значительным количеством параметров, задавая которые можно контролировать

аспекты работы модуля LiveAudio. Однако у этих возможностей есть и оборотная сторона. Если страница, рассчитанная на использование модуля LiveAudio, будет просматриваться другим браузером, который, вероятнее всего, будет иметь свои средства для воспроизведения звука, то все уникальные для модуля настройки не будут работать. Использование же ссылки на файл является стандартным способом, применяемым для любых типов файлов.

В примере 7.1 приведен HTML-документ, в котором имеются встроенные файлы, а также есть ссылка на звуковой файл.

Пример 7.1. Пример документа, использующего звуковые фрагменты

```
<HTML>
<HEAD>
<TITLE>Пример использования звуковых файлов</TITLE>
</HEAD>

<BODY>
Встраивание звукового файла
с <U>полным</U> набором органов управления консоли
<P>
<EMBED SRC="example1.wav" HEIGHT=60 WIDTH=144 AUTOSTART=TRUE>

<P>
Встраивание звукового файла
с <U>ограниченным</U> набором органов управления консоли
<P>
<EMBED SRC="example2.wav" HEIGHT=15 WIDTH=144 CONTROLS=SMALLCONSOLE>

<P>
<A HREF="example.mid">Это ссылка на звуковой файл</A>

</BODY>
</HTML>
```

В документе, приведенном в примере 7.1, используются два встроенных звуковых файла (example1.wav и example2.wav), а также есть ссылка на звуковой файл example.mid. При просмотре этого документа в браузере Netscape для каждого из встроенных файлов будет отображена консоль управления модуля LiveAudio (рис. 7.2).

Для файла example1.wav консоль управления будет иметь полный набор кнопок управления (этот режим используется по умолчанию), причем воспроизведение начнется автоматически при загрузке документа (`AUTOSTART=TRUE`). Окно консоли управления будет иметь размеры 144 на 60 пикселов.

Для звукового файла example2.wav консоль управления будет иметь более компактные размеры (144 на 15 пикселов) и урезанный набор органов управ-

ления, что определено инструкцией `CONTROLS=SMALLCONSOLE`. Воспроизведение этого файла начнется только в случае, если пользователь нажмет соответствующую кнопку.

Примечание

Размеры консоли органов управления модуля LiveAudio имеют постоянные значения и не могут изменяться (например, при полном наборе органов управления — 144 на 60 пикселов). Значения размеров окна, отводимых под встраиваемый элемент параметрами `HEIGHT` и `WIDTH`, не должны быть меньше требуемых. Если же этими параметрами заданы размеры больше требуемых, то справа и снизу от консоли управления будет оставлено соответствующее свободное место.

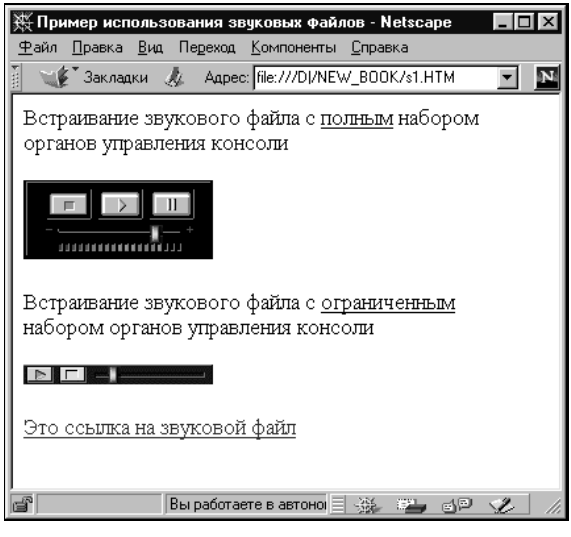


Рис. 7.2. Отображение документа со встроенными звуковыми файлами в браузере Netscape

На третий звуковой файл, используемый в документе (example.mid), дана стандартная ссылка. Реализация этой ссылки откроет новое окно браузера Netscape с полной консолью управления и автоматически запустит файл на воспроизведение (рис. 7.1).

В табл. 7.1 перечислены параметры тэга `<EMBED>` и их значения, использующиеся совместно с модулем LiveAudio. Все параметры необязательные, за исключением `SRC`, `WIDTH` и `HEIGHT`, применяющихся при встраивании в HTML-документ большинства объектов.

Таблица 7.1. *Параметры тэга* `<EMBED>`

Параметр	Описание
`SRC="filename"`	Указывает имя воспроизводимого файла (форматы AU, AIFF, AIF, WAV, MIDI и MID). Обязательный параметр

Таблица 7.1 *(окончание)*

Параметр	Описание
WIDTH=n	Определяет ширину консоли в пикселах. Обязательный параметр
HEIGHT=m	Определяет высоту консоли в пикселах. Обязательный параметр
AUTOSTART=TRUE|FALSE	Если имеет значение TRUE, воспроизведение начинается автоматически. По умолчанию — значение FALSE
AUTOLOAD=TRUE|FALSE	Если имеет значение FALSE, файл не загружается автоматически. По умолчанию — значение TRUE
STARTTIME="mm:ss"	Воспроизведение начинается с указанного в минутах и секундах момента от начала файла. По умолчанию — 00:00
ENDTIME="mm:ss"	Воспроизведение оканчивается с указанного в минутах и секундах момента от начала файла. По умолчанию — это конец файла
VOLUME=percentage	Громкость воспроизведения, указанная в процентах от максимальной. По умолчанию — последнее установленное значение
ALIGN="value"	Выравнивает консоль управления по отношению к тексту страницы. Может иметь значения CENTER, BASELINE, TOP, LEFT, RIGHT. По умолчанию — BASELINE
CONTROLS="value"	Задает набор органов управления консоли. Может иметь следующие значения (они объяснены далее в этой таблице): CONSOLE, SMALLCONSOLE, PLAYBUTTON, PAUSEBUTTON, STOPBUTTON и VOLUMELEVER. По умолчанию — CONSOLE
CONSOLE	Полный набор органов управления
SMALLCONSOLE	Компактное представление консоли. Полный набор органов управления, кроме кнопки паузы
PLAYBUTTON	Только кнопка воспроизведения
PAUSEBUTTON	Только кнопка паузы
STOPBUTTON	Только кнопка остановки. При этом файл выгружается
VOLUMELEVER	Только регулятор громкости
CONSOLE="name"	Комбинация органов управления, позволяющая включить в страницу несколько клипов. Например, можно указать CONSOLE="MySetup" в двух тэгах <EMBED> одной страницы; при этом в обоих случаях вид консоли будет одинаковым

Примечание

Если для клипа задать параметры `CONTROLS="VOLUMELEVER"` и `CONSOLE="_MASTERVOLUME"`, то изменение громкости звучания данного клипа будет отражаться на громкости всех остальных клипов, т. е. изменять громкость работы звуковой карты.

Отображение документа, рассчитанного на определенный браузер, с помощью другого браузера может нарушить весь замысел автора. Посмотрим, как пример 7.2, будет выглядеть в браузере Microsoft Internet Explorer (рис. 7.3). Здесь для проигрывания встроенных звуковых файлов используется вспомогательное приложение Microsoft ActiveMovie, работающее как компонент браузера. Естественно, что это приложение не распознает большинства специфических параметров тэга `<EMBED>`, рассчитанных на использование модуля LiveAudio браузера Netscape. В итоге, хотя оказывается вполне возможным прослушать все имеющиеся на странице звуковые файлы, внешний вид документа оставляет желать лучшего.

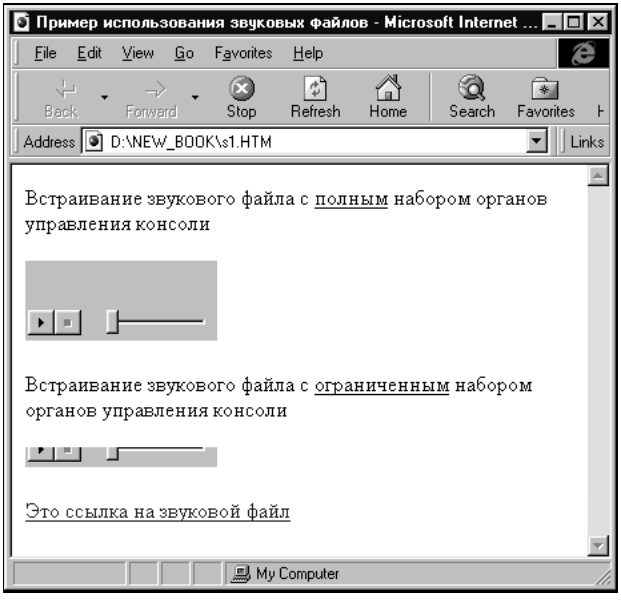


Рис. 7.3. Отображение документа со встроенными звуковыми файлами в браузере Microsoft Internet Explorer

Возможна и обратная ситуация. Документ, подготовленный с учетом требований браузера Internet Explorer, может неудачно отображаться браузером Netscape.

Заметим, что приложение ActiveMovie предоставляет больше возможностей по управлению прослушиванием (рис. 7.4). Так, например, можно задать режим повторения фонограммы заранее заданное или бесконечное число раз. Есть возможность одновременного исполнения (с наложением звука) нескольких файлов встроенных на страницу, даже если они имеют одина-

ковый тип. Например, можно заставить играть два файла типа WAV одновременно, соорудив из браузера своего рода микшерский пульт. Этой возможности лишен модуль LiveAudio, в котором запуск одной фонограммы автоматически останавливает воспроизведение файла такого же типа. Исключением является возможность одновременного воспроизведения файла с оцифрованной музыкой и файла типа MIDI, что возможно всегда.

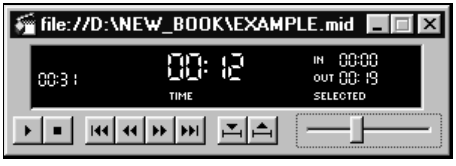


Рис. 7.4. Консоль управления приложения ActiveMovie с полным набором кнопок

Если вам потребуется сконфигурировать программу-сервер для работы со звуковыми файлами, то нужно сначала задать соответствующие MIME-типы. В табл. 7.2 перечислены MIME-типы и соответствующие им типы файлов, с которыми работает модуль LiveAudio.

Таблица 7.2. *MIME-типы и соответствующие им файлы*

MIME-тип	Расширение
audio/basic	AU
audio/x-aiff	AIF, AIFF
audio/aiff	AIF, AIFF
audio/x-wav	WAV
audio/wav	WAV
audio/x-midi	MID, MIDI
audio/midi	MID, MIDI

Другие звуковые модули

Кроме модуля LiveAudio, поставляемого с браузером Netscape, имеется довольно много других звуковых модулей, часть которых работает со своим, уникальным форматом хранения звука. Приведем краткую характеристику некоторых из них.

В ряде случаев нет необходимости сохранять оцифрованный звук с параметрами, обеспечивающими качество звучания аудио CD. Напомним, что такое качество обеспечивается при частоте дискретизации 44,1 кГц, разрядностью 16 бит, стерео. Нетрудно подсчитать, что минута звучания потребует хранения около 10 Мб данных (44 100×2 байта×2×60 = 10 584 000 байт). Записи та-

кого качества сохраняют все оттенки звука, в частности, дают возможность воспроизведения всех высоких частот, слышимых человеческих ухом. Однако, для многих применений, например для передачи речевых сообщений, такое качество излишне. Для звуковых данных такого рода можно понизить объем хранимой информации, уменьшив характеристики оцифровки звука. Минимально возможными параметрами для хранения звука в стандартном формате WAV являются частота оцифровки 8 кГц, разрядность данных 8 бит, моно.

Примечание

Речь идет о хранении в подформате PCM, (Pulse Code Modulation, импульсно-кодовая модуляция), использующегося в большинстве случаев применения формата WAV. Этот подформат не предусматривает возможностей сжатия данных.

При таких параметрах минута звучания уже потребует всего лишь около 0,5 Мб (8000×1×60 = 480000 байт). Выигрыш в 20 раз. Тем не менее, даже для такого звукового сообщения необходим поток 64 Кбит/с, что не дает возможности передавать данные по большинству модемов в реальном времени.

Для решения этой задачи рядом компаний были разработаны модули, которые могут сохранять и воспроизводить звуковые сообщения со значительной степенью сжатия. Для речевых сообщений важнейшим фактором является разборчивость речи, которая сохраняется даже при сильном сжатии.

К таким модулям относятся TrueSpeech, ToolVox, EchoSpeech и др.

Модуль TrueSpeech компании DSP Group хранит звуки в своем собственном формате, сжимая данные до 60 Кб на одну минуту звучания (передача таких данных требует 8 Кбит/с). Поскольку в состав операционной системы Windows входит кодер/декодер для данного формата, то конвертировать оцифрованный звук в него можно при помощи стандартного приложения Фонограф (Sound Recorder). Более подробную информацию можно получить на сайте компании DSP Group: **http://www.dspg.com/**.

Модуль ToolVox компании Voxware обеспечивает еще большую степень сжатия (до 53:1). После конвертирования речевого файла в формат VOX скорость потока данных сокращается до 2,4 Кбит/с. Для кодирования/декодирования данных используется кодировщик RT24, лицензию на использование которого получила компания Netscape. Появился также более новый кодер RT29, сокращающий поток данных до 2,9 Кбит/с. Компания Voxware ведет разработки программ для конвертирования текста в звуковые файлы и программ распознавания речи. Информацию о программном обеспечении, разрабатываемом компанией Voxware, можно получить на сайте: **http://www.voxware.com/**.

Модуль EchoSpeech выполняет сжатие речи на частоте дискретизации 11025 Гц. Степень сжатия достигает значения 18.5:1, что уменьшает поток данных до

9,6 Кбит/с. Для создания файлов формата EchoSpeech нужно воспользоваться специальной программой-кодером Speech Coder. Программное обеспечение работы с файлами этого формата имеется по адресу **http://www.echospeech.com/**.

Примечание

Не стоит обольщаться, узнав о возможностях сжатия звука перечисленными выше программами. Запись любого музыкального произведения, сохраненного в этих форматах, будет звучать как со старой, заезженной пластинки.

Существуют модули и для воспроизведения музыки в формате MIDI. Одним из наиболее известных среди них является модуль Crescendo компании LiveUpdate. При помощи браузера, оснащенного таким модулем, можно создавать Web-страницы с фоновыми музыкальными клипами и специальными звуковыми эффектами.

Модуль запускается автоматически и работает без контрольной панели на экране. Модуль выпускается для систем Windows 95/NT и Macintosh и может быть скопирован со страницы **http://www.liveupdate.com/midi.html**.

Усовершенствованная версия модуля, Crescendo Plus, имеет панель управления и поддерживает работу в реальном режиме времени, т. е. чтобы начать

Рис. 7.5. Web-страница, содержащая MIDI-файлы знаменитых рок-групп

прослушивание, вам не надо ждать окончания загрузки звукового файла. Эту версию также можно найти на сайте компании LiveUpdate.

Формат электронных музыкальных инструментов MIDI вовсе не потерял своей популярности в связи с бурным развитием новых форматов хранения оцифрованного звука. До сих пор на Web-страницах встречается довольно много ссылок на файлы этого формата. Например, на сервере, посвященном трехсотлетию Санкт-Петербурга (1703—2003), можно услышать гимн города, хранящегося в формате MIDI. Кстати, этот сервер обещает быть очень интересным, его концепция, структура и наполнение определяются специальным наблюдательным советом, председателем которого является писатель Даниил Гранин. Содержание сервера (**http://www.300.spb.ru**) планируется обновлять вплоть до 2003 года.

Встречаются целые сайты, посвященные музыкальным произведениям в формате MIDI (например, **http://www.midi.ru**). Множество страниц в Интернете посвящено дискографии различных музыкальных групп (например, **http://rem.ifmo.ru/midi/**, страничка этого сайта показана на рис. 7.5).

Технология RealAudio

С развитием возможностей Интернета актуальной стала задача проведения различных аудио- и видеоконференций, передачи вещания радиостанций в реальном времени. Технология RealAudio стала первой, предоставившей возможность передачи звуковых файлов в режиме реального времени. На сегодняшний день эта технология получила в Интернете широкое распространение, модуль RealAudio, несомненно, является самым популярным в Интернете приложением для вывода звуковых файлов. С сайта компании, разработавшей этот модуль, было скопировано несколько миллионов экземпляров плейера. В начале 1999 г. промелькнуло сообщение, что число зарегистрированных пользователей этой программы достигло 50 Мбиллионов.

Разработка технологии RealAudio была выполнена компанией Progressive Networks (сейчас она называется RealNetworks). Компания в сети Интернет имеет свой сайт, популярность которого за последние годы необычайно увеличилась. Адрес сайта **http://www.real.com**.

Примечание

До недавнего времени сайт компании-разработчика технологии RealAudio имел адрес **www.realaudio.com**. Сейчас обращение по такому адресу приведет к странице **www.real.com**.

RealAudio состоит из трех функционально связанных программ. Программа-кодировщик (RealProducer Pro) конвертирует готовые звуковые файлы или живой звук в файлы формата RealAudio. Программа-сервер (RealServer) передает эти файлы по сети. И, наконец, программа-плейер (RealPlayer G2)

проигрывает файлы по мере их поступления на компьютер пользователя. Все три компонента доступны на сайте: **http://www.real.com** (рис. 7.6). Сайт содержит много полезной технической информации и звуковых примеров.

Рис. 7.6. Сайт компании RealNetworks, посвященный технологии RealAudio

Для прослушивания звуковых клипов достаточно иметь только программу-плейер RealPlayer. В случае если вы также планируете создавать свои собственные звуковые файлы и размещать их на сервере, необходимо иметь программу-кодировщик и программу-сервер.

Примечание

Ранние версии программы-плейера назывались RealAudio Player. Сегодня последняя версия программы называется RealPlayer G2. Эта программа является свободно распространяемой. Существует также коммерческая версия, имеющая название RealPlayer Plus G2. Коммерческая версия обладает несколько большими возможностями по сравнению с бесплатным вариантом. Различия в возможностях описаны в справочном файле, поставляемом вместе с программой.

Исходными данными для кодирования в формат RealAudio являются готовые звуковые файлы, либо "живой" звук. Программа-кодировщик может

создавать файлы, оптимизированные для модемов с различными скоростями. Типичными вариантами являются скорости 14,4 и 28,8 Кбит/с. В результате кодирования получается файл, размеры которого намного меньше размера исходного файла. Например, одноминутный файл формата WAV (частота 22 кГц, разрядность данных 16 бит, моно) занимает 2,6 Мб; кодированный файл формата RealAudio для скорости 14,4 составляет всего 60 Кб, а для скорости 28,8 — в два раза больше. Таким образом, достигается сжатие — 40:1 и 20:1 соответственно. Файл для скорости 14,4 по качеству сравним с AM-вещанием и очень хорош для речи, а файл для скорости 28,8 звучит как монофоническая FM-передача и подходит практически для любой музыки.

Процедура сжатия выполняет отбрасывание некоторой звуковой информации. Приходится пожертвовать качеством, чтобы получить выигрыш в скорости передачи. Это проявляется при прослушивании некоторой потерей глубины звучания.

В Интернете информация может передаваться двумя способами. Одним вариантом является протокол TCP, лежащий в основе WWW, который подразумевает безошибочную передачу данных. Любая возникающая при передаче ошибка тут же исправляется повторной передачей. С другой стороны, существует протокол UDP, который подразумевает скоростную передачу с некоторыми потерями в точности. Наличие ошибок недопустимо, когда мы имеем дело с текстом или программным кодом, где важен каждый бит. Некоторые ошибки и неточности передачи звука воспринимаются нашим ухом как небольшой шум или статические разряды в процессе обычного радиоприема. При передаче звука более важна непрерывность передачи, здесь главное — не потерять целые блоки данных и поддерживать передачу в режиме реального времени. Поэтому сервер RealAudio использует протокол UDP. Возможен случайный пропуск того или другого звукового байта, зато звук передается непрерывно и в режиме реального времени.

Примечание

На самом деле в противоречии с названием технологии RealAudio передача радиопрограмм идет не совсем в реальном времени. Вещание различных серверов RealAudio обычно идет с некоторой задержкой, измеряемой минутами. С коллегой автора этих строк однажды приключилось следующее. Он зашел в комнату, где работало несколько компьютеров, на одном из которых шло вещание программы RealAudio. Связь была достаточно хорошая, что обеспечивало безупречную передачу звуковой программы. Поэтому определить на слух, идет ли настоящая радиопередача, пойманная из эфира радиоприемником, или же это программа RealAudio, не представлялось возможным. В некий момент в передаче прозвучали привычные сигналы точного времени. Как оказалось впоследствии, коллега, нисколько не сомневаясь в их правильности, автоматически взглянул на часы и, несколько удивившись их спешке, поставил точное время. В этот же день он чуть было не опоздал на поезд, так как программа RealAudio шла с почти десятиминутным запозданием.

Сервер RealAudio является двухканальной системой: протокол UDP используется для передачи данных, а протокол TCP — для согласования с системой пользователя скорости передачи и трансляции команд.

Программа-плейер RealPlayer Plus G2

Программа воспроизведения клипов формата RealAudio выпускается для систем Windows, Macintosh и UNIX. Программа для Windows называется RealPlayer G2 и является свободно распространяемой. Коммерческая версия имеет название RealPlayer Plus G2. Установка любого варианта программы на компьютер не составляет труда. Программа поставляется в виде самораспаковывающегося файла, запустив который и, следуя дальнейшим указаниям, можно осуществить установку. В состав устанавливаемого пакета входит как самостоятельная программа-плейер, так и подключаемые модули для браузеров. После завершения установки браузеры Netscape и Internet Explorer получат возможность воспроизводить клипы формата RealAudio.

Как самостоятельная программа RealPlayer Plus G2 (рис. 7.7) является полнофункциональным приложением, позволяющим воспроизводить звуковые файлы от любого источника, включая локальный диск или удаленный сервер. Эта программа является одним из представителей растущего числа приложений, способных напрямую, без участия браузера, общаться с Интернетом. Программа-плейер RealPlayer Plus G2 как отдельная программа может служить вспомогательным приложением браузера или самостоятельно воспроизводить звуковые файлы.

Если в Web-страницу включается ссылка на файл формата RealAudio, то при щелчке мышью на ней браузер запустит программу-плейер как вспомогательное приложение. Это приложение работает как отдельная задача и после окончания прослушивания требует ручного закрытия.

Если же файл встроен непосредственно в страницу с помощью тэга `<EMBED>`, браузер автоматически отобразит на экране консоль управления подключаемого модуля.

Органы управления плейера очень просты. Панель управления содержит типичный набор кнопок: **Воспроизведение**, **Пауза**, **Останов**, **Запись** (доступна только для коммерческой версии) и индикатор текущего момента проигрывания с бегунком. Имеется также кнопка включения эквалайзера (только для коммерческой версии) и кнопка переключения внешнего вида плейера (полный/компактный). Может выводиться информация о проигрываемом клипе, время воспроизведения и длительность клипа. Имеется движковый регулятор громкости. Конкретный внешний вид плейера в значительной степени зависит от установок, выбранных пользователем в пункте меню **View**. Заметим, что внешний вид, а также набор пунктов меню сильно изменились по сравнению с более ранними версиями программы, поэтому их детальное описание представляется излишним.

Рис. 7.7. Программа-плейер RealPlayer Plus G2

Если звуковой файл встраивается в страницу, то набор органов управления определяется автором страницы и зависит от значений параметров соответствующего тэга `<EMBED>`. Правила записи параметров рассматриваются ниже.

Встраивание в страницу звуковых файлов формата RealAudio

Звуковые файлы, используемые технологией RealAudio, должны иметь расширение RA. Кроме того, принято использовать метафайлы с расширением RAM, которые представляют собой обычные текстовые файлы, каждая строчка которых содержит полный URL-адрес файла RA. Например:

```
pnm://audio.real.com/example.ra
```

Такому файлу логично присвоить имя example.ram. Строчка `pnm://` означает, что данные находятся на сервере RealAudio. Если на Web-страницах необходимо расположить ссылку на файл формата RealAudio, то обычно ссылаются на метафайл RAM, а не на сам файл RA, хотя это и не запрещено. Для чего так сделано? Метафайлы могут содержать целый список адресов файлов RA,

которые будут воспроизводиться последовательно. Кроме того, в метафайлах могут располагаться сведения о моменте времени начала воспроизведения файлов, если требуется проигрывать их с определенного места.

Для этого следует указать точку начала воспроизведения после адреса файла, отделив символом $:

```
pnm://audio.real.com/example.ra$1:20
```

Полностью формат указания момента начала воспроизведения записывается следующим образом:

```
$dd:hh:mm:ss.t
```

где dd — дни, hh — часы, mm — минуты, ss — секунды и t — десятые доли секунды.

Если вы хотите, чтобы при просмотре страницы использовался плейер как вспомогательное приложение, то достаточно лишь применить стандартную ссылку, например:

```
<A HREF=example.ram>Ссылка на метафайл</A>
```

Другим вариантом является встраивание с помощью тэга <EMBED>, например:

```
<EMBED SRC=example.rpm WIDTH=500 HEIGHT=100>
```

Обратите внимание, что при встраивании файлов для активизации подключаемого модуля, а не самостоятельного приложения, необходимо ссылаться на метафайл с расширением RPM, а не RAM. Никаких иных отличий, кроме расширения, правила записи метафайлов не имеют.

Параметры WIDTH и HEIGHT указывают размеры окна для органов управления подключаемого модуля. Как и параметр SRC, они являются обязательными.

Примечание

Выбор значений размеров окна полностью определяется автором страницы. Подключаемый модуль построен таким образом, что его органы управления автоматически масштабируются по указанным размерам и, таким образом, всегда полностью занимают отводимое окно. В этом отличие, например, от модуля LiveAudio, описанного выше.

Вот как выглядит обобщенный синтаксис тэга <EMBED>, если он используется для встраивания файлов RealAudio:

```
<EMBED SRC=source_URL WIDTH=width_value HEIGHT=height_value
[CONTROLS=option] [AUTOSTART=True] [CONSOLE=value] [NOLABELS=True]>
```

Параметры CONTROLS, AUTOSTART, NOLABELS и CONSOLE являются специфическими для RealAudio и могут отсутствовать.

Параметр CONTROLS определяет набор органов управления модуля плейера RealAudio. В табл. 7.3 описаны возможные значения этого параметра.

Таблица 7.3. *Значения параметра* `CONTROLS`

Значение	Описание
`CONTROLS=All`	Окно плейера содержит все органы управления, включая `ControlPanel`, `InfoVolumePanel` и `StatusBar`. (Это значение используется по умолчанию)
`CONTROLS=ControlPanel`	Окно плейера содержит кнопки воспроизведения, паузы, остановки, индикатор текущего положения и изображение регулятора громкости. (Аналог окна самостоятельного плейера, если не отмечена ни одна из опций меню **View**)
`CONTROLS=InfoVolumePanel`	Окно плейера содержит строки для вывода информации о файле и движковый регулятор громкости. (Аналог окна самостоятельного плейера, если отмечена опция **Clip Info** меню **View**)
`CONTROLS=InfoPanel`	Окно плейера содержит строки для вывода информации о файле
`CONTROLS=StatusBar`	Окно плейера содержит строку состояния, указатели времени воспроизведения и длительности клипа. (Аналог окна самостоятельного плейера, если отмечена опция **Status Bar** меню **View**)
`CONTROLS=PlayButton`	Окно плейера содержит только кнопки воспроизведения и паузы
`CONTROLS=StopButton`	Окно плейера содержит только кнопку остановки
`CONTROLS=VolumeSlider`	Окно плейера содержит только движковый регулятор громкости
`CONTROLS=PositionSlider`	Окно плейера содержит только индикатор текущего положения
`CONTROLS=PositionField`	Встраивает в строку состояния время воспроизведения и длительность клипа
`CONTROLS=StatusField`	Встраивает в строку состояния текстовые сообщения

Если параметру `AUTOSTART` присвоено значение `TRUE`, проигрывание файла начинается автоматически. Несколько файлов не могут звучать одновременно. Поэтому, если более одного тэга `<EMBED>` имеют `AUTOSTART=TRUE`, только последний из них будет запущен автоматически. Порядок загрузки файлов зависит от программы-сервера и кэш-памяти браузера Netscape.

Параметр `CONSOLE` позволяет создать несколько окон с различным набором органов управления, осуществляющими управление одним и тем же клипом. Изначально каждое окно связано со своим клипом; если же применить параметр `CONSOLE`, то предоставится возможность управлять одним клипом из

различных окон. Например, запустить клип из одного окна, а остановить, пользуясь кнопками другого окна. Появляется возможность создания произвольного набора кнопок для отдельного клипа, расположенных также произвольным образом. Для реализации этого свойства следует указать одно и то же имя клипа в каждом из тэгов `<EMBED>`, а также присвоить одинаковые значения параметру `CONSOLE`:

```
<EMBED SRC="sample.rpm" WIDTH=30 HEIGHT=33 CONTROLS="PlayButton"
CONSOLE="Clip1">
<EMBED SRC="sample.rpm" WIDTH=300 HEIGHT=33 CONTROLS="PositionSlider"
CONSOLE="Clip1">
```

Еще один пример на эту тему:

Пример 7.2. Пример документа со встроенными файлами RealAudio

```
<HTML>
<HEAD>
<TITLE>Пример встраивания файлов формата RealAudio</TITLE>
</HEAD>

<BODY>
<H2>Различные варианты встраивания клипов формата RealAudio</H2>

<H3>Кнопки воспроизведения/паузы, кнопка остановки<BR>
и индикатор текущего положения</H3>

<EMBED SRC=sample1.rpm WIDTH=50 HEIGHT=30 CONTROLS=PlayButton
CONSOLE=clip1>
<EMBED SRC=sample1.rpm WIDTH=50 HEIGHT=30 CONTROLS=StopButton
CONSOLE=clip1>
<EMBED SRC=sample1.rpm WIDTH=350 HEIGHT=30 CONTROLS=PositionSlider
CONSOLE=clip1>

<H3>Только панель управления</H3>
<EMBED SRC=sample2.rpm WIDTH=300 HEIGHT=35 CONTROLS=ControlPanel>

<H3>Полный набор органов управления</H3>
<EMBED SRC=sample3.rpm WIDTH=500 HEIGHT=100 CONTROLS=All>

<P>
<A HREF=sample3.ram>Ссылка, при выборе которой
 будет запущена самостоятельная программа-плейер</A>

</BODY>
</HTML>
```

В примере 7.2 в HTML-документ встроено три клипа RealAudio и имеется ссылка на еще один файл. Отображение этого документа в браузере приве-

дено на рис. 7.8. Первый из файлов sample1.rpm встраивается трижды с различным набором органов управления. Поскольку во всех этих тэгах `<EMBED>` есть строчка `CONSOLE=clip1`, то кнопки управления будут относиться к одному и тому же клипу. Например, запустив клип кнопкой воспроизведения, можно его остановить, нажав кнопку в другом окне (расположенном от него справа). По существу, все три окна, с точки зрения их функционирования, будут являться единой консолью управления для этого клипа. Естественно, что эти окна могли быть разбросаны по странице в произвольном порядке. При встраивании клипов RealAudio вид консоли управления определяется автором страницы.

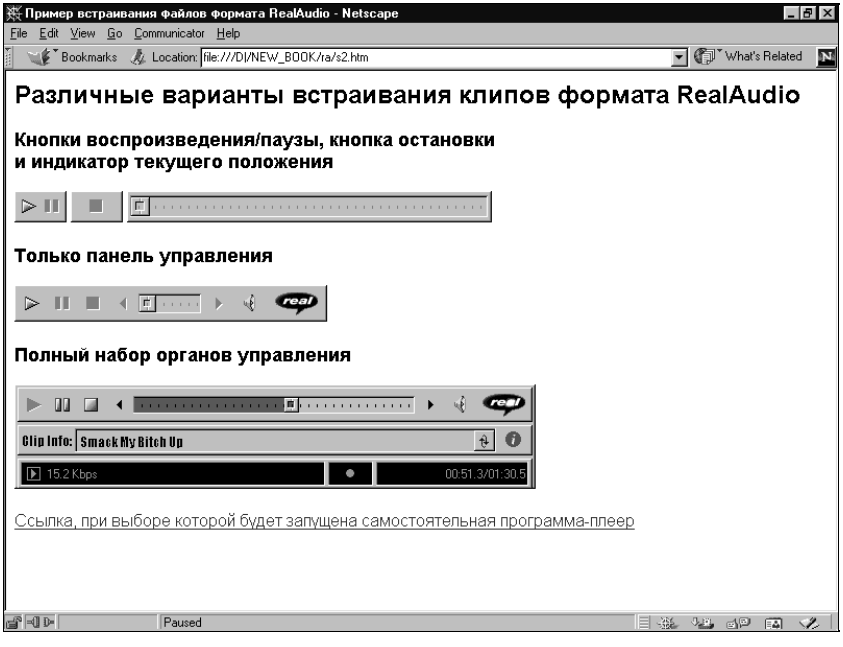


Рис. 7.8. Различные варианты встраивания клипов RealAudio

Следующие два клипа иллюстрируют возможности встраивания консоли управления с различным набором кнопок.

Файл sample1.rpm имеет следующее содержание:

```
file://smack.ra
```

Остальные метафайлы имеют аналогичное содержание. Здесь для примера показано, что данные для плейера могут поступать не только от соответствующего сервера в реальном времени, а и представлять собой обычные файлы формата RA, расположенные локально.

Продолжим рассмотрение значений параметров тэга `<EMBED>`. Если параметру `CONSOLE` одного из тэгов `<EMBED>` присвоить значение `"_master"`, то при

запуске любого другого клипа на странице им можно управлять при помощи кнопок этого "главного" окна. Это удобно использовать, если вы, например, хотите, чтобы в странице было размещено несколько клипов с ограниченным набором органов управления, но при запуске любого из них иметь полный набор кнопок. Пример использования данного свойства приведен ниже.

Пример 7.3. Пример документа с мастер-консолью

```
<HTML>
<HEAD>
<TITLE>The Prodigy</TITLE>
</HEAD>
<BODY>
<H3>Встраивание нескольких клипов, управляемых одной консолью</H3>

<TABLE BORDER=1 CELLSPACING=0 CELLPADDING=3 ALIGN=left>
<TR><TH COLSPAN=3>Альбом "The fat of the land"</TH></TR>

<TR><TD><EMBED SRC=sample1.rpm WIDTH=50 HEIGHT=30
CONTROLS=PlayButton></TD>
<TD>Smack my bitch up</TD><TD>5:42</TD></TR>

<TR><TD><EMBED SRC=sample2.rpm WIDTH=50 HEIGHT=30
CONTROLS=PlayButton></TD>
<TD>Breathe</TD><TD>5:34</TD></TR>

<TR><TD><EMBED SRC=sample3.rpm WIDTH=50 HEIGHT=30
CONTROLS=PlayButton></TD>
<TD>Diesel power</TD><TD>4:17</TD></TR>

<TR><TD><EMBED SRC=sample4.rpm WIDTH=50 HEIGHT=30
CONTROLS=PlayButton></TD>
<TD>Funky shit</TD><TD>5:15</TD></TR>

<TR><TD><EMBED SRC=sample5.rpm WIDTH=50 HEIGHT=30
CONTROLS=PlayButton></TD>
<TD>Serial thrilla</TD><TD>5:10</TD></TR>

<TR><TD><EMBED SRC=sample6.rpm WIDTH=50 HEIGHT=30
CONTROLS=PlayButton></TD>
<TD>Mindfiels</TD><TD>5:39</TD></TR>

<TR><TD><EMBED SRC=sample7.rpm WIDTH=50 HEIGHT=30
CONTROLS=PlayButton></TD>
<TD>Narayan</TD><TD>9:04</TD></TR>

<TR><TD><EMBED SRC=sample8.rpm WIDTH=50 HEIGHT=30
CONTROLS=PlayButton></TD>
<TD>Firestarter</TD><TD>4:40</TD></TR>
```

```
<TR><TD><EMBED SRC=sample9.rpm WIDTH=50 HEIGHT=30
CONTROLS=PlayButton></TD>
<TD>Climbatize</TD><TD>6:36</TD></TR>

<TR><TD><EMBED SRC=sample10.rpm WIDTH=50 HEIGHT=30
CONTROLS=PlayButton></TD>
<TD>Fuel my fire</TD><TD>4:18</TD></TR>
</TABLE>

<EMBED SRC=empty.rpm WIDTH=300 HEIGHT=100 CONTROLS=All CONSOLE="_master">

<CENTER><IMG SRC=newlogo.gif></CENTER>
</BODY>
</HTML>
```

В приведенном примере встроено десять клипов, объединенных для удобства в таблицу. Каждый из встроенных клипов имеет минимальный набор кнопок управления: воспроизведения и паузы, что определяется параметром `CONTROLS=PlayButton`. Справа от таблицы (рис. 7.9) располагается еще одна консоль управления с полным набором кнопок, для которой указано значение `CONSOLE="_master"`. Заметим, что для этого окна также имеется отдельный встроенный файл (empty.rpm), содержимое которого никогда не используется. Единственное, что необходимо, это обеспечить существование непустого файла с таким именем.

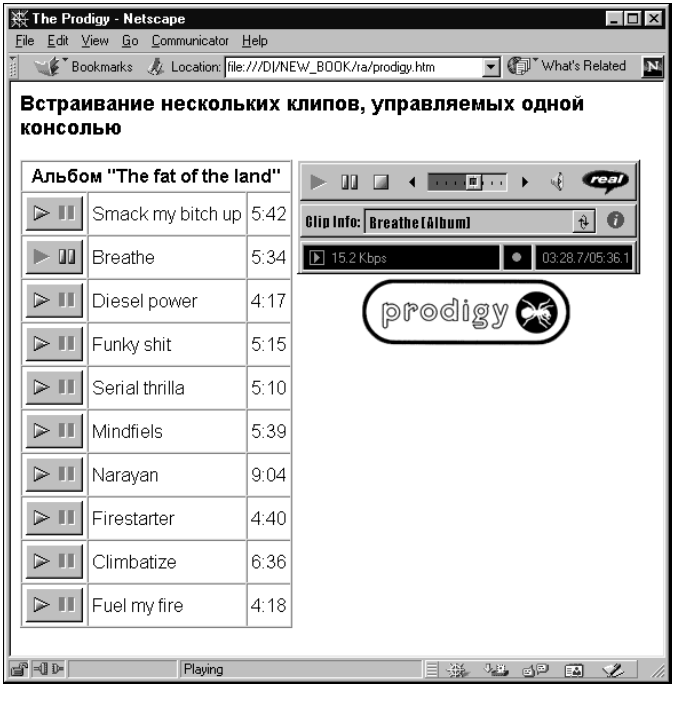


Рис. 7.9. Несколько встроенных клипов, управляемых из одного окна

Запуск на воспроизведение любого из десяти встроенных клипов приводит к активизации мастер-консоли. В этом окне будут отображаться все параметры запущенного клипа, а также будут работать все кнопки управления.

Последним параметром тэга `<EMBED>` является `NOLABELS`. Если ему присвоено значение `TRUE`, то вывод информации о клипе (название, автор и копирайт) будет запрещен.

Для тех пользователей, браузеры которых не поддерживают тэг `<EMBED>`, включите после этого тэга контейнер `<NOEMBED>` с альтернативным содержанием:

```
<NOEMBED>Что-то для браузеров, не поддерживающих подключаемые
модули</NOEMBED>
```

К примеру, можно такой строке дать ссылку для запуска самостоятельной программы-плейера:

```
<EMBED SRC="sample.rpm" WIDTH=300 HEIGHT=100>
<NOEMBED><A SRC="sample.ram">Сейчас запустится
программа RealPlayer</A></NOEMBED>
```

Примечание

Несмотря на то, что все примеры встраивания файлов формата RealAudio приведены для браузера Netscape, они будут точно также работать и в браузере Internet Explorer. Это заслуга компании-разработчика плейера.

Если вам потребуется сконфигурировать программу-сервер для работы со звуковыми файлами формата RealAudio, то нужно задать соответствующие MIME-типы. В табл. 7.4 перечислены MIME-типы и соответствующие им расширения файлов.

Таблица 7.4. *MIME-типы для RealAudio*

MIME-тип	Расширение
audio/x-pn-realaudio	RA, RAM
audio/x-pn-realaudio/plugin	RPM

Ресурсы RealAudio в Интернете

Основным сайтом для получения разнообразной информации, связанной с технологией RealAudio, является официальный сайт разработчика этой технологии **http://www.real.com.**

Информацию о передачах в формате RealAudio можно получить по адресу **http://www.timecast.com**. В недавнем прошлом это был отдельный сервер,

однако, сейчас обращение по указанному адресу перебрасывается на страничку **http://www.real.com/realguide/index.html**.

Появилось достаточное количество серверов RealAudio на русском языке. Их число постоянно растет, найти их адреса можно, воспользовавшись любой поисковой системой, например, по ключевому слову RealAudio. Укажем лишь некоторые из наиболее интересных адресов.

Большой список серверов RealAudio, работающих на русском языке, можно найти по адресу **http://www.guzei.com/radio/**.

Весьма обширная коллекция файлов в формате RealAudio располагается по адресу **http://www.rus.org/mtrros/real/** (рис. 7.10).

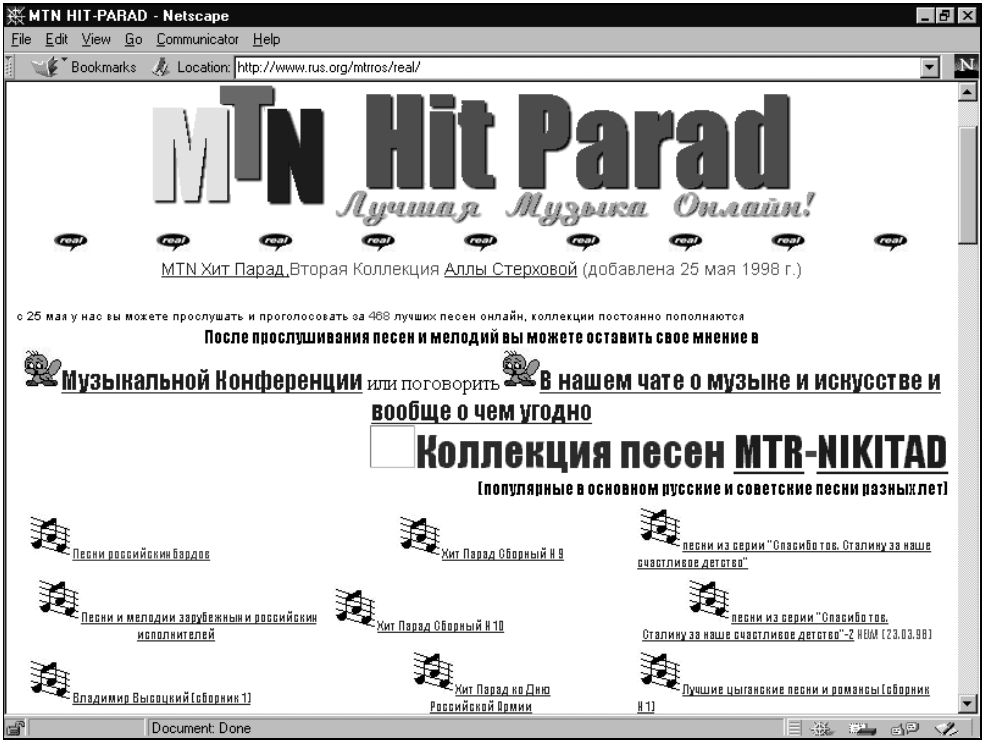


Рис. 7.10. Постоянно растущая коллекция песен на русском языке в формате RealAudio

Не обошел вниманием возможности использования технологии RealAudio и известный российский Web-мастер Артемий Лебедев. В своем музыкальном разделе он расположил гимны всех республик бывшего СССР в этом формате для четырех разных скоростей соединений (рис. 7.11).

Естественно, что возможности Интернета стали осваивать и ведущие российские радиостанции. Радиостанция "Маяк" является одной из старейших

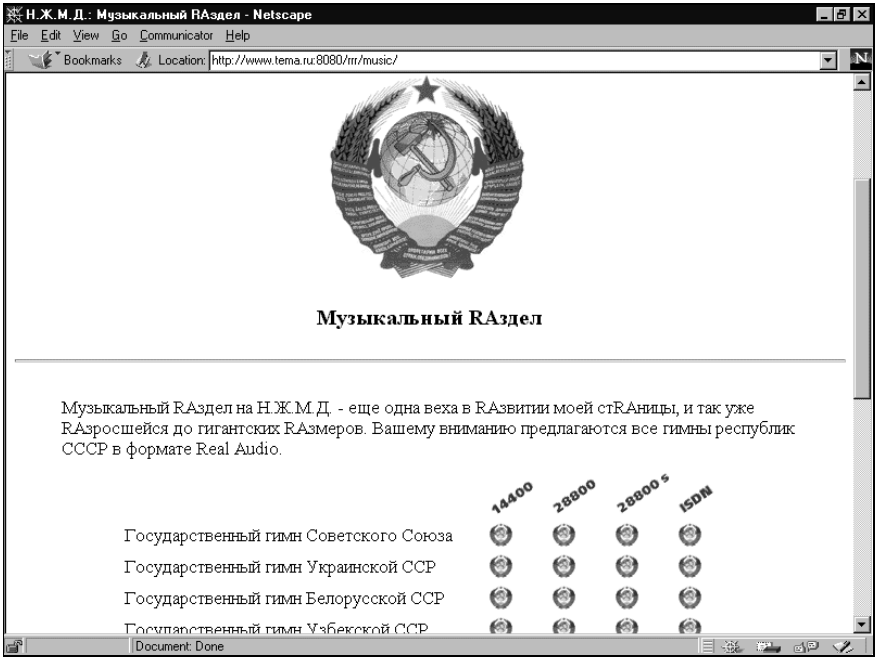


Рис. 7.11. Музыкальный раздел с гимнами всех республик бывшего СССР

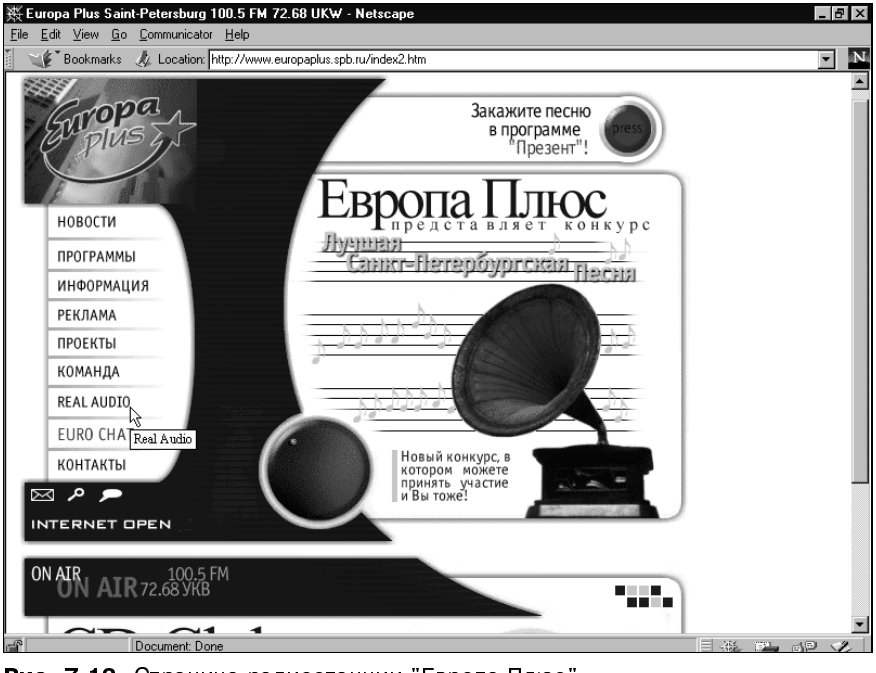


Рис. 7.12. Страница радиостанции "Европа Плюс"

и наиболее известных в России. Пожалуй, нет в нашей стране ни одного человека, который бы ни разу не слышал ее позывных. Станция начала вещание 1 августа 1964 г. и с успехом продолжает его и сегодня. Начато вещание и в сети Интернет в формате RealAudio (**http://www.radiomayak.ru/real /live.ram**).

Одной из наиболее популярных радиостанций, появившихся в последние годы, является станция "Европа Плюс". Она уже отметила свое 8-летие (основана в 1991 г.) и на сегодняшний день занимает второе место среди коммерческих радиостанций России. На рис. 7.12 показана страничка Санкт-Петербургского отделения радиостанции (**http://www.europaplus.spb.ru**). Как видно из рисунка, в планах компании осуществление вещания в формате RealAudio.

Звуковые файлы формата MP3

Как уже отмечалось в начале главы, хранение звуковых данных в оцифрованной форме с высоким качеством требует довольно больших затрат дисковой памяти. Попытки сократить объем файлов, используя стандартные архиваторы, обычно не приводят к значительному выигрышу из-за специфичности звуковых данных. Тем не менее, добиться довольно значительного уровня сжатия видео- и аудиоинформации удается при использовании специфических методов, основанных на анализе структуры данных и последующем сжатии с некоторыми потерями.

Реальная возможность обработки изображений и звука, сравнимых по качеству с существовавшими аналоговыми примерами, появилась только в конце 80-х годов.

В 1988 году Международной организацией стандартов ISO (International Standards Organization) был сформирован комитет MPEG (Moving Pictures Expert Group, группа экспертов в области движущихся изображений), основной задачей которого являлась разработка стандартов кодирования подвижных изображений, звука и их комбинации. За десять лет своего существования комитет выработал ряд стандартов по данному вопросу. В результате, обобщив обширные исследования в этой области, был рекомендован определенный набор методов сжатия аудио- и видеоданных. Был определен ряд специфических форматов для хранения данных, отличающихся по качеству результатов и скорости передачи данных.

В настоящее время существует три стандарта хранения видеоданных: MPEG-1, MPEG-2 и MPEG-4. В рамках форматов MPEG-1 и MPEG-2 существуют также форматы хранения звуковой информации, которые носят название Layer-1, Layer-2 и Layer-3. Эти три звуковых формата определены для MPEG-1 и незначительными расширениями используются в MPEG-2. Все три формата похожи друг на друга, но используют различные уровни

компромисса между сжатием и сложностью. Уровень Layer-1 — наиболее простой, не требует значительных затрат на сжатие, но и дает небольшую степень сжатия. Уровень Layer-3 — наиболее трудоемкий и предлагает самое лучшее сжатие. В последнее время огромную популярность завоевал формат Layer-3, который часто называют просто MP3. Такое название связано с типичным расширением MP3 для звуковых файлов, хранящихся в этом формате. Это сокращение укоренилось, стало общепринятым, в частности, дало название ряду сайтов, посвященных этой технологии (**www.mp3.com**, **mp3.box.sk**, **mp3soft.da.ru** и др.).

Основная идея, на которой основаны все методики сжатия аудиосигнала с потерями, — пренебрежение тонкими деталями звучания оригинала, лежащими вне пределов возможностей человеческого слуха. Здесь можно выделить несколько моментов.

Уровень шума. Звуковое сжатие базируется на простом факте. Если человек находится рядом с громко воющей сиреной, то вряд ли он услышит разговор стоящих неподалеку людей. Причем это происходит не оттого, что человек обращает больше внимание на громкий звук, а в большей степени оттого, что человеческое ухо фактически теряет звуки, лежащие в том же диапазоне часто, что и значительно более громкий звук. Этот эффект носит название маскирующего, он изменяется с различием в громкости и частоте звука.

Одним из основных способов сжатия звука является уменьшение числа бит, используемых для хранения данных при дискретизации звука. Уменьшение числа бит эквивалентно добавления шума к звуку. Сжатие MPEG использует маскирующий эффект, имея в виду, что человек все равно не услышит появляющийся в результате уменьшения числа бит шум.

Вторым моментом является деление полосы звуковых частот на подполосы. Каждая из выбранных подполос далее обрабатывается отдельно. Программа кодирования выделяет самые громкие звуки в каждой подполосе и использует эту информацию для определения приемлемого уровня шума для этой подполосы. Лучшие программы кодирования учитывают также влияние соседних полос. Очень громкий звук в одной подполосе может повлиять на маскирующий эффект и на близлежащие полосы.

Еще одним моментом кодирования является использование психоакустической модели, опирающейся на особенности человеческого восприятия звука. Сжатие с использованием этой модели основано на удалении заведомо не слышимых частот с более тщательным сохранением звуков, хорошо различаемых человеческим ухом. К сожалению, здесь не может быть точных математических формул. Восприятие звука человеком — сложный, до конца не исследованный процесс, поэтому выбор методов сжатия выполняется на основе анализа прослушивания и сравнения по-разному сжатых звуков группами экспертов. Зато здесь имеются практически неограниченные возможности в сфере улучшения психоакустических моделей.

Большинство существующих алгоритмов для кодировки человеческого голоса основано на высокой степени предсказуемости такого сигнала — универсальные алгоритмы сжатия MPEG с переменным успехом пытаются применить этот прием. Еще одним приемом сжатия является использование так называемого совмещенного стерео. Известно, что слуховой аппарат человека может определить направление лишь средних частот — высокие и низкие звучат как бы отдельно от источника. Значит, эти фоновые частоты можно кодировать в моно сигнал.

Кроме всего этого для сжатия используется различие в сложности потоков в каналах. Например, если в правом канале какое-то время полная тишина, это "зарезервированное" место используется для повышения качества левого канала или туда впихиваются необходимые биты, не влезшие в поток чуть раньше.

На последней стадии сжатия используется алгоритм сжатия Хаффмана, используемый во многих стандартных программах сжатия. Этот процесс позволяет улучшить степень сжатия для относительно однородных сигналов, которые плохо сжимаются с помощью описанных выше приемов.

На основе описанных идей строятся алгоритмы сжатия, позволяющие достичь степени компрессии 10:1 и выше практически без потери в качестве звучания. При кодировании задают требуемый уровень компрессии, а алгоритмы сжатия добиваются требуемого уровня сжатия за счет качества. Для данного применения требуемый уровень сжатия обычно указывают в виде величины потока данных (bit rate), измеряемого в Кбит/с.

Примечание

Термин "bit rate" обозначает количество битов передаваемой информации в секунду. На русский язык этот термин переводится по-разному в различных источниках. В последнее время часто вместо формального перевода употребляют новое для русского языка слово "битрейт". Вариантами перевода также являются следующие: "ширина потока данных", "сложность потока битов", "скорость потока", "битовая частота". Иногда для звуковых файлов этот же параметр называют степенью сжатия файла. Например, говорят, что файл сжат до 128 Кбит/с. Дело в том, что величина битрейта напрямую связана с размером звукового файла в расчете на одну секунду звучания (уменьшение размера файла в определенное количество раз обеспечивает снижение битрейта в такое же количество раз).

Воспроизведение файлов формата MP3

Для проигрывания файлов MP3 существует довольно большое количество программ-плейеров, число которых измеряется десятками. Большой список таких программ можно найти на сайте **http://www.dailymp3.com**. Однако среди перечисленных программ есть всего несколько, пользующихся огромной популярностью и занимающих практически весь рынок программ. Безус-

ловным лидером является программа Winamp, обсуждение которой будет дано ниже. Дальнейшему росту популярности программы Winamp будет способствовать ее включение в дистрибутив пакета Netscape Communicator версии 4.7.

Программы воспроизведения отличаются друг от друга как интерфейсом, так и такими важными параметрами, как качество проигрывания и скорость декодирования. По скорости декодирования самым быстрым плейером был и остается на сегодняшний день K-Jofol, краткие характеристики которого также будут приведены ниже.

Вопрос качества воспроизведения не так прост, как может показаться. Несмотря на то, что стандарт MP3 однозначно определяет, какие именно данные содержатся в MP3-файле, это не значит, что их декодирование обязательно должно быть однозначным. Причин неоднозначности несколько. Во-первых, независимо от сути алгоритма декодирования, оно может проводиться с разной точностью. По данному признаку декодеры разделяют на 32-битовые и 64-битовые. Существует также 110-битовый декодер NAD.

Во-вторых, как и в случае с цифровыми данными, записанными на CD, содержащаяся в файлах информация является лишь приближением к исходному сигналу. И как для более адекватного воспроизведения аудио CD с компенсацией дефектов, вызванных записью на CD, в высококлассной аппаратуре применяются различные методы обработки звука, так и в случае файлов MP3 в принципе можно значительно улучшить звучание, если знать, какие из вносимых сжатием в MP3 искажений более заметны и какие поддаются коррекции.

По качеству звука лидером считается программа NAD, возрождающаяся ныне под названием NADDY (**http://ae.dmusic.com**).

Установка программы Winamp

Программа-плейер Winamp (Windows Audio Mpeg Player) предназначена для воспроизведения звуковых файлов в различных форматах (MP3, WAV, MIDI и др.) и может устанавливаться в среде Windows 95/98/NT. Для качественного воспроизведения файлов MP3 минимальным требованием является компьютер на базе процессора 486 последних моделей. Рекомендуется использовать Pentium и выше. Программа разработана компанией Nullsoft (**http://www.nullsoft.com**), для получения сведений о новейших разработках можно обратиться к официальному сайту **http://www.winamp.com**. Совсем недавно появился сайт **http://www.winamp.ru**, на котором будет дублироваться информация официального сайта на русском языке.

Инсталляция программы на компьютер в среде Windows очень проста. Инсталляционный файл программы представляет собой исполняемый файл (EXE-файл) размером порядка 500 Кб (для популярной в свое время версии 2.05).

Примечание

Последующие версии имеют незначительно отличающийся размер инсталляционного файла. Правда, начиная с версии 2.24, компания-разработчик стала предлагать программу в двух вариантах — базовом и полном. Базовый комплект в отличие от полного не имеет некоторых возможностей, которые на самом деле не нужны для обычного прослушивания звуковых файлов, поэтому вполне годится для типичного применения программы. Размер базового комплекта версии 2.5с составляет примерно 600 Кб, а полного комплекта той же версии — 1,8 Мб.

Запустите этот файл на исполнение, и вы увидите окно, как на рис. 7.13.

Выберите имя каталога, в который необходимо установить программу и нажмите кнопку **Install**. Появится окно (рис. 7.14), в котором предлагается обновить список ссылок на ресурсы программы Winamp в сети Интернет. Если ваш компьютер подключен к Интернету, нажмите кнопку **Да**, в противном случае нажимайте **Нет**.

Рис. 7.13. Окно инсталлятора программы Winamp

Рис. 7.14. Окно запроса обновления динамических ссылок

После этих действий программа автоматически запустится в режиме воспроизведения демонстрационного файла формата MP3 (demo.mp3). На этом установка будет завершена.

Заметим, что на настоящий момент существует довольно много версий программы. Новые версии программы Winamp в отдельные моменты появлялись чуть ли не каждую неделю (табл. 7.5, в этой таблице приведен далеко не исчерпывающий список версий). Это, с одной стороны, свидетельствует о том, что разработчики программы активно работают над ее модернизацией. С другой стороны, скоропалительный выпуск новых версий не позволяет тщательно отлаживать программу. В итоге выпуск новой версии таит в себе и новые ошибки, исправляемые в последующих версиях. В частности, для версии 2.09 авторам пришлось буквально через два дня выпустить файл-заплатку (patch), которая исправляет допущенные ошибки и изменяет номер

версии на 2.091. Разные версии программы имеют и определенные возможности и особенности, зачастую не описываемые разработчиками. Основные возможности различных версий можно всегда посмотреть на сайте компании-разработчика, а также в справочном текстовом файле, появляющемся в каталоге при установке программы. Пользователю обычно не интересно изучать нюансы версий программы, однако полезно знать основные особенности и рекомендации по использованию, так как с программами различных версий приходится сталкиваться в различных архивах и на CD-дисках с музыкальными файлами. Активные пользователи по возможности стремятся устанавливать последнюю версию, считая, что она, по крайней мере, не хуже, что для данной программы не всегда справедливо.

Отметим лишь важнейшие и наиболее интересные особенности версий. Версия 1.92 ознаменовала рождение нового поколения программы Winamp, включив в себя новый декодер Nitrane 1.0, обеспечивающий поддержку технологий MMX и 3DNow! Декодер Nitrane использовался во многих последующих версиях программы, однако в версии 2.20 был заменен на другой декодер, лицензированный немецким институтом Fraunhofer IIS (о вкладе этой организации в развитие технологий кодирования звука см. ниже в этой главе). Этот декодер использовался в версиях 2.20-2.22. Естественно, что при выходе этих версий говорилось, что программа оснащена новым декодером, улучшившим качество звука. При этом лишь кое-где мельком упоминалось, что этот декодер работает лишь с форматом звука Layer-3, а форматы Layer-1 и Layer-2 корректно не воспроизводит. Незнание этой особенности программы оказалось очень неприятным. Дело в том, что в настоящее время большинство звуковых файлов MPEG записываются в формате Layer-3, однако встречаются и старые записи Layer-1 и Layer-2. Причем довольно часто в фонотеках все файлы независимо от формата имеют расширение MP3. Отличить формат файла по расширению становится невозможно. Это можно сделать лишь посмотрев информацию самого музыкального файла (**View file info**). Что же происходит в итоге. Установив новую версию программы в определенный момент обнаруживаешь, что некоторые звуковые файлы, прекрасно воспроизводившиеся предыдущими версиями программы, перестали корректно воспроизводиться новой версией по непонятным причинам. Такая ситуация весьма неприятна для пользователя и весьма подрывает авторитет компании-разработчика. Данный декодер использовался лишь в трех версиях программы, и в версии 2.23 разработчики вновь вернулись к декодеру Nitrane обновленной версии 1.60. Возвращение декодера Nitrane сопровождалось рекламными характеристиками об уменьшении загрузки процессора при декодировании файлов, появилась даже фраза "Nitrane жив!". Все эти изменения обросли легендами и мифами, в которых правда соседствует с вымыслом. Так возврат к декодеру Nitrane был обыгран следующим образом. Было сказано, что автор программы Winamp Дж. Френкель разбил свой автомобиль Audi и после случившегося потерял всякое доверие к не-

мецкой инженерии, что и послужило поводом для возврата к собственному декодеру Nitrane. Косвенным доказательством реальности случившегося может служить фотография разбитого автомобиля, файл с которой можно найти на просторах Интернета. В других источниках указывалось, что в течение некоторого времени происходила судебная тяжба между компанией Nullsoft и неким истцом, в продолжение которой компании Nullsoft было запрещено использовать собственный декодер. По окончании тяжбы все встало на свои места. Как все происходило на самом деле, обычному пользователю совершенно не интересно. Эти подробности приведены лишь для того, чтобы показать разнообразие возможных мотивов изменения характеристик программ и их неожиданное влияние на потребительские качества.

Отметим еще некоторые возможности различных версий. Возможность локализации программы Winamp была заложена довольно давно, однако лишь летом 1999 г. появился модуль русификации (Language Pack). Файл для русификации имеет имя Russian.lng и существует в трех вариантах — для версии 2.10, группы версий 2.2x и версии 2.5x.

Программа Winamp распространялась по принципу shareware, подразумевающему двухнедельный оценочный период и последующую регистрацию программы с небольшой оплатой (см. рис. 7.13). К чести разработчиков программы следует заметить, что не было сделано никаких ограничений к использованию программы после прохождения двухнедельного срока после ее установки. Более того, начиная с версии 2.50, программа стала полностью бесплатной (freeware).

Таблица 7.5. *Версии программы Winamp*

№ п/п	Номер версии	Дата появления	№ п/п	Номер версии	Дата появления
1	0.2a	21.04.97	12	2.0	08.09.98
2	0.92	01.05.97	13	2.01	25.09.98
3	1.00	07.06.97	14	2.02	29.09.98
4	1.20	31.07.97	15	2.03	08.10.98
5	1.40	03.09.97	16	2.04	25.10.98
6	1.50	19.09.97	17	2.05	15.11.98
7	1.66	14.01.98	18	2.06	31.12.98
8	1.80	18.03.98	19	2.09	11.01.99
9	1.90	31.03.98	20	2.091	13.01.99
10	1.91	20.05.98	21	2.10	24.03.99
11	1.92	27.06.98	22	2.20	03.05.99

Таблица 7.5 *(окончание)*

№ п/п	Номер версии	Дата появления	№ п/п	Номер версии	Дата появления
23	2.21	12.05.99	27	2.5	25.08.99
24	2.22	26.05.99	28	2.5c	27.08.99
25	2.23	08.06.99	29	2.5e	04.10.99
26	2.24	13.07.99			

Довольно популярной долгое время была версия 2.05, которая работает весьма устойчиво. Начиная с версии 2.06, программа Winamp была переписана на Visual C++. По утверждению авторов программа должна была стать меньше по размерам и работать надежнее.

Управление программой Winamp

Работа с программой проигрывания звуковых файлов Winamp не вызывает затруднений. Вы выделяете необходимые звуковые файлы и кидаете их (drag-and-drop) в любое место окна программы Winamp. Файлы попадают в список проигрываемых файлов и будут воспроизводиться по очереди. Это наиболее простой способ использования программы.

Внешний вид программы с графическим эквалайзером (Winamp Equalizer) и списком проигрываемых файлов (Winamp Playlist) показан на рис. 7.15. Заметим, что на рисунке показано одновременно три окна: самой программы, эквалайзера и списка файлов. Эти окна могут быть разбросаны по экрану, а могут быть составлены в произвольном порядке друг над другом и перемещаться как единое целое.

Рис. 7.15. Окна программы Winamp

Окошко самой программы имеет следующие кнопки управления (нижний ряд кнопок): переход к предыдущему звуковому файлу из списка (**Previous Track**), воспроизведение (**Play**), пауза (**Pause**), останов (**Stop**), переход к следующему звуковому файлу из списка (**Next Track**), выбор нового файла или файлов (**Eject (open file (s))**). В этом же ряду имеются две кнопки-переключателя: **Shuffle** и **Repeat**, а также кнопка **About**. Если включена кнопка **Shuffle**, то воспроизведение файлов из списка будет выполняться в случайном порядке, в противном случае они будут проигрываться подряд. Включение кнопки **Repeat** обеспечивает бесконечный цикл воспроизведения.

Выше ряда кнопок управления располагается ползунок, перемещающийся по мере исполнения произведения. Перемещая ползунок, можно обеспечить воспроизведение файла с любого места.

Еще выше располагаются ползунки управления громкостью звука и балансом. Тут же расположены кнопки включения/выключения окон эквалайзера (**EQ**) и списка файлов (**PL**).

В окне программы Winamp отображается также информация о воспроизводимом в текущий момент файле. Указывается битрейт файла (например, 128 Кбит/с), частота оцифровки (например, 44 кГц), режим воспроизведения (моно или стерео).

Двойным щелчком по строчке с информацией о текущей фонограмме можно вызвать редактор идентификационных параметров звукового файла (рис. 7.16). В окошках редактора можно увидеть данные, хранящиеся в файле, и при необходимости их отредактировать. Заметим, что программа Winamp даже в нерусифицированном виде ограниченно поддерживает русские буквы.

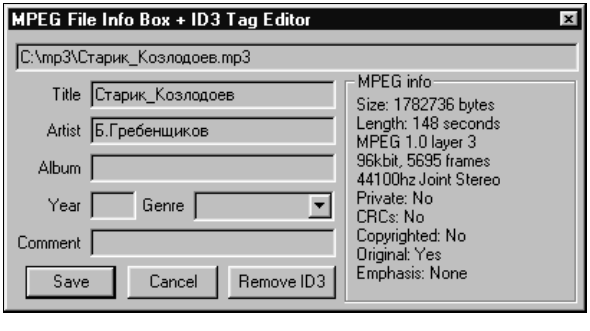


Рис. 7.16. Окно редактора параметров звукового файла программы Winamp

Подключаемые модули программы Winamp

Возможности программы Winamp можно существенно расширить при помощи технологии подключаемых модулей (plug-ins). Подключаемые модули представляют собой дополнительные, специально разработанные программы, которые присоединяются к Winamp и работают совместно с ней. Это может быть программа, которая выводит (под музыку) на экран монитора

какую-нибудь графическую информацию (модули визуализации), программа, добавляющая звуковые эффекты (модули DSP — Digital Sound Processing, цифровая обработка звука), или модули общего назначения (General Purpose plug-ins) — например, добавляют кнопки управления проигрывателем в панель задач. Существуют специальные модули для открытия нестандартных типов файлов, например VQF-файлов, а также модули для вывода звука через нестандартные устройства (например, при переводе звука из формата MP3 в формат WAV).

Примечание

Начиная с версии 2.09 в файл с дистрибутивом программы Winamp более не включаются подключаемые модули (plugins) для визуализации и обработки звука (vis/dsp). Теперь их необходимо скачивать и устанавливать отдельно. Разумеется, все ранее установленные модули сохраняются, если новая версия программы устанавливается поверх предыдущей.

Установка модулей не представляет сложностей. В простейшем варианте подключаемые модули представляют собой небольшие dll-файлы, начало имени которых отражает их назначение (vis — модуль визуализации, dsp — модуль обработки звука, gen — модуль общего назначения, in/out для ввода/вывода). Как правило, модули хранятся в упакованном виде. Необходимо распаковать архив в каталог \Winamp\Plugins\.

Затем запустите Winamp и настройте модуль. Для этого откройте окно Preferences (Ctrl-P), выберите соответствующий модуль и выполните его настройки.

Подключаемые модули программы Winamp можно найти на различных сайтах сети Интернет. В частности, около сотни разнообразных модулей имеется на официальном сайте **http://www.winamp.com**.

Одним из наиболее интересных модулей визуализации является модуль Cthugha, реализующий огромное количество светомузыкальных эффектов (рис. 7.17).

Рис. 7.17. Окно модуля визуализации Cthugha

Информацию о модуле Cthugha можно получить на следующих сайтах:

http://www.afn.org/~cthugha

http://islands.zesoi.fer.hr/~kpisacic/cthugha

http://www.geocities.com/SiliconValley/Lab/6531.

Изменение внешнего вида программы Winamp

В программе Winamp для изменения внешнего вида любого окошка можно воспользоваться технологией установки так называемых Skin'ов. Изображения главного окна проигрывателя, всех кнопок, регуляторов, букв хранятся в отдельных графических BMP-файлах, которые легко редактируются. Это позволяет изменять внешний вид программы по своему усмотрению. Набор таких файлов называется Skin'ом. Слово skin в переводе с английского языка означает кожа, шкура, кожура. Для данного применения этого термина пока не найдено удачного перевода, верно отражающего смысл. Часто употребляется без перевода, например, говорят "установить новый скин". Встречаются термины "сменить шкуру", "установить обшивку". Как бы это ни называли, речь идет об изменении внешнего вида окошек программы Winamp, которое

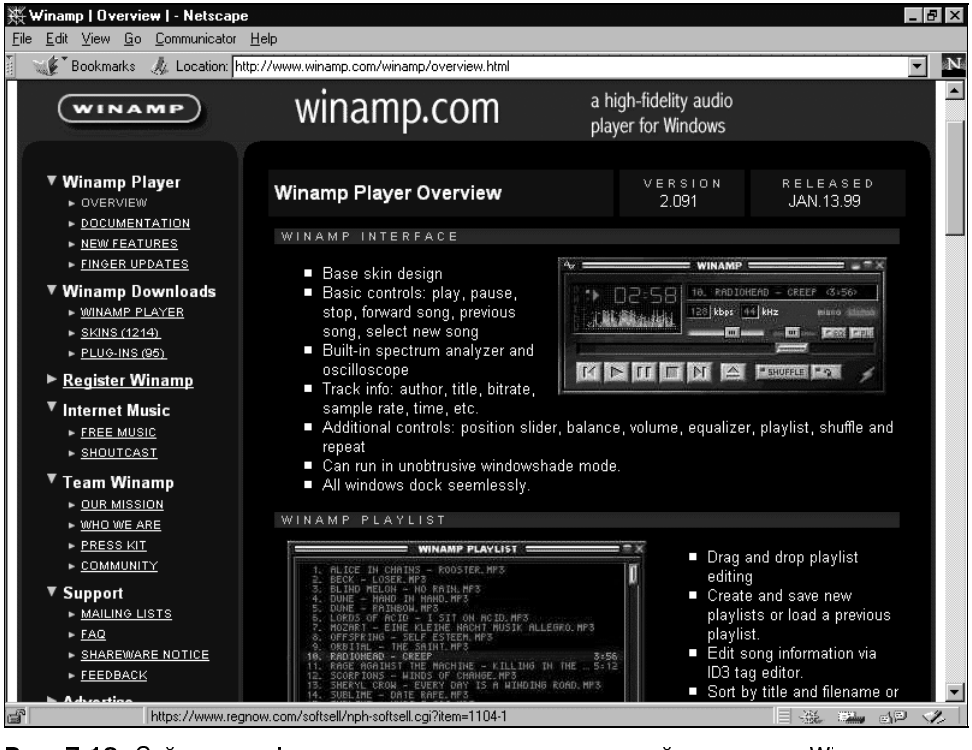


Рис. 7.18. Сайт **www.winamp.com**, целиком посвященный программе Winamp

никак не влияет ни на расположение элементов окон, ни на функциональные возможности программы.

Технология использования Skin'ов, как это ни удивительно, получила очень широкое распространение. В сети Интернет можно найти огромное количество разнообразных Skin'ов для Winamp, даже найти целые коллекции, в которых они разбиты по отдельным рубрикам. Достаточно зайти на специальный сайт (рис. 7.18), посвященный программе Winamp (**http://www.winamp.com**), и скачать оттуда любой из имеющихся Skin'ов, число которых перевалило за тысячу на начало 1999 г. и за три тысячи еще через полгода.

Установить новый Skin довольно просто. Сделайте подкаталог в каталоге \Winamp\Skins\ с именем, соответствующим названию Skin'а. Разархивируйте туда содержимое архива со Skin'ом. Отдельный Skin представляет собой набор примерно десяти файлов с расширением BMP и как минимум одного текстового файла. Некоторые Skin'ы имеют встроенные подключаемые модули (файлы *.DLL), которые нужно скопировать в каталог \Winamp\Plugins\.

После совершения этих операций нажмите в главном окне проигрывателя Alt-S. Появится Skin Browser, где можно опробовать новый Skin.

На рис. 7.19 показаны примеры Skin'ов, изменяющих внешний вид основного окна программы Winamp.

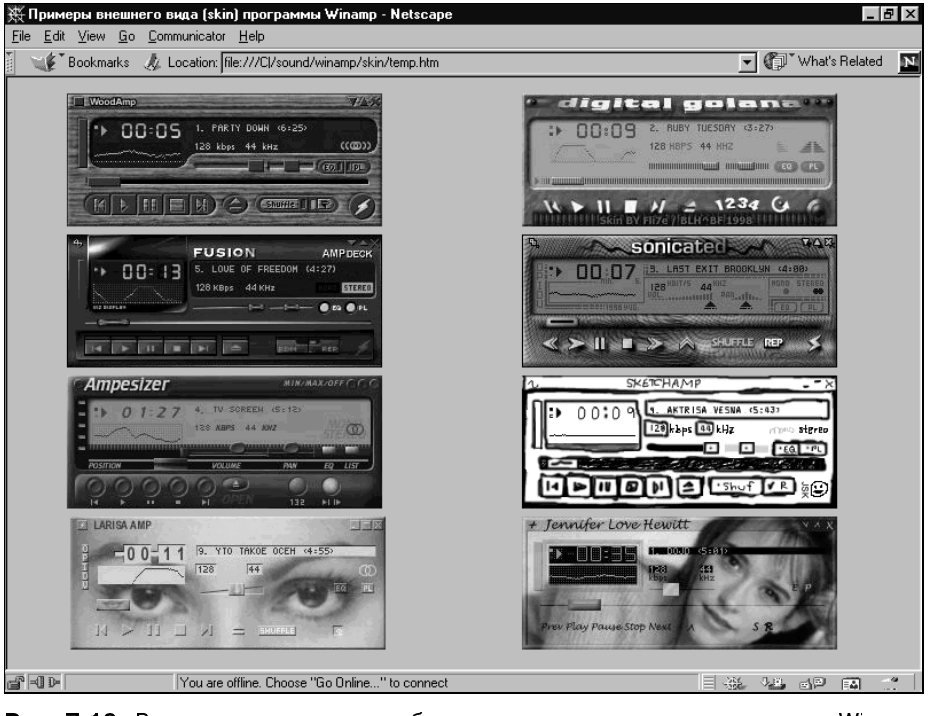


Рис. 7.19. Различные варианты отображения основного окна программы Winamp

Скрытые возможности программы Winamp

Многие программисты закладывают в свои программные продукты некоторые недокументируемые возможности. Иногда это требуется для отладочных, проверочных и других технологических целей. В ряде случаев авторы программ закладывают в них различные шутки ("приколы", как говорят программисты), не влияющие на возможности программ. К последним относятся и несколько скрытых возможностей программы Winamp версии 2.xx.

1. Откройте окно **About Winamp**, выберите вкладку **Winamp** (рис. 7.20). На рисунке вы увидите стилизованное изображение любителя музыки в наушниках, перемещающееся по экрану. Однако, если, находясь на этой вкладке, вы выполните двойной щелчок мышью на слове Copyright, одновременно удерживая клавиши <Ctrl>+<Alt>+<Shift>, то увидите фотографию главного программиста фирмы Nullsoft Джастина Френкеля (Justin Frankel).

Рис. 7.20. Вкладка **Shareware** окна **About Winamp** программы Winamp

2. Откройте окно **About Winamp**, выберите вкладку **Shareware** (рис. 7.21). Вы увидите информацию об использовании вами программы, в частности, количество запусков программы, число исполненных произведений и общее число минут воспроизведения. Это полезная статистическая информация. Попробуйте сделать двойной щелчок мышью на строчке Usage statistics. Появится строчка, извещающая о том, сколько дней на сегодня прожил автор программы.
3. Выберите стандартный внешний вид (skin) программы Winamp. На клавиатуре медленно наберите слово nullsoft, нажимая клавишу <Ecs> после

каждой буквы. Вместо названия окна программы Winamp появится строчка текста, которая, в частности, озвучена в демонстрационном файле demo.mp3. Перевод текста оставляем читателю.

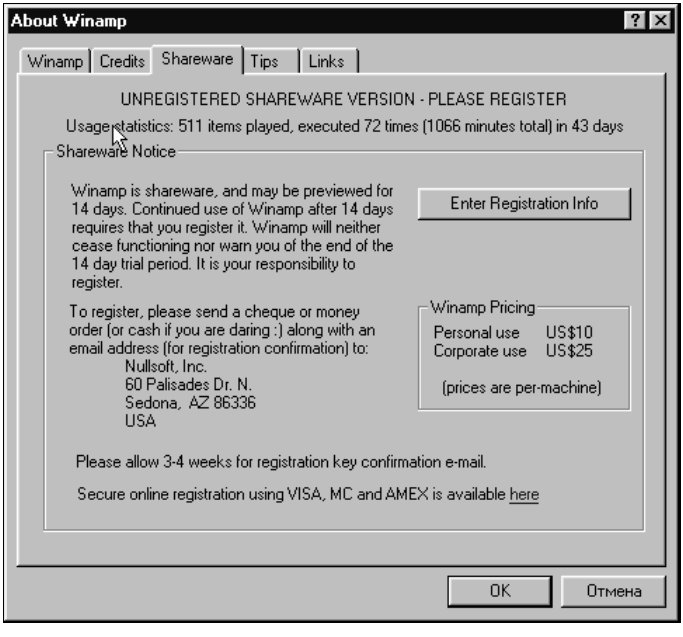


Рис. 7.21. Программа Winamp накапливает статистику своего использования

Декодирование файлов формата MP3

Для декодирования звуковых файлов формата MP3 и их сохранения, например, в формате WAV можно воспользоваться специальными программами-декодерами. Однако многие программы-проигрыватели файлов MP3 обладают возможностью такого преобразования. В частности, таким свойством обладает программа Winamp.

Для того чтобы преобразовать звуковой файл MP3 в формат WAV необходимо сделать следующее.

- В меню **Options/Preferences** выберите вкладку **Audio I/O**.
- В окне **Output Plug-Ins** выберите строчку Nullsoft Disk Writer Plug-In... (рис. 7.22).
- Нажмите кнопку **Configure** и выберите имя каталога, в котором будут сохраняться создаваемые файлы.
- Запустите режим воспроизведения. Вместо воспроизведения звука будут создаваться файлы формата WAV.

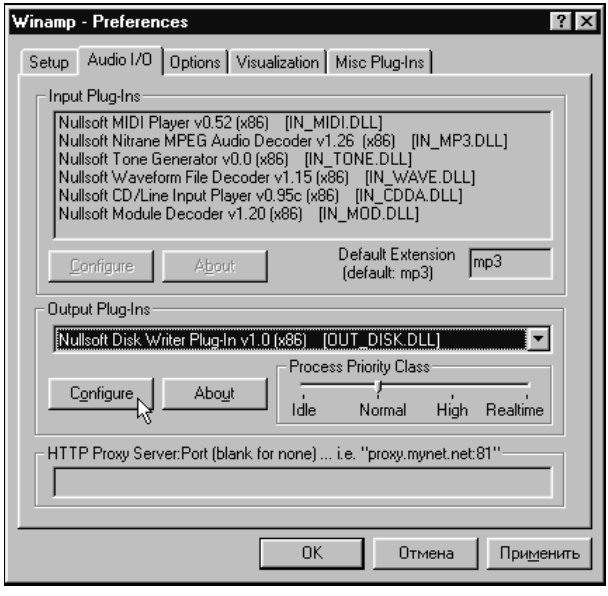


Рис. 7.22. Вкладка **Audio I/O** окна **Winamp/Preferences**

Предупреждение

Программа Winamp сохраняет сделанные вами настройки. Если в следующий раз вы захотите прослушать фонограмму вместо преобразования файла, то нужно не забыть вернуться к исходным установкам. Процесс преобразования внешне выглядит точно так же, как и воспроизведения. Даже модули визуализации в процессе преобразования будут работать, однако звука не будет слышно.

Где найти файлы MP3?

В последнее время хранение музыкальных записей в формате MP3 получило очень широкое распространение. В сети Интернет уже существует довольно большое количество сайтов, посвященных коллекциям фонограмм в этом формате, причем их число постоянно растет. Найти сайты не представляет труда, воспользовавшись любой поисковой системой. Для этого достаточно для поиска ввести ключевое слово "MP3". Заметим, что таким же образом можно найти и коллекции в других форматах, описанных ниже (например, VQF). Что же касается нового формата хранения звука MP4, то о нем пока ничего не слышно, поэтому поиск по ключу "MP4" приведет к чему угодно, но не к музыкальным сайтам. Приведем адреса лишь некоторых наиболее интересных сайтов.

На рис. 7.23 показана одна из страничек сети Интернет, посвященная музыке (**http://www.omen.orc.ru/fl8.htm**).

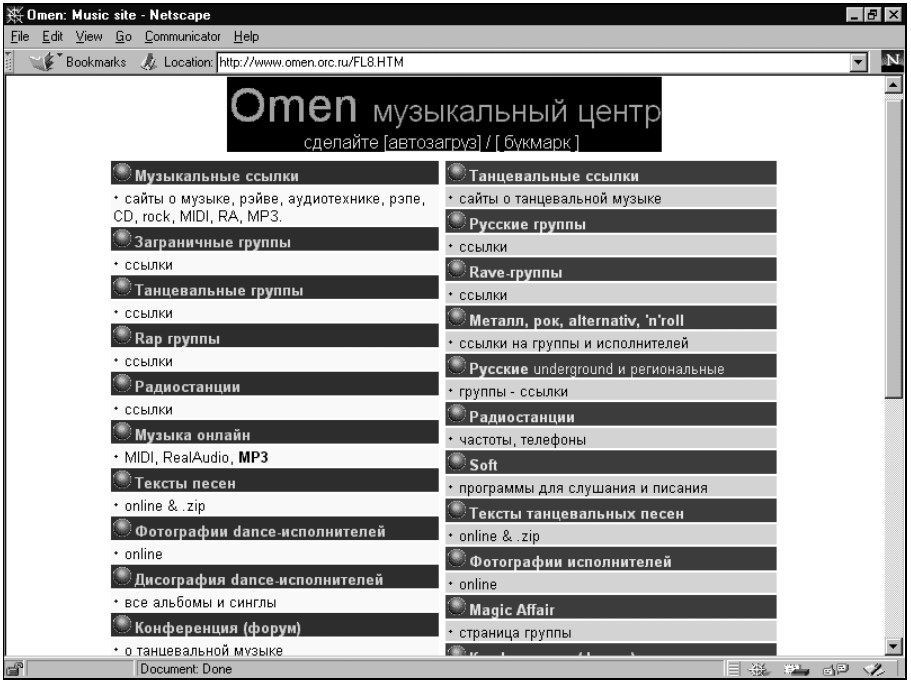


Рис. 7.23. Одна из многочисленных русскоязычных страничек, посвященных музыкальным ресурсам в сети Интернет

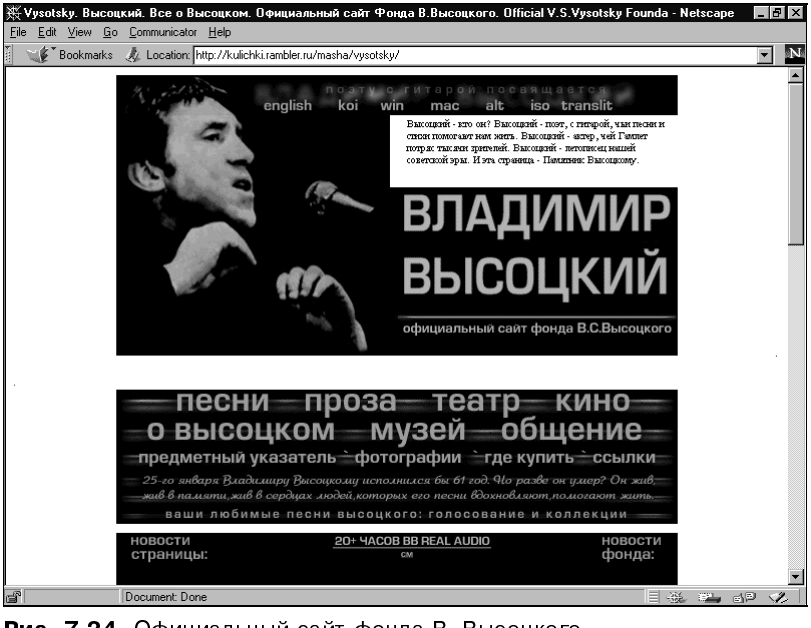


Рис. 7.24. Официальный сайт фонда В. Высоцкого

Множество сайтов на русском языке посвящено авторской песне, в частности, отдельным исполнителям. Например, можно найти страничку, целиком посвященную В. Высоцкому (рис. 7.24). Адрес сайта **http://kulichki.rambler.ru /masha/vysotsky**.

Приведем список некоторых наиболее известных русскоязычных сайтов с музыкальными коллекциями:

http://www.garret.ru/mp3/

http://www.user.cityline.ru/~taxerppp/taxerppp.htm

http://ksp.edison.ru/sound/

http://audio.bard-cafe.komkon.org/

http://bards.net.ru/kfti/

http://soft.hardware.ru/ap/

Создание звуковых файлов MP3

Оказывается, что создать звуковые файлы формата MP3 может практически каждый в домашних условиях. Для этого необходимо иметь источник аудиоданных (обычно это аудио CD) и пару программ.

Создание файла выполняется в два почти независимых этапа. На первом этапе следует получить файл формата WAV (стандартного звукового файла Microsoft Windows), а на втором этапе выполнить его кодирование в формат MP3. Эти этапы можно проделать непосредственно друг за другом, но можно и разделить промежутком времени. Что они в себя включают?

Входным источником для большинства программ-кодировщиков служат WAV-файлы. Некоторые из кодировщиков (для UNIX и OS Macintosh) поддерживают и другие входные форматы. Информация же на CD содержится в специфическом "звуковом" формате в соответствии со стандартом ISO-9680 для записи аудио CD. Перед кодировкой необходимо вычленить такой WAV-файл, что и является первым этапом. Для считывания музыки с аудио CD можно, конечно, прогнать звук через звуковую карту и, воспользовавшись каким-либо звуковым редактором, сохранить данные в виде WAV-файла. Однако это наихудший путь для вычленения звука. Кроме музыки будет сохранен весь набор шумов, который только может породить звуковая карта. Использование звуковой карты оправдано только для тех случаев, когда нет возможности взять музыку непосредственно с CD, например, при записи с радио, магнитофона и т. д. Для считывания музыки с CD пользуются специальными программами, способными перевести данные из аудиоформата в WAV-файл без использования звуковой карты. Такие программы носят название "рипперы" (rippers). В переводе на русский язык для их обозначения используются также термины "нарезальщик", "сдиратель", "грабилка" и др. В последнее время таких программ появилось довольно много, причем сре-

ди них есть условно-бесплатные и вовсе бесплатные. Их обилие определяется в основном тем, что каждая программа поддерживает определенный набор моделей CD, с которым можно ознакомиться в описании программы.

Заметим, что далеко не все приводы компакт-дисков поддерживают операцию чтения аудиодорожек (Reading raw audio stream). Для выбора программы-риппера, просмотрите таблицу, приведенную в приложении 3, и определите главное: поддерживает ли вообще ваша модель CD-ROM такое считывание, и если да, то каким из нарезальщиков рекомендуется пользоваться.

Программы вычленения файлов с аудио CD

Одной из наиболее популярных программ-рипперов является WinDAC (Windows Digital Audio Copy). Эта программа поддерживает большое количество приводов CD-ROM, в частности все модели с интерфейсом SCSI. По адресу **http://www.windac.de/** можно получить полную версию программы.

Второе место по популярности занимает программа AudioGrabber. Эта программа не умеет работать с драйверами SCSI, однако прекрасно работает с устройствами интерфейса IDE. Информацию о программе можно получить по адресам **http://www.audiograbber.com-us.net**, **http://audiograbber.da.ru/**.

Менее популярной является программа CDCopy, работающая в Windows (**http://www.cdcopy.sk/**). Среди программ, работающих в среде DOS, следует упомянуть CDDA (CD Digital Audio) и CD2WAV.

Информацию о работе с этими программами можно получить на сайтах **http://www.mp3.com**, **http://www.multimania.com**. Из русскоязычных сайтов можно рекомендовать обратиться к страничке **http://members.xoom.com/kandid/**.

Программы кодирования

Остановимся теперь на программах-кодировщиках. Эти программы на входе принимают WAV-файл и выполняют его кодирование в формат MP3. Часть программ позволяет работать из командной строки, что дает возможность выполнять их запуск непосредственно из программ-рипперов после считывания очередного трека. Такой возможностью, например, обладает программа-риппер WinDAC, которая имеет механизм написания и запуска скриптов. Некоторые из программ-кодировщиков могут принимать поток данных, поступающих от программы-риппера, и кодировать его без создания промежуточных WAV-файлов. Появились также программы, представляющие собой объединение кодировщика вместе с риппером (к числу таких можно отнести программу CDex).

Несмотря на большое разнообразие программ-кодировщиков, многие из них используют одни и те же библиотеки, т. е. в сущности, являются программами-оболочками, предоставляющие удобный интерфейс к кодировщикам, работающим из командной строки, или установленным в системе кодерам.

Программы такого рода (их называют front-end-программами) передают введенные параметры непосредственно кодировщику и ожидают завершения процесса кодирования. Сами кодировщики называют двигателями кодирования.

Разнообразных двигателей кодирования, в отличие от программ, не так уж и много. Прежде всего, нужно выделить кодировщики, разработанные немецким институтом Fraunhofer IIS (Institut Integrierte Schaltungen), варианты которых используются в программах MP3 Producer, MP3 Compressor, l3enc, mp3enc, AudioActive. Институт Fraunhofer IIS (рис. 7.25), в стенах которого выполнены основополагающие разработки систем сжатия звука, является признанным лидером среди научно-исследовательских лабораторий в области мультимедийных приложений (**http://www.iis.fhg.de**).

Одной из первых программ-кодировщиков является l3enc, 16-битовая программа, работающая в среде DOS. Другая программа mp3enc — дальнейшее развитие l3enc для 32-битовых систем, все также работающая из командной строки, выполняет кодирование значительно быстрее, однако проявляет себя хуже в плане сохранения высоких частот.

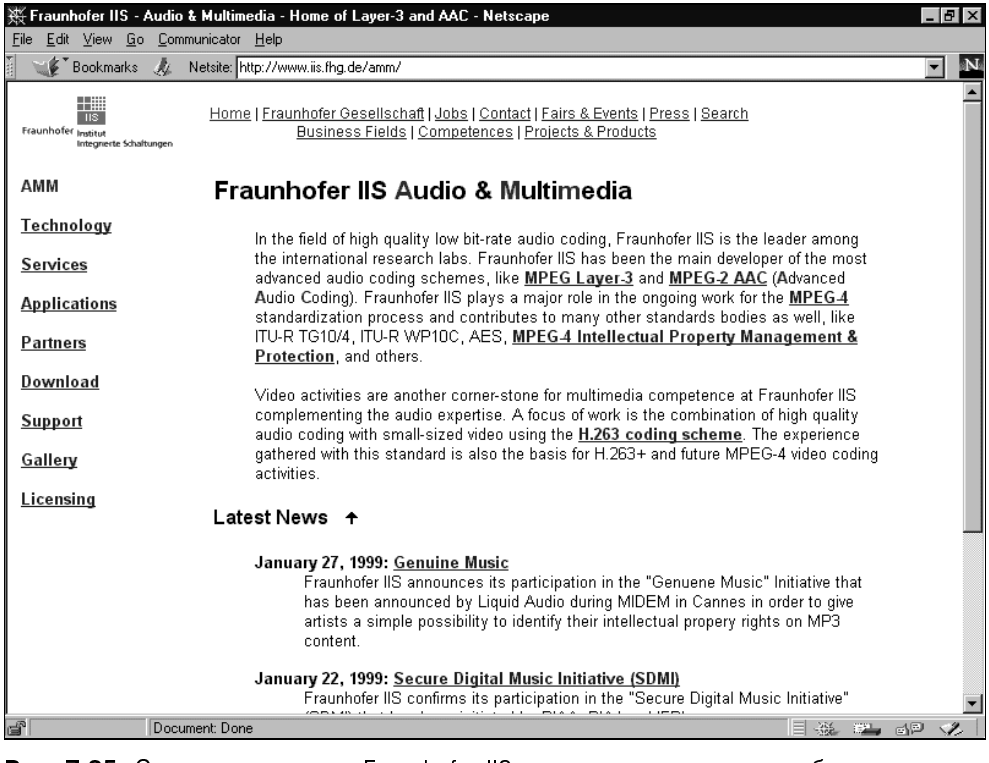


Рис. 7.25. Страница института Fraunhofer IIS, известного своими разработками в области сжатия аудиоданных

Программа MP3 Producer (рис. 7.26) считается лучшим кодировщиком (по мнению авторов сайта **www.mp3bench.com**). При установке этой программы в систему Windows становится доступен кодер MP3, который может затем использоваться из любой программы работы со звуком. Например, можно воспользоваться стандартной программой Фонограф (Sound Recorder) и сохранять файлы в формате MP3. Список установленных в системе кодеров можно увидеть, выбрав нужный пункт в панели управления (**Control panel /Multimedia/Audio compression**).

Примечание

Последние версии MP3 Producer устанавливают в систему Windows только декодер файлов MP3, а код кодера интегрирован с самой программой.

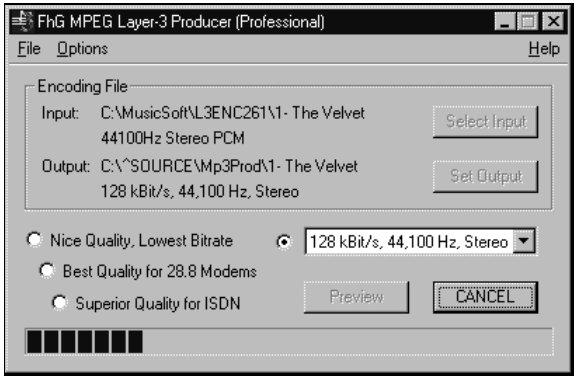


Рис. 7.26. Окно программы-кодировщика MP3 Producer

Программа MP3 Compressor является оболочкой-интерфейсом к тому же кодеру, на котором основан и MP3 Producer. Интерфейс программы позволяет несколько иначе строить задачи кодирования и следить за их процессом. Это приложение предоставляет некоторые дополнительные возможности, не допускаемые в MP3 Producer. В частности, поддерживает работу в режиме LQ (режим, противоположный HQ — High Quality). При включенной опции LQ (Low Quality) кодировщик производит над каждым фрагментом некоторое фиксированное количество итераций, затрачивая на кодировку фрагмента определенное количество времени. Если же активизировать HQ, кодировщик будет продолжать итерационный процесс до тех пор, пока не перестанет улучшаться результат. Повышение качества в этом случае требует дополнительных временных затрат на кодировку, при этом размер выходного файла не меняется. Время кодирования при изменении режима HQ/LQ может изменяться в 2—3 раза.

Другим несомненным достоинством программы является возможность задания большого количества файлов для кодировки и запуска программы из командной строки.

Примечание

Последние версии программы MP3 Producer (начиная с версии 2.1) также предоставляют возможность отключения режима HQ, а также имеют опцию пакетной обработки файлов (Batch Processor). Кроме того, теперь предоставляется возможность задания битрейта вплоть до 256 Кбит/с для высококачественных записей (ранние версии имели ограничение — до 128 Кбит/с).

Другой известный производитель коммерческих кодеров MP3, компания Xing Tech (Xing Technology Corporation, адрес сайта корпорации **http://www.xingtech.com**), специализируется на выпуске низкокачественных кодеров. Среди этой группы кодеров можно назвать XingMP3 Encoder, Mplifier, AudioCatalyst и др. Характерной особенностью кодировщиков от Xing Tech является высокая скорость работы (программа XingMP3 Encoder выполняет кодирование в 6—8 раз быстрее, чем MP3 Producer). Такая скорость достигается за счет безжалостного вырезания частот свыше 16 кГц, которые еще слышны человеку. В итоге, когда речь заходит о высоком качестве, кодировщики Xing Tech использованы быть не могут.

Примечание

Программа AudioCatalyst представляет собой объединение программы-риппера (на базе AudioGrabber) и кодировщика от Xing Tech.

У кодировщиков данной группы (от Xing Tech) есть и некоторые интересные свойства. В последней версии кодировщика AudioCatalyst 1.5 реализована довольно интересная техника переменного битрейта (VBR, Variable Bit Rate). В результате кодер способен для того же суммарного объема в определенных ситуациях давать MP3-файлы даже более высокого качества, чем кодеры от Fraunhofer IIS, так как для сложно кодируемых участков становится возможным использование повышенных битрейтов, а для легко кодируемых, соответственно, — пониженных. Несомненным достоинством техники VBR-кодирования является то, что она базируется на внутренних возможностях формата MP3, не добавляя по сути ничего нового — просто эти возможности в полной мере не были использованы ранее. Соответственно, полученные MP3-файлы можно проиграть на любом MP3-плейере. Проблемы могут возникнуть иногда только при определении длительности звучания композиции, но на качество и полноценность воспроизведения это не влияет. Полностью технология переменного битрейта поддерживается программой Winamp, начиная с версии 2.09.

Все остальные известные кодеры, авторство которых не принадлежит ни Fraunhofer IIS, ни Xing Tech, основаны на общедоступных исходных текстах кодера (ISO source), равномерно кодирующего разные частотные диапазоны, что приводит к наивысшему качеству кодирования на высоких битрейтах и вполне приличному на 128 Кбит/с. Но этот код достаточно медленный, поэтому существует единственный известный кодер, в котором он не подвергся обширным изменениям в целях оптимизации — mpegEnc. Эта про-

грамма считается эталоном качества для высших битрейтов. Из кодировщиков этой группы достойны также упоминания BladeEnc, Cdex и SoundLimit.

Выбор кодировщика в конечном итоге определяется требованиями к качеству и времени кодирования. Для большинства любителей музыки среднего качества вполне достаточно возможностей кодировщика AudioCatalyst 1.5. Он обеспечивает наивысшую скорость кодирования и при этом дает вполне удовлетворительное качество звучания. Для обеспечения высшего качества придется прибегнуть к большим битрейтам и воспользоваться кодировщиками типа BladeEnc или mpegEnc. Из кодировщиков от Fraunhofer IIS, кроме популярных программ MP3 Producer и MP3 Compressor, рекомендуется опробовать их последнюю разработку — программу AudioActive.

Информацию о программном обеспечении для работы со звуковыми файлами можно получить на сайтах:

http://www.dailymp3.com
http://www.mp3.com
http://www.mp3bench.com
http://www.mp3-2000.com
http://www.hugemp3.com
http://www.direct-mp3.com
http://www.mp3now.com
http://www.mp3archive.com
http://www.mp3place.com
http://www.mp3place.net.

Выбор параметров кодирования

Значительное количество встречающихся в сети Интернет файлов MP3 имеют частоту оцифровки 44,1 кГц, битрейт 128 Кбит/с и режим стерео. Считается, что такие параметры звуковых файлов обеспечивают качество, сопоставимое с аудио CD. На эти параметры и следует ориентироваться при подготовке звуковых файлов для дальнейшего прослушивания на аппаратуре среднего класса.

Примечание

На самом деле звучание файлов MP3 становится невозможно отличить от аудио CD только на битрейтах 256 Кбит/с и выше, что подтверждено профессиональными прослушивателями, нанятыми в свое время институтом Fraunhofer IIS. Однако использование битрейта 128 Кбит/с вместо 256 Кбит/с позволяет в два раза уменьшить размер файлов, что играет значительную роль при передаче по сети. При этом искажения сигнала можно ощутить только на высококлассной аппаратуре и то не на всякой фонограмме. Учитывая эти особенности принято считать, что 128 Кбит/с обеспечивает качество аудио CD.

При кодировании можно поступиться качеством и выиграть в размере получаемых файлов (в зависимости от предназначения звуковых файлов). Для выбора параметров кодирования можно рекомендовать обратиться к табл. 7.6, по которой в зависимости от требуемого качества звука можно выбрать

нужные параметры кодирования (важнейшим параметром для кодировщика является значение битрейта).

Таблица 7.6. *Приблизительные характеристики качества звука для различных режимов*

Качество звука	Частотный диапазон, кГц	Режим	Битрейт, Кбит/с	Степень сжатия
Телефонная линия	2,5	моно	8	96:1
Лучше, чем LW-радио	4,5	моно	16	48:1
Лучше, чем AM-радио	7,5	моно	32	24:1
FM-радио	11	стерео	56—64	26-24:1
Почти CD-качество	15	стерео	96	16:1
CD-качество	15	стерео	112—128	14-12:1

Заметим, что процесс кодирования достаточно длителен. Так, например, кодирование 4 минут музыки на Pentium-200 MMX программой MP3 Producer в режиме HQ с битрейтом 128 Кбит/с занимает примерно 12 минут процессорного времени в отсутствие других задач.

Переносные плейеры звуковых файлов MP3

Прочитав о возможностях хранения звука в формате MP3, у проницательного читателя должен появиться вопрос, не пора ли начать выпускать переносные устройства проигрывания (наподобие популярных уже на протяжении двух десятилетий аудиоплейеров). Спешим вас обрадовать, такие устройства уже существуют! Одни устройства предназначены для прослушивания файлов "на улице", другие же — для установки в автомобиле. Их можно разделить по принципу хранения данных — флэш-память, жесткие диски PC-card type I и II, а также способу расшифровки файлов — специальный процессор, предназначенный только для этих целей, или любой достаточно мощный RISC-процессор с низким энергопотреблением.

Одним из первых переносных плейеров является MPMan, точной копией которого является плейер MPStation (**http://www.mpstation.com**) фирмы DigitalCast. Компания Z Company (которой, кстати, принадлежит сайт **www.mp3.com**) уже начала продажу в США плейеров MPMan (**http://www.mpman.com**), которые производятся корейской фирмой Saehan Information Systems.

Плейер MPMan имеет размер с пачку сигарет (91×70×16,5 мм), весит всего 65 г (без батареек), не содержит движущихся частей и выпускается в двух вариантах. Модель с 32 Мб флэш-памяти может вместить до 30 минут записи, а модель с 64 Мб, соответственно, — один час. Питание осуществляется

от двух аккумуляторных батарей напряжением 1,2 В, емкости которых хватает на 9 часов звучания. Плейер может устанавливаться в специальное устройство (Docking Station), оборудованное зарядным устройством и подключаться к компьютеру через стандартный параллельный порт.

Один из крупнейших производителей аппаратуры для персональных компьютеров — компания Diamond Multimedia Systems выпустила свой портативный цифровой музыкальный плейер Rio PMP300 — первое устройство от крупной фирмы, способное воспроизводить записи, загруженные из Интернета. Плейер также имеет небольшие размеры (89×63×16 мм), весит всего 70 г. Данная модель имеет флэш-память емкостью 32 Мб с возможностью увеличения. Может подключаться к компьютеру через параллельный порт. Время заполнения флэш-карты не превышает 6 мин. Емкости батареек хватает на 12 часов непрерывного прослушивания.

Вице-президент компании Diamond Multimedia Systems по корпоративному маркетингу Кен Вирт (Ken Wirt) считает, что этот продукт станет очень популярным. Компания планирует в ближайшее время довести объем производства до 10 тыс. штук в неделю. Это устройство размером меньше плейера Walkman обещает произвести переворот в музыкальной индустрии и приблизить эру цифрового распространения записей, предоставив больше возможностей независимым музыкальным группам.

Видимо в ближайшее время произойдет резкий рост количества компаний, производящих подобную технику. В частности, компания Samsung объявила о завершении разработки своего плейера с названием "Yepp", выпускаемого в трех вариантах (от простого к сложному). Отличительными характеристиками этого плейера является наличие встроенного радиоприемника FM-диапазона с цифровой настройкой, а также возможность записи голоса, т. е. это устройство может использоваться в качестве диктофона. Для записи файлов формата MP3 предлагается отдельное устройство. Сведения о этих разработках можно получить на специальном сайте **http://www.yepp.co.kr**.

Еще один переносной плейер — Clikman. Помимо основных функций в это устройство встроили органайзер для записи телефонов, расписания и т. д. Управление устройством осуществляется как при помощи встроенной цифровой клавиатуры, так и голосовыми командами. Для хранения всех данных используются диски Clik! от накопителей ZIP-Iomega емкостью 40 Мб. Все перечисленные возможности потребовали усложнения устройства, в результате его размеры и вес несколько превышают соответствующие параметры аналогичных устройств (вес составляет почти 200 г). Ознакомиться с новинкой более подробно можно непосредственно на сайте компании-производителя: Varo Vision Co (**http://www.varovision.com/sub1/clik.html**).

Однако, технологии использования формата MP3, по крайней мере, для крупной музыкальной индустрии свойственен один существенный недостаток, а именно, отсутствие средств защиты от копирования. Каждый желаю-

щий может распространять в Интернете любую преобразованную в этот формат песню: чтобы из одной копии сделать миллион, достаточно нажать на несколько клавиш. Это не нравится Американской ассоциации индустрии звукозаписи (RIAA — Recording Industry Association of America), представляющей около 90% этого бизнеса (**http://www.riaa.com**). Старший исполнительный вице-президент и генеральный юрисконсульт RIAA Кэри Шерман (Cary Sherman) считает, что, бесплатно распространяя пиратские записи, новое устройство способно полностью разрушить индустрию.

Сейчас большая часть музыкальных записей MP3 в Интернете — это пиратские копии. RIAA называет эту разработку шагом не вперед, а назад.

Согласно заявлению ее представителей, Ассоциация видит очень мало легальных применений MP3 по сравнению с множеством вариантов пиратского использования. Компания также рассматривает возможность запрета импорта устройств типа MPMan в США.

Встроить в плейер технологию флэш-памяти легко могли и другие компании, не менее смышленые, чем Diamond. Но они ожидали появления технологии защиты содержимого. С выпуском плейера Rio ситуация изменилась. Почему Sony должна ждать, если Diamond уже делает деньги? Эти слова Кэри Шермана оказались пророческими. Боясь отстать, Sony продемонстрировала на выставке Comdex прототипы подобных изделий. Это вызовет эффект домино, вы не успеете оглянуться, как артисты останутся абсолютно беззащитными. Опасения такого рода не беспочвенны. Интернет способен коренным образом изменить соотношение сил в музыкальной индустрии, что уже произошло в информационной технологии в целом.

Споры о нарушении авторских прав при использовании цифровых плейеров дошли до суда. В результате эти электронные устройства признаны американским судом не относящимися к записывающим устройствам и потому не нарушающими авторские права. Суд Лос-Анджелеса в октябре 1998 г. разрешил производство и продажу устройства Rio и отказал истцу — Американской ассоциации индустрии звукозаписи в требовании к производителю прибора, фирме Diamond Multimedia, отчислять им проценты с продаж за "копирование музыкального материала". По оценке Роберта Кона, президента интернетовского сайта GoodNoise, предлагающего посетителям тысячи музыкальных записей, это решение суда "является большой победой молодых исполнителей, которые получают альтернативный способ распространения своих произведений".

Влияние подобных устройств еще только осознается шоу-бизнесом. Они могут буквально перевернуть все аспекты работы музыкальной промышленности, включая практику распространения и лицензирования копий. Например, группы начнут конкурировать в реальном времени, выпуская не диск за диском, а песенку за песенкой, публикуя особо удачные концертные исполнения, и т. д.

Легко представить себе, например, как трансформируется существующая в российских городах инфраструктура распространения популярной музыки среди подростков. Киоски с кассетами превратятся в пункты записи звуковых файлов со своими каналами доступа к Интернету и прокси-серверами. Успех у публики (а не у диск-жокеев радиостанций) станет доступен непосредственному измерению. Опубликовать свою запись через сеть Интернет сможет буквально любой, что правда может привести к засилию низкопробной продукции.

Потоковое воспроизведение

Технология сжатия звуковых файлов MP3 постепенно проникает во все сферы использования звука. С развитием каналов передачи данных и возможностей сжатия эта технология вышла на уровень потоковой передачи данных в реальном времени, где до сих пор безраздельно господствовала технология RealAudio. Компания Nullsoft (**http://www.nullsoft.com/**), создатель знаменитого плейера Winamp теперь, кроме продвижения самого плейера, активно занимается развитием и пропагандой технологии ShoutCast, предназначенной для потоковой передачи аудиоданных в формате MP3. Единственное, что нужно обычному пользователю для того, чтобы послушать сетевую MP3-станцию, или открыть собственную, это сам плейер Winamp. По адресу **http://www.shoutcast.com/** можно найти адреса более чем 300 серверов, вещающих в формате MP3, и список десяти лучших из них.

Звуковые файлы формата VQF

Популярность и широкое распространение технологий MP3 вызвали бурный рост числа разработок в области сжатия звука. Стали появляться новые стандарты и форматы хранения звука, которые в будущем, возможно, придут на смену существующим. В последующих разделах кратко остановимся на современных достижениях в этой области.

Наряду с широко известным на сегодняшний день форматом MP3 сравнительно недавно появился формат VQF, имеющий более высокую степень сжатия и качество воспроизведения.

Формат VQF основывается на технологии TwinVQ (Transform-domain Weighted Interleave Vector Quantization — векторное квантование с трансформными доменами и взвешенным чередованием), разработанной в Японии в лаборатории NTT Human Interface Laboratories (**http://www.hil.ntt.co.jp/top/index_e.html**). Патент на использование этого формата принадлежит фирме NTT, однако основное программное обеспечение для работы с этим форматом предлагает фирма Yamaha. Компания Yamaha предлагает, пожалуй, лучшее программное обеспечение для создания и воспроизведения файлов формата VQF под маркой SoundVQ (**http://www.yamaha.co.jp/english/xg**

/SoundVQ/). При установке плейера одновременно инсталлируются подключаемые модули для воспроизведения файлов VQF в браузеры Netscape и Internet Explorer.

Существует также и программное обеспечение от компании разработчика технологии TwinVQ (**http://music.jpn.net/software-e.html**).

Этот формат по использованию идей сжатия весьма похож на MP3. Приведем их сравнительные характеристики.

VQF-файлы примерно на 30—35% меньше, чем MP3, при одинаковом качестве звука. Потоку 128 Кбит/с у файлов MP3 соответствует поток 80 Кбит/с у файлов VQF. У этих достоинств есть и обратная сторона. При декодировании загрузка процессора также примерно на 30% выше, чем при декодировании MP3. Это определяет повышенные требования к компьютеру, на котором планируется проигрывать такие файлы. Для качественного воспроизведения минимальным требованием является процессор Pentium-90, рекомендуется Pentium-200 MMX. Для кодирования минимальным требованием является компьютер с процессором Pentium-66. Высокая степень сжатия требует значительных временных затрат на кодирование файлов в формате VQF. Так кодирование 4 минут музыки на Pentium-200 MMX занимает примерно 20 минут процессорного времени. Это значительно больше, чем при кодировании файлов MP3.

Тесты показывают превосходство VQF по всем параметрам на нижних частотах и гораздо меньшее искажение формы сигнала с большим динамическим диапазоном (реальная музыка). Однако по завалу верхних частот звукового спектра VQF на 2—3 дБ уступает MP3 на частотах выше 15 кГц. Это впрочем, легко компенсируется настройкой эквалайзера плейера, что объективно ставит VQF на ступень выше по качеству звука по сравнению с MP3, но субъективное восприятие различных композиций формата VQF дало старт затяжной полемике между приверженцами хорошо проверенного старого (MP3) и новинок (формата VQF и формата AAC, о котором пойдет речь далее). Узкопрофессиональные споры не мешают сделать широкой интернетовской общественности выбор в пользу нового формата. Наиболее распространенное мнение сейчас можно сформулировать примерно так: "Если и есть какие-то различия, то нужно быть профессиональным музыкантом, чтобы их заметить".

На текущий момент в сети Интернет можно найти довольно много музыкальных произведений в формате VQF, однако их количество и степень популярности значительно уступают MP3. Также значительно меньше существует программ-плейеров и кодеров.

Для проигрывания файлов можно рекомендовать программу K-Jofol. Появились также подключаемые модули для Winamp, проигрывающие файлы VQF (этот модуль использует декодер компании Yamaha).

Программа K-Jofol версии 0.51 занимает 1,4 Мб (как раз одна дискетка). Позволяет воспроизводить файлы форматов VQF, AAC, MP3 и др. Дает воз-

можность декодировать файлы в формат WAV. На сегодняшний день считается самым быстрым плейером, использующим свои собственные декодеры для всех форматов.

Программе характерен оригинальный дизайн. Уже при инсталляции можно заметить (рис. 7.27), что авторы попытались уйти от традиционных прямоугольных окон, придав им вычурные формы (рис. 7.28, 7.29). На первых порах немного непривычно, однако довольно быстро удается приспособиться. Возможности и методы работы с программой во многом аналогичны другим плейерам.

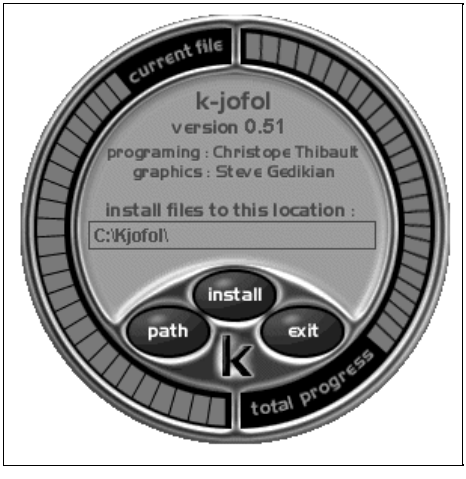


Рис. 7.27. Окно инсталлятора программы K-Jofol

Рис. 7.28. Основное окно программы K-Jofol

Рис. 7.29. Список проигрываемых файлов программы K-Jofol

Информацию о формате VQF можно получить с официального сайта **http://www.vqf.com**. О программе K-Jofol можно узнать на сайте **http://www.kjofol.org**. Найти иную информацию в Интернете, а также файлы в этом формате, можно с помощью любой поисковой системы, указав для поиска слово "vqf". О быстром росте популярности этого формата свидетельствует следующий факт. Если в сентябре 1998 г. запрос по слову "vqf" поисковой системе AltaVista давал всего 20 ссылок, то в начале 1999 г. по такому запросу выдавалось уже несколько тысяч ссылок.

Примечание

Популярность формата VQF пока значительно уступает MP3. Рынок программного обеспечения работы с такими звуковыми файлами еще не вполне устоялся. Частенько появляются сообщения о весьма неустойчивой работе многих программ-проигрывателей этого формата. Это относится и к подключаемым модулям программы Winamp (попадаются просто неработоспособные версии), так и программе K-Jofol. Если у вас воспроизведение файлов такого формата идет с явными искажениями (даже на достаточно мощной машине), попробуйте прежде всего обновить версию программы-проигрывателя или используйте другую программу. Только во вторую очередь следует грешить на сам звуковой файл.

Звуковые файлы формата AAC

Формат AAC (Advanced Audio Coding) представляет еще один из числа появившихся сравнительно форматов кодирования звуковой информации, в котором осуществляется сжатие с потерями. На сегодняшний момент AAC представляет собой лучшую систему сжатия высококачественного звука. Это последний из стандартов MPEG-2. Перевести AAC можно как "Продвинутое Аудио Кодирование", название, которое появилось из-за исключительно высокого качества звучания и очень сильной системы сжатия.

На данный момент различают четыре разновидности формата AAC:

- HomeboyAAC
- AT&T a2bAAC

- Liquifier Pro AAC
- Astrid/Quartex AAC

По всем объективным параметрам последние две модификации AAC превосходят как MP3, так и VQF.

Все четыре разновидности несовместимы между собой и используют свои собственные программы кодирования и воспроизведения.

Далее мы кратко их охарактеризуем.

HomeboyAAC

Это первая версия AAC, кодер для которой стал общедоступен. Ее можно охарактеризовать одним словом — отвратительно. Крайне корявые и медленные кодеры в сочетании с отчетливо слышимыми искажениями сыграли свою роль в отталкивании широких масс от этого формата. Единственными его плюсами является достижение качества MP3 128 Кбит/с на более низких битрейтах и наличие приятного и быстрого плейера BitAAC. Сейчас этот формат уже практически не используется и даже ссылки на программное обеспечение для него почти исчезли со страниц Интернета. Следует иметь в виду, что большинство программ, распространяемых через Интернет под видом кодеров AAC, являются кодерами именно этого формата. Во времена появления и развития этой ветви AAC уже существовал формат VQF.

AT&T a2bAAC

Известная компания AT&T не могла остаться в стороне от технологий новых систем сжатия аудиоданных. При фирме было создано специальное подразделение, занимающееся созданием программ и раскруткой нового формата, который получил название a2b. Строго говоря, этот формат практически полностью следует тем же самым принципам компрессии, что и предыдущий. Однако фирмой были предприняты значительные усилия по повышению качества звучания (в результате чего снизилась степень сжатия) и внесению новых элементов в формат. Составной частью формата a2b стала возможность включения изображения исполнителя и текста песни внутрь аудиофайла. Также была разработана технология получения самовоспроизводящихся песен (исходную композицию можно преобразовать в .exe-файл, включающий в себя все необходимые для воспроизведения, что увеличивало файл всего примерно на 170 Кб). Для файлов этого формата существует удобный и бесплатный плейер с поддержкой многих дополнительных возможностей. Однако, несмотря на мощнейшую рекламную компанию и на очевидные достоинства формата, он не нашел широкого применения по одной простой причине — отсутствию общедоступного кодера! Мало того, эта уважаемая фирма пыталась запатентовать и запретить к использованию все составные части формата. К счастью, это у нее не вышло, — патент был по-

лучен только на собственные нововведения. Сейчас уже рекламный пыл немного угас и со своего сайта компания предлагает приобрести уже готовые файлы или бесплатно скачать несколько композиций.

Формат по степени сжатия превосходит MP3, но на 15—20% уступает VQF, PAC, Astrid/Quartex AAC и Liquifier Pro AAC.

Качество звучания a2b 96 Кбит/с сравнимо с качеством MP3 128 Кбит/с и VQF 96 Кбит/с, но уступает Liquifier Pro AAC 96, Astrid/Quartex AAC 96 и PAC 96.

Дополнительную информацию можно получить в Интернете по адресу **http://www.a2bmusic.com/**.

Liquifier Pro AAC

Естественно компания AT&T не могла остаться одинокой на рынке борьбы за первенство в области высококачественной компрессии звука. Через некоторое время фирма Liquid Audio нанесла если не сокрушительный, то очень опасный удар по честолюбивым замыслам остальных. Предложенная ей версия формата AAC наиболее четко следовала всем техническим тонкостям и в результате появилась лучшая на сегодняшний день система сжатия звука. Она по степени сжатия и качеству звучания превосходит все существующие в данный момент форматы. И хотя превосходство по сжатию над VQF и Astrid/Quartex AAC незначительное — минимальный битрейт, заслуживающий внимания, все те же 96 Кбит/с, звук превосходит все ожидания!

К сожалению, политика компании Liquid Audio на данный момент слабо отличается от политики AT&T — есть отличный плейер, но нет общедоступного кодера (существовала демо-версия, но она работала только в онлайне и через несколько дней неожиданно самоуничтожалась, а все закодированные ей файлы становились неработоспособными).

Подводя итоги, заметим, что этот формат по всем параметрам превосходит все остальные и объективно является лучшим. Среди самых последних новостей отметим появление звуковых файлов в этом формате (они имеют расширение LQT), причем для их воспроизведения можно воспользоваться плейером компании Liquid Audio или же установить plugin для программы Winamp (in_lqt.dll), который на самом деле пользуется все тем же, несколько "исправленным" плейером, перенаправляя звук в программу Winamp.

Дополнительную информацию можно получить в Интернете по адресу **http://www.liquidaudio.com/**.

Astrid/Quartex AAC

В октябре 1998 г. в "спор гигантов" вступило частное лицо. Никому не известный и скромный программист создал, на основе изданных фирмой NTT

в мае исходников MP4, свою собственную систему компрессии! Им было предложено протестировать кодер, для чего он был выложен в Интернете по адресу **http://www.geocities.com/ResearchTriangle/Facility/2141/**.

Уже через несколько дней после выхода новой версии популярного плейера K-Jofol, для которого автор формата Astrid/Quartex AAC написал декодирующий модуль, появились восторженные отклики. Кодер, обладающий непритязательным интерфейсом командной строки, по сжатию и качеству звучания превосходил кодировщики от AT&T AAC и YAMAHA VQF. При этом качество звука практически не уступало хваленому Liquifier Pro AAC. Почти одновременно появились графические интерфейсы (front-end) для нового кодировщика и масса хвалебных отзывов в разных источниках. При всех видимых недостатках сегодняшней версии 0.2 (поддержка только WAV 44 кГц/16 бит PCM и степени сжатия только 64, 96 и 128 Кбит/с), она уже широко применяется (есть целые ftp-сайты с музыкой в этом формате) и все с нетерпением ждут новых версий.

По степени сжатия и качеству звучания формат превосходит все остальные, кроме Liquifier Pro AAC, и является объективно лучшим среди тех, что имеют общедоступные кодеры и доступные для распространения в Интернете звуковые файлы. По сравнению с MP3 обеспечивается аналогичное качество при потоке на 30% меньше. Качество Astrid AAC 96 Кбит/с много лучше качества VQF/96 Кбит/с. Возможно также потоковое воспроизведение.

Звуковые файлы формата PAC

Несколько особняком от остальных систем кодирования стоит формат PAC (Perceptive Audio Coding). Впервые эта аббревиатура появилась в стенах лаборатории Bell Labs/Lucent Technologies (кстати, там же впервые появилось сокращение AAC) в самом конце 1997 года. Через некоторое время стали появляться очень интересные сообщения от других фирм-разработчиков, которые не могли не заинтриговать интересующегося человека.

Так было заявлено о следующих возможностях:

- достижении качества MP3 128 Кбит/с в формате PAC с битрейтом 64 Кбит/с;
- поддержка потокового воспроизведения через Интернет;
- прямое преобразование из Audio-CD в PAC без промежуточных огромных WAV-файлов на диске;
- кодирование в реальном времени на Pentium-166 (даже не MMX);
- защита от свободного распространения PAC-файлов по Интернету и на CD-R;
- широкие возможности упорядочения готовых PAC-файлов (плей-листы, каталоги, описания и т. д.).

В середине лета 1998 г. известная фирма Celestial Technologies выпустила первую и до сих пор единственную программу для кодирования и воспроизведения звука в формате PAC — "Audio Library 1.0". Хотя программа бесплатно работает только 15 дней и позволяет в одном каталоге хранить не более 5-ти композиций, уже можно сделать кое-какие выводы. Первое, что бросается в глаза, это отсутствие, как таковых, файлов в формате PAC. Во время своей работы программа создает базу данных, состоящую из 8 файлов с расширением TPS, в отдельном каталоге, причем для воспроизведения необходимы минимум 7 из них (GREGLANG.TPS можно выкинуть) Увеличение количества композиций не меняет количества файлов — при добавлении новой песни в базу просто увеличивается размер файла SONGDATA.TPS.

И во всем остальном программа удивляет. Хотя пока на битрейте 64 Кбит/с звук уступает MP3 128 Кбит/с, качество звучания PAC 128 Кбит/с поражает воображение. При этом кодер обеспечивает обещанную скорость кодирования.

Большего сказать о данном формате пока не представляется возможным. Нужно подождать новых версий кодеров.

Новую информацию можно получить по адресу **http://www.celestialtech.com/**.

Новый формат хранения звука MP4

Работа над новыми стандартами в области хранения звука не стоит на месте. И вот, наконец, в ноябре 1998 г. все заинтересованные стороны пришли к согласию об окончательной спецификации нового формата сжатия звука MP4. Как и ожидалось, в основе нового формата лежат VQF и AAC.

Появление нового стандарта, однако, не предполагает быстрого появления программ, работающих в этом формате. По самым оптимистичным прогнозам первые кодеры и плейеры появятся к началу-середине лета 1999 г., а более или менее продвинутые, с оптимизацией скоростей кодирования и воспроизведения, программы следует ожидать не ранее 2000-го года. Следовательно, в ближайшее время всем придется пользоваться уже существующими форматами.

Первые разработки по новому формату были представлены в январе 1999 г. компанией Global Music Outlet. Пока рано делать какие-либо выводы о возможностях и перспективах этого формата. Вполне может случиться, что формат с таким названием останется лишь в виде спецификации, а в реальной жизни будут появляться и использоваться другие форматы, предложенные отдельными компаниями.

Вызывает некоторое удивление само название формата — MP4 из-за возможной путаницы с MPEG-4, форматом аудио- и видеокомпрессии, принятым Международной организацией по стандартизации в качестве открытого стандарта распространения мультимедийной информации. Между обеими технологиями нет ничего общего.

Новости о развитии ситуации можно узнать на сайте **http://mp4.globalmusic.com/**. Компания Global Music Outlet обещала, что одной из первых записей в новом формате будут произведения группы Public Enemy, которые будут опубликованы на ее Web-сайте (**http://www.public-enemy.com/**).

Компания также рассчитывает соблазнить "китов" музыкальной индустрии возможностью формата MP4 записывать звуковые файлы с защитой от нелицензионного копирования.

Формат WMA

Компания Microsoft, естественно, не могла обойти вниманием быстро развивающиеся технологии сжатия звуковых данных. Была разработана собственная технология кодирования звука, быстро получившая распространение. Формат соответствующих файлов получил название WMA (изначально он был назван ASF). Быстрому внедрению формата должна способствовать маркетинговая политика компании Microsoft, которая включила поддержку нового формата в свой проигрыватель (Media Player), который можно свободно получить с сайта компании. Также в свободном доступе кодировщик файлов формата WMA.

Поддержка воспроизведения файлов WMA включена в программу Winamp с версии 2.20. Эту же программу можно использовать и в качестве кодировщика.

ГЛАВА 8

Разработка HTML-страниц при помощи текстового процессора Microsoft Word

Создание и изменение HTML-страниц может выполняться с помощью различных средств. В эпоху зарождения и развития Web большинство авторов использовало для этих целей обычные текстовые редакторы. При этом тэги языка HTML добавлялись вручную в ходе редактирования документа. Такой способ вполне допустим и сегодня, однако требует хорошего знания самого языка HTML и значительных затрат времени на набивку и исправление тэгов. Со временем появились специальные средства редактирования HTML-документов, сделавшие работу над документами проще и эффективнее.

Редакторы HTML-документов обычно принято разбивать на две группы. К первой группе относятся так называемые редакторы тэгов, которые предоставляют возможность записывать коды непосредственно на языке HTML и имеют специальные дополнительные возможности для облегчения процесса включения тэгов в создаваемый документ. Современные редакторы тэгов имеют дополнительные инструменты для генерации элементов HTML. К ним относятся программы-мастера, шаблоны и иные средства, упрощающие работу с типичными HTML-элементами типа списков, таблиц, форм и фреймов. Некоторые редакторы проверяют синтаксис записи тэгов, что позволяет быстрее находить ошибки форматирования HTML-документов. Одним из наиболее популярных редакторов этой группы является программа HotDog Web Editor, разработанная компанией Sausage Software. Этот редактор используется разработчиками на протяжении уже многих лет и сильно менялся в процессе своего развития.

К другой группе относятся так называемые редакторы WYSIWYG (What You See Is What You Get — что видишь, то и получаешь), типичными представителями которой являются Netscape Composer (редактор, входящий в состав пакета Netscape Communicator) и Microsoft FrontPage. Редакторы такого рода позволяют изменять внешний вид и компоновку страницы визуально, не вдаваясь в подробности реализации в виде тэгов. При этом результирующий HTML-код документа создается автоматически. При работе с этими редак-

торами разработчик может вообще не представлять правил записи тэгов HTML и целиком положиться на возможности соответствующего редактора. С другой стороны такие редакторы могут не обеспечивать гибкости, свойственной работе на уровне тэгов, и не обладают всеми возможностями разметки языка HTML.

На современном этапе деление редакторов на две группы становится все более и более условным. Редакторы тэгов все более приближаются по своим возможностям и удобству работы к редакторам WYSIWYG, а последние, в свою очередь, становятся все более мощными и гибкими, позволяя включать все более сложные элементы. Скорее всего, в ближайшем будущем деление редакторов на группы исчезнет.

Одним из средств создания и редактирования HTML-документов является текстовый процессор пакета Microsoft Office, носящий название Microsoft Word. В принципе любая версия этой программы, как обычный текстовый редактор, может служить в качестве HTML-редактора (о различиях в возможностях версий пакета говорится в конце этой главы). Удобство его использования для редактирования HTML-документов определяется широким распространением Microsoft Word среди пользователей для создания отчетов, писем, документации, анкет и даже брошюр и книг. Огромное количество людей, использующих в своей повседневной деятельности Microsoft Word,

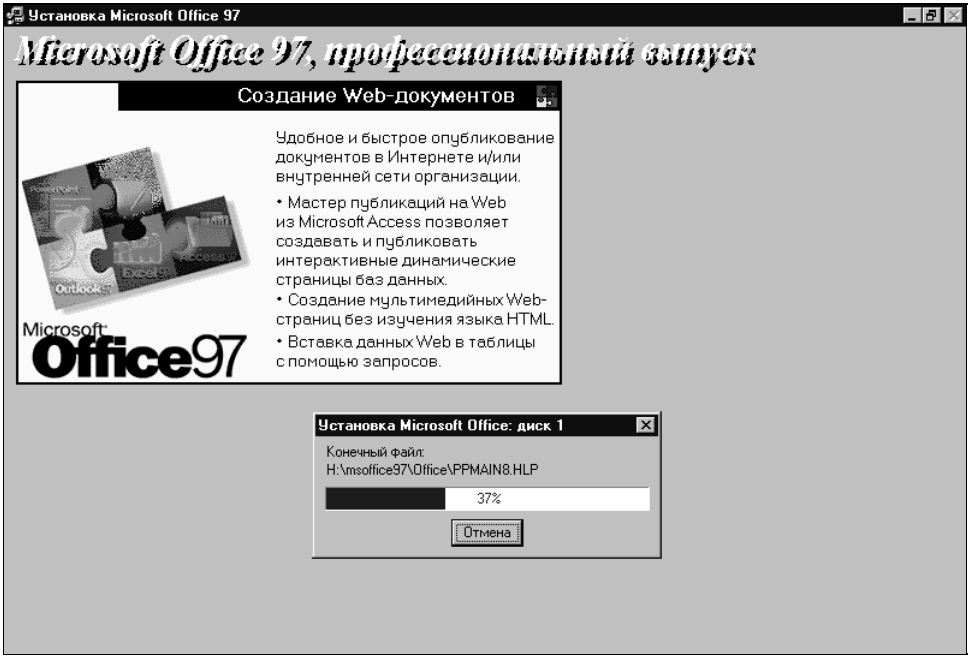


Рис. 8.1. Реклама возможностей Microsoft Office 97 при его инсталляции

становятся потенциальными разработчиками HTML-документов. В версии Microsoft Word 97 добавлены специальные возможности для работы с HTML-документами, которые переводят его в разряд мощных средств создания страничек для Интернета (рис. 8.1).

В данной главе будут рассмотрены основные возможности текстового процессора Microsoft Word 97, используемые при редактировании HTML-документов.

Создание Web-страниц

Создать новую Web-страницу с помощью Microsoft Word можно одним из двух способов: с помощью мастера или шаблона, либо преобразовать существующий документ Word в формат HTML, используемый для Web-страниц. Для использования мастера (или шаблона) достаточно воспользоваться вкладкой **Web-страницы** меню **Создание документа**. Далее следует воспользоваться указаниями мастера, отвечая на его вопросы и заполняя соответствующие поля и графы нужным текстом. Выполнение этих действий не вызовет трудностей у пользователей, владеющих основами работы в Microsoft Word. Другим способом является преобразование существующего документа Word в формат HTML. Это простейшая операция, однако здесь могут возникать некоторые проблемы, описываемые ниже в данной главе.

Создав HTML-документ одним из приведенных выше способов, можно в дальнейшем выполнять его редактирование. Для этого в Word предусмотрены специальные панели инструментов, команды меню и функции. Краткому описанию этих возможностей и посвящена данная глава.

Создание маркированных и нумерованных списков на Web-страницах

Так же, как и в документах Word, в Web-страницах можно создавать списки. Списки в HTML могут быть маркированные (Bulleted) и нумерованные (Numbered). В маркированных списках могут употребляться стандартные маркеры, реализация вида которых возлагается на браузер, а также графические изображения, загружаемые из отдельного файла. Для нумерованных списков могут быть использованы арабские или римские цифры, а также латинские буквы. Более подробная информация о правилах организации списков приводится в соответствующей главе.

Создание списков выполняется точно так же, как и в обычных документах Word. Выделите текст, который должен быть представлен в виде списка, и выберите пункт **Список** из меню **Формат**. Будет выдана панель **Список**, содержащая две вкладки — **Маркированный** (рис. 8.2) и **Нумерованный** (рис. 8.3).

Можно также воспользоваться кнопками панели инструментов **Нумерация** или **Маркеры**.

Рис. 8.2. Выбор вида маркированного списка

Рис. 8.3. Выбор вида нумерованного списка

При создании маркированного списка можно выбрать один из трех стандартных маркеров (последние три элемента первой строки, изображенные на рис. 8.2) или одно из предлагаемых графических изображений. Воспользовавшись кнопкой **Другой** можно в качестве маркера выбрать любой существующий графический файл. В зависимости от выбора типа маркеров список в HTML будет реализован по-разному. Если выбран один из трех стандартных маркеров, то список будет сформирован с помощью тэга `<UL>` и элементов списка `<LI>`. При этом будет генерироваться одна из следующих строк кода HTML:

`<UL>` — для первого стандартного маркера (маленький черный кружок)

`<UL TYPE="CIRCLE">` — для второго маркера (квадратик)

`<UL TYPE="SQUARE">` — для третьего маркера (закрашенный квадратик)

Приведем пример. Пусть был создан следующий текст:

```
Первый элемент списка
Второй элемент списка
Третий элемент списка
```

При разметке этого текста как списка и выборе второго стандартного маркера будет сгенерирован следующий фрагмент кода HTML:

```
<UL TYPE="CIRCLE"><FONT FACE="Times New Roman" SIZE=2>
<LI>Первый элемент списка </LI>
<LI>Второй элемент списка </LI>
```

```
<LI>Третий элемент списка</LI></UL>
</FONT>
```

Если в качестве маркера выбирается графическое изображение, то список, по существу, не формируется, то есть специальный тэг списка <UL> использоваться не будет. Здесь каждый элемент списка будет выделен в отдельный абзац тэгом <P> и предварен ссылкой на встроенное изображение. Само графическое изображение маркера будет сохранено в том же каталоге, что и HTML-файл. Для приведенного примера при выборе одного из графических маркеров будет сгенерирован следующий код:

```
<DIR>
<P><IMG SRC="Bullet5.gif" WIDTH=13 HEIGHT=13><FONT SIZE=5> </FONT>
<FONT FACE="Times New Roman" SIZE=5>Первый элемент списка </P> </FONT>
<P><IMG SRC="Bullet5.gif" WIDTH=13 HEIGHT=13><FONT SIZE=5> </FONT>
<FONT FACE="Times New Roman" SIZE=5>Второй элемент списка </P> </FONT>
<P><IMG SRC="Bullet5.gif" WIDTH=13 HEIGHT=13><FONT SIZE=5> </FONT>
<FONT FACE="Times New Roman" SIZE=5>Третий элемент списка</P>
</DIR>
</FONT>
```

Примечание

Различие в использовании стандартных и графических маркеров проявляется не только в реализации, но и возможности отмены списка. Если были выбраны стандартные маркеры, то список можно отменить так же, как это делается при работе с документом Word. Достаточно выделить список и выбрать режим **Нет** в панели **Список** или же нажать соответствующую кнопку на панели инструментов. Если же были выбраны графические маркеры, то отменить список не удается. Необходимо удалить все графические маркеры по отдельности, выделяя их и нажимая клавишу <Delete>.

Перейдем к рассмотрению нумерованных списков. Нумерация списков на Web-страницах почти не отличается от нумерации документов Word. Отличие состоит в том, что для Web-страниц невозможна автоматическая нумерация структурных списков и заголовков.

В зависимости от выбранного формата нумерации списка будут генерироваться следующие коды:

<OL> — для нумерации арабскими цифрами

<OL TYPE="I">> — большими римскими цифрами

<OL TYPE="i">> — маленькими римскими цифрами

<OL TYPE="A">> — прописными латинскими буквами

<OL TYPE="a">> — строчными латинскими буквами

Например:

```
<OL TYPE="A">
<FONT FACE="Times New Roman" SIZE=4>
<LI>Первый элемент списка</LI>
<LI>Второй элемент списка</LI>
<LI>Третий элемент списка</LI></OL>
</FONT>
```

Примечание

Выбрать формат нумерованного или маркированного списка можно только при использовании меню. Если вы воспользовались соответствующими кнопками панели инструментов, то будет выбран формат, использованный последним.

Хотя автоматическая нумерация многоуровневых списков не предусмотрена, списку можно придать многоуровневый вид, изменяя размер отступа и используя различные формы нумерации. Для этого нужно выполнить следующие действия:

1. Выделите текст, являющийся верхним уровнем списка.
2. Выберите нужный формат нумерации из вкладки **Нумерованный** панели **Список**.
3. Сдвиньте текст, принадлежащий следующему уровню списка, установив курсор в начале каждого абзаца или строки и нажав клавишу <Tab>. Хотя табуляция не применяется в HTML, в редакторе Word символы табуляции списка будут преобразованы в отступы.
4. Выберите формат нумерации для очередного уровня.
5. Для каждого уровня в списке повторите шаги 3—4.

Примечание

Несмотря на то, что приведенная рекомендация по созданию многоуровневых списков дана разработчиками Word 97, она не всегда работает корректно. Иногда для того, чтобы добиться желаемого результата, приходится доводить вручную получаемый HTML-код.

Вставка горизонтальной линии в Web-страницу

Горизонтальные линии часто используются на Web-страницах для разделения текста на части. Для того чтобы вставить в документ горизонтальную линию, нужно выполнить команду **Горизонтальная линия** в меню **Вставка**. Появится панель, показанная на рис. 8.4, содержащая список возможных вариантов горизонтальных линий. Если приведенного списка покажется недостаточно, то можно нажать кнопку **Другая** и выбрать другой вид линии.

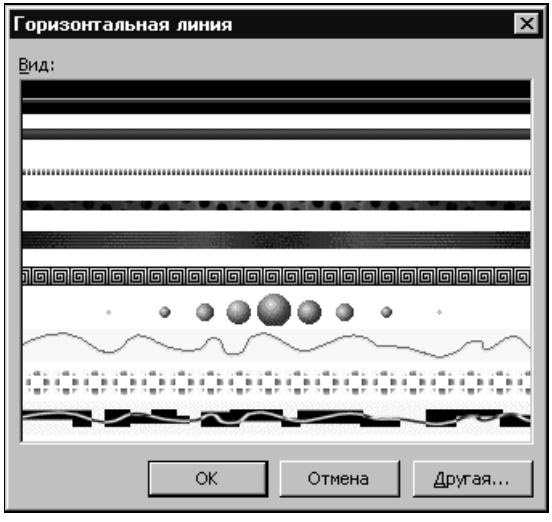


Рис. 8.4. Выбор вида горизонтальной линии для HTML-страницы

Все линии, кроме самой верхней в списке, представляют собой обычные файлы с графическими изображениями. Эти линии будут размещены в Web-документе как обычные встроенные изображения. Например, если выбрать самую нижнюю линию, то она будет вставлена в документ с помощью следующего кода:

```
<P ALIGN="CENTER"><IMG SRC="line9.gif" WIDTH=623 HEIGHT=11></P>
```

Первая же линия из списка реализуется с помощью стандартного тэга `<HR>`. Параметры горизонтальной линии могут быть изменены, если нажать правую кнопку мыши и в появившемся меню выбрать пункт **Формат автофигуры** (Format Autoshape). Можно изменить длину и толщину линии, ее расположение на странице в горизонтальном направлении. Например:

```
<HR ALIGN="LEFT" WIDTH="65%" SIZE=7>
```

Для того чтобы быстро вставить в текст страницы еще одну линию того же стиля, можно нажать кнопку **Горизонтальная линия** на панели инструментов.

Выбор фона создаваемого документа

Чтобы сделать Web-страницы более привлекательными, часто используют различные фоновые изображения, в том числе текстурную заливку.

Чтобы задать цвет фона Web-страницы или добавить фоновое изображение, следует воспользоваться командой **Фон** меню **Формат** (Format/Background). После выполнения этой команды появится диалоговая панель (рис. 8.5), с помощью которой можно выбрать требуемый цвет фона документа из

списка основных или дополнительных цветов или же задать изображение, которое будет служить фоновым. Для выбора фонового изображения необходимо нажать кнопку **Способы заливки** (Fill Effects), при этом появится панель **Заливка** (Fill Effects) с вкладкой **Текстура** (Texture), из которой можно подобрать нужное изображение (рис. 8.6).

Рис. 8.5. Задание требуемого цвета фона

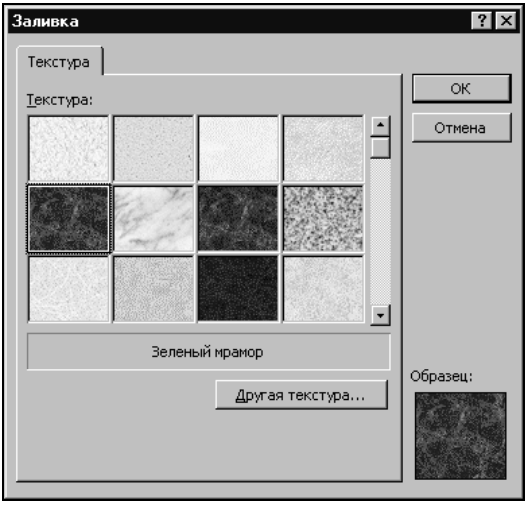


Рис. 8.6. Выбор текстуры

Кроме стандартных текстур, отображаемых на вкладке **Текстура**, в качестве фонового изображения можно использовать любой рисунок. Для этого нажмите кнопку **Другая текстура** и укажите файл, содержащий нужный рисунок.

Если будет определен цвет фона документа, то выбранное значение сохраняется в HTML-документе в качестве значения параметра `BGCOLOR` тэга `<BODY>`. При задании фонового изображения имя соответствующего файла графического изображения будет являться значением параметра `BACKGROUND`. Приведем примеры обоих вариантов:

```
<BODY LINK="#0000ff" VLINK="#800080" BGCOLOR="#ccffff">
<BODY LINK="#0000ff" VLINK="#800080" BACKGROUND="Image15.jpg">
```

Напомним, что фоновые изображения всегда заполняют весь экран браузера, при этом изображение будет повторяться столько раз, сколько необходимо для заполнения страницы.

При создании Web-страницы фоновое изображение сохраняется в виде отдельного файла, например `Image.gif`, который помещается в тот же каталог, что и сама Web-страница.

Изменение цвета и форматирование текста Web-страниц

При создании Web-страницы для текста и ссылок можно определить используемые по умолчанию цвета. Для этого следует воспользоваться командой **Цвет текста** меню **Формат** (Format/Text Colors). Появится панель, показанная на рис. 8.7, в которой можно задать цвет текста, цвет ссылок и цвет ссылок, соответствующий уже посещенным страницам. Зададим для примера в качестве цвета текста — синий, цвета ссылок — бирюзовый, а цвета просмотренных ссылок — зеленый. Заданные значения определят параметры тэга `<BODY>`:

```
<BODY TEXT="#0000ff" LINK="#00ffff" VLINK="#008000">
```

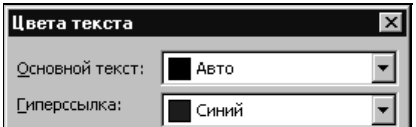


Рис. 8.7. Выбор цвета всего текста для HTML-страницы

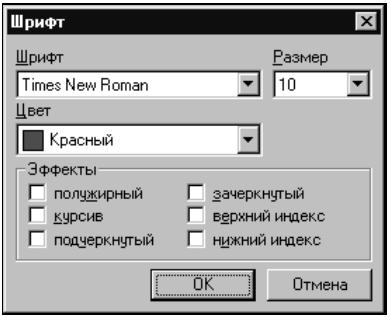


Рис. 8.8. Выбор цвета фрагмента текста для HTML-страницы

Эти параметры задают цвета по умолчанию для всего документа. Если же необходимо задать цвет для выбранного фрагмента документа, то следует использовать кнопку **Цвет шрифта** на панели инструментов или команду **Шрифт** меню **Формат** (рис. 8.8). Если для фрагмента текста будет выбран, например, красный цвет, то результат такого выбора в HTML-документе будет отражен так:

```
<FONT SIZE=2 COLOR="#ff0000">Этот текст будет отображаться
красным цветом</FONT>
```

Читатель Web-страницы может задать свои собственные устанавливаемые по умолчанию цвета в браузере. Чтобы текст и ссылки изображались в выбираемых по умолчанию цветах браузера, задайте значение **Авто** во всех трех списках панели **Цвета текста**.

При создании Web-страниц в редакторе Word можно использовать многие средства форматирования, предназначенные для форматирования документов Word. Например, можно присвоить тексту полужирное начертание с по-

мощью кнопки **Полужирный** или выбрать стиль **Заголовок 1** из списка стилей. Задание элементов форматирования текста определяет вставку соответствующих тэгов HTML в генерируемый код.

Панель **Шрифт**, вызываемая из меню **Формат**, служит также для задания типа шрифта, его размеров и форматирования выделенных фрагментов текста. Установка флажков, соответствующих каждому из шести доступных в данной панели эффектов, приводит к вставке в HTML-код следующих тэгов:

`<B>` — Полужирный

`<I>` — Курсив

`<U>` — Подчеркнутый

`<STRIKE>` — Зачеркнутый

`<SUP>` — Верхний индекс

`<SUB>` — Нижний индекс

Эти тэги вставляются в начало выделенного фрагмента текста, а в конце фрагмента записывается соответствующий закрывающий тэг. Заметим, что может использоваться любая совокупность перечисленных эффектов, за исключением двух последних, которые являются взаимоисключающими. Для полужирного начертания, курсива и подчеркнутого текста на панели инструментов имеются специальные кнопки **Ж** *К* Ч.

Для быстрого перехода к следующему доступному размеру шрифта используйте кнопки **Увеличить размер шрифта** A и **Уменьшить размер шрифта** A.

Для каждого абзаца в HTML генерируется тэг начала абзаца `<P>`, а также тэг конца абзаца `</P>`. Последний тэг при создании документов вручную обычно опускается. Большинство браузеров абзацы автоматически отделяют пустой строкой. Чтобы реализовать переход на новую строку без образования абзаца, нажмите клавиши <Shift>+<Enter>. В этом случае будет генерироваться тэг перехода на новую строку `<BR>`.

Примечание

В справочной системе Word 97 "для создания абзацев без интервалов" (именно так сформулировано) ошибочно предлагается воспользоваться клавишами <Ctrl>+<Enter>. Напомним, что совокупность клавиш <Ctrl>+<Enter> обычно используется при редактировании документов Word для принудительного перехода на следующую страницу. При преобразовании документов Word в HTML в этом случае будет лишь создан дополнительный абзац, а при редактировании Web-документов клавиши <Ctrl>+<Enter> вообще недопустимы.

Форматирование, не поддерживаемое языком HTML или некоторыми средствами просмотра Web, недоступно в разделе редактирования Web-страниц редактора Word. В число таких средств входят эффекты форматирования символов **Приподнятый**, **С тенью** и **Утопленный**, междустрочный интервал,

поля, межсимвольный интервал, кернинг, параметры обтекания текста и интервалы между абзацами. Не следует использовать символы табуляции, так как многие браузеры изображают их как пробелы; вместо них для сдвига первой строки текста вправо можно использовать отступ.

Для изменения отступа текста используйте кнопки **Увеличить отступ** и **Уменьшить отступ**. С точки зрения языка HTML, отступы реализуются путем обрамления текста парой тэгов `<DIR>` и `</DIR>`. Формально данный тэг определяет начало списка и подразумевает использование тэгов элементов списка `<LI>`. Однако здесь тэг `<LI>` не используется, а тэг `<DIR>` нужен только для организации отступов, которые невозможно задать в HTML естественным образом. Вложенные друг в друга тэги `<DIR>` подразумевают многоуровневые списки, причем каждый уровень сдвинут на один отступ по отношению к предыдущему. Например:

```
<DIR><P>Этот текст имеет один отступ</P>
<DIR>
<P>Этот текст имеет два отступа</P>
</DIR>
</DIR>
```

Примечание

Аналогичный подход для создания отступов применяют и другие HTML-редакторы. Например, Netscape Composer (так же, как и редактор пакета Netscape Navigator Gold) для отступов используется аналогичный прием, только в качестве тэга списка применяется `<UL>`.

Примечание

При увеличении отступа Word часто генерирует сразу два тэга `<DIR>`, создавая двойной отступ. Соответственно при уменьшении отступа удаляются два тэга `<DIR>`. Иногда генерируется и удаляется по одному тэгу `<DIR>`. Логику поведения редактора понять чрезвычайно затруднительно, если вообще возможно. По всей видимости мы имеем дело с очередной некорректностью работы программы.

Для изменения выравнивания текста используйте кнопки **По левому краю**, **По центру** и **По правому краю**. Заметим, однако, что текст на Web-страницах невозможно выровнять по ширине.

Примечание

Невозможность выравнивания текста на Web-страницах по ширине отмечается в справочной системе Microsoft Word. Действительно, в панели инструментов **Форматирование** при работе с HTML-документом нет кнопки выравнивания по ширине. Это связано с тем, что в спецификации HTML вплоть до версии 3.2 для параметров выравнивания тэга абзаца `<P>` не было выравнивания по ширине. Однако в настоящее время ведущие браузеры поддерживают эту возможность,

реализуемую следующим образом: `<P ALIGN=JUSTIFY>`. Более того, даже в Microsoft Word это уже заложено. Если работать с документом Word, в котором имеется текст, выровненный по ширине, а затем преобразовать его HTML-документ, то появятся абзацы, имеющие приведенный выше код.

Работа со стилями на Web-страницах

На Web-страницах можно использовать встроенные стили, соответствующие форматированию, принятому для языка HTML. Стиль назначается тексту Web-страниц так же, как документам Word, однако в работе стилей есть некоторые различия.

При создании Web-страницы в Word стили HTML добавляются в список стилей на панели форматирования и в диалоговом окне **Стиль** (меню **Формат**) (рис. 8.9). Один из стилей символов **Разметка HTML** специально предназначен для записи исходных кодов HTML вручную.

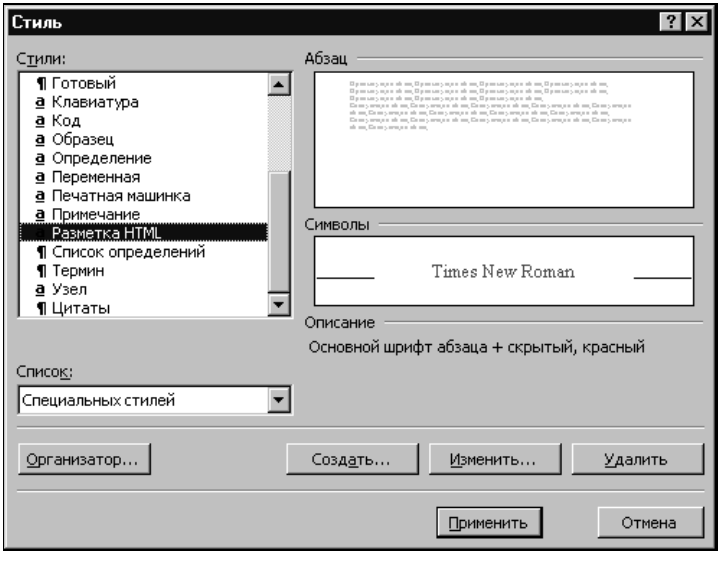


Рис. 8.9. Выбор стиля **Разметка HTML**

Специальные стили HTML, например **Адреса** и **H2**, непосредственно генерируют соответствующие команды языка HTML; поэтому изменения, внесенные в эти стили, не сохраняются. При внесении изменений во встроенный стиль Word, например **Заголовок 1**, связанное с этим стилем форматирование будет перенесено на соответствующую команду HTML, если такое форматирование поддерживается HTML.

Допускается создание и изменение собственных стилей. При сохранении страницы в формате HTML преобразуется только форматирование, поддер-

живаемое языком HTML, при этом все другие виды форматирования будут утрачены.

Непосредственное редактирование HTML-кода

Microsoft Word 97 рассчитан прежде всего на создание и изменение Web-страниц в визуальном режиме без явного применения тэгов HTML. Однако при необходимости на созданную страницу можно вручную вставить собственные коды HTML. Есть два пути решения этой задачи. Во-первых, можно редактировать HTML-код в режиме просмотра исходных кодов Web-страницы, а во-вторых, присвоить тексту специальный стиль **Разметка HTML**.

Перейти в режим просмотра исходных кодов Web-страницы можно, выполнив пункт **Источник HTML** в меню **Вид** (рис. 8.10). До того, как выполнить переход в этот режим, следует выполнить сохранение файла. Если этого не сделано, то будет выдано предупреждающее сообщение (рис. 8.11). В режиме просмотра исходных кодов Word можно использовать в качестве обычного текстового редактора, изменяя текст кода HTML. Обратный переход выполняется кнопкой **Закрыть источник HTML**. При этом переходе также выдается предупреждающее сообщение о необходимости сохранения сделанных изменений.

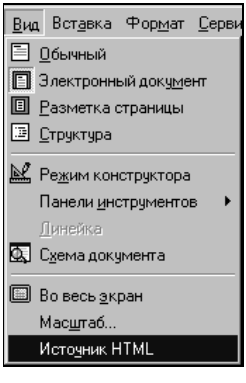


Рис. 8.10. Меню **Вид** Word 97 с командой **Источник HTML**

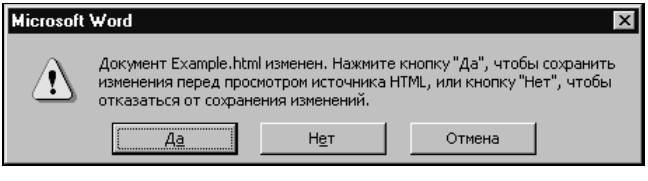


Рис. 8.11. Запрос подтверждения для сохранения изменений в HTML-файле

Другой путь состоит в записи необходимых HTML-кодов при работе в режиме визуального редактирования. При этом не требуется переходить в иной режим работы. Чтобы указать, что введенный текст является исходным кодом HTML, необходимо пометить его специальным стилем символов **Разметка HTML**. Этот стиль делает текст скрытым, поэтому, чтобы видеть его, нужно работать в режиме просмотра непечатаемых символов. Переключение данного режима осуществляется кнопкой **Непечатаемые символы** ¶.

Предварительный просмотр Web-страницы в процессе редактирования

Хотя Word 97 отображает редактируемый HTML-документ практически в таком же виде, как он в дальнейшем будет отображаться браузерами, всегда следует выполнять предварительный просмотр созданной страницы в предпочитаемом вами браузере. Это позволит вам убедиться в том, что и при размещении документа на Web-сервере он будет выглядеть соответствующим образом.

Для предварительного просмотра Web-страницы в процессе ее редактирования необходимо установить на компьютере браузер. Для просмотра следует нажать кнопку **Просмотр Web-страницы** на панели инструментов. Для возврата в Word следует выбрать значок Word на панели задач или закрыть браузер.

Таблицы на Web-страницах

Работа с таблицами на Web-страницах не отличается от работы с таблицами в документах Word. Для создания таблицы можно использовать два способа: вставить таблицу, воспользовавшись командой **Добавить таблицу** из меню **Таблица** (для этого случая необходимо задать размеры создаваемой таблицы) или путем преобразования существующего текста с разделителями (символ абзаца, символ табуляции, точка с запятой или иной символ) в таблицу (пункт **Преобразовать в таблицу** из меню **Таблица**). Для редактирования структуры таблицы используется команда **Нарисовать таблицу**. При этом возникнет панель инструментов **Таблицы и границы** (рис. 8.12), содержащая кнопки для изменения параметров таблицы.

Рис. 8.12. Изменение параметров таблицы с помощью панели инструментов **Таблицы и границы**

Например, если добавить таблицу размером 4×2, то Word создаст следующий код:

```
<TABLE CELLSPACING=0 BORDER=0 CELLPADDING=7 WIDTH=638>
<TR><TD WIDTH="50%" VALIGN="TOP"> </TD>
<TD WIDTH="50%" VALIGN="TOP"> </TD>
</TR>
<TR><TD WIDTH="50%" VALIGN="TOP"> </TD>
<TD WIDTH="50%" VALIGN="TOP"> </TD>
</TR>
<TR><TD WIDTH="50%" VALIGN="TOP"> </TD>
<TD WIDTH="50%" VALIGN="TOP"> </TD>
</TR>
<TR><TD WIDTH="50%" VALIGN="TOP"> </TD>
<TD WIDTH="50%" VALIGN="TOP"> </TD>
</TR>
</TABLE>
```

Так как на Web-страницах таблицы часто используются как скрытое средство форматирования (например, для обеспечения требуемого взаимного расположения текста и рисунков), вставляемые в текст таблицы по умолчанию не имеют границ. Для добавления границ к таблицам используйте команду **Границы** (меню **Таблица**).

После создания таблицы можно заняться изменением ее параметров. Для этого следует воспользоваться пунктом **Свойства таблицы** (рис. 8.13) или пунктом **Свойства ячейки** (рис. 8.14).

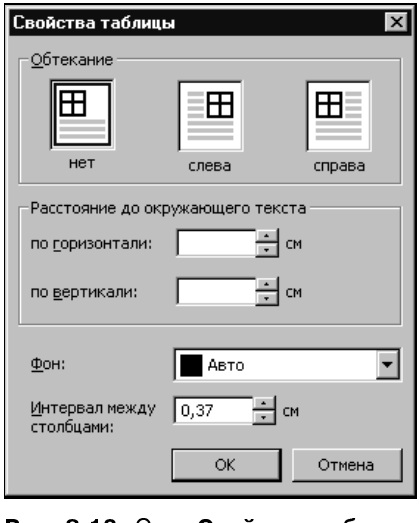


Рис. 8.13. Окно **Свойства таблицы** для изменения параметров всей таблицы

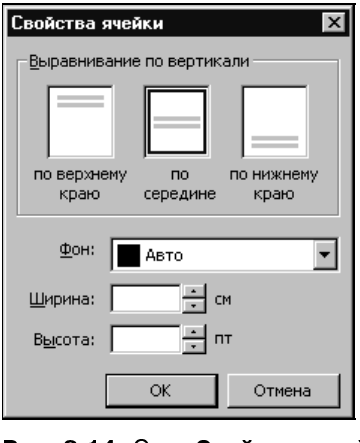


Рис. 8.14. Окно **Свойства ячейки** для изменения параметров отдельных ячеек таблицы

Читатели, имеющие опыт работы с таблицами в документах Word, сразу же смогут создавать таблицы и для HTML-документов. При отсутствии такого опыта можно научиться создавать таблицы по любой книге, посвященной работе с Microsoft Word, или воспользоваться справкой. Поэтому мы не будет здесь более подробно освещать вопросы создания таблиц, а отметим лишь некоторые особенности.

Поскольку возможности форматирования HTML-документов уступают аналогичным возможностям документов Word, то часто при преобразовании в HTML-документ некоторые элементы форматирования могут быть потеряны или изменены. Более того, даже при работе в режиме создания HTML-документа можно сконструировать на экране такую таблицу, которая не может быть в точно таком же виде отображена браузерами. Приведем пример. На рис. 8.15 показана таблица, созданная в Word в режиме редактирования HTML-документа.

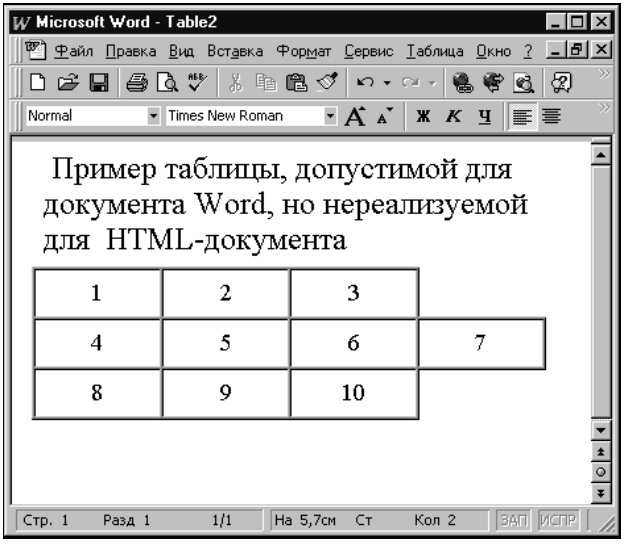


Рис. 8.15. Пример таблицы, существование которой в HTML-документе невозможно

Вторая строка таблицы содержит на одну ячейку больше, чем остальные строки. Язык разметки HTML не допускает ситуации, когда в разных строчках содержится разное число ячеек. Точнее, если при записи данных для отдельных строк указывается меньшее количество ячеек, чем в остальных строках, то браузеры автоматически добавляют недостающее число ячеек, считая, что они пустые. Естественно, при подсчете числа ячеек учитываются протяженные ячейки, которые занимают несколько смежных ячеек. (Напомним, что для протяженной ячейки требуется указать параметр `COLSPAN` с соответствующим значением.) Поэтому очевидно, что приведенный пример не может точно также отображаться браузерами. Редактор Word должен каким-то образом учитывать сложившуюся ситуацию. Чтобы убедиться в этом,

достаточно сохранить редактируемый документ в файле и вновь его загрузить в редактор. Заметим, что можно сделать даже проще, а именно, перейти в режим просмотра исходного кода документа, а затем вернуться обратно. При этих переходах предлагается сохранять изменения в файле, поэтому результат будет тот же. В результате этих действий вид таблицы изменится (рис. 8.16). Заметим, что редактор не создал дополнительной пустой ячейки, а растянул последнюю ячейку первой и третьей строк на две, создав протяженные ячейки.

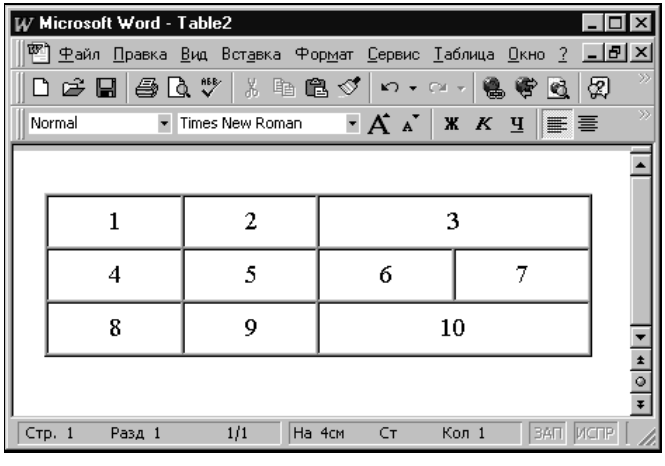


Рис. 8.16. Измененный вариант таблицы после ее сохранения в виде HTML-файла

Приведенный пример показывает, что вид HTML-страницы может отличаться не только при просмотре в различных браузерах, но даже может изменяться после сохранения редактируемой в Word страницы в файле. Такого быть, конечно, не должно, но поскольку факты свидетельствуют об обратном, то не преминем дать следующий совет.

Совет

После сохранения редактируемого в Word документа формата HTML не будет излишним вновь его загрузить и просмотреть. Эти действия, впрочем, не отменяют необходимость просмотра окончательного варианта документа в браузере.

Работа с рисунками на Web-страницах

При сохранении Web-страницы в формате HTML все изображения преобразуются в формат GIF или JPEG. При этом в формате JPEG сохраняются только те изображения, которые исходно были записаны в этом формате. Все остальные форматы графических файлов преобразуются в GIF. Чтобы вставить рисунок в Web-страницу, выберите команду **Рисунок** в меню **Вставка**, а затем команду **Из файла** или **Картинки**.

Если рисунок вставляется из файла, при сохранении он копируется в ту же папку, что и Web-страница, если не установлен флажок **Связь с файлом**. Если флажок **Связь с файлом** установлен, можно задать ссылку на рисунок в заданном каталоге, как на другой Web-узел.

Вставленный на Web-страницу рисунок по умолчанию выравнивается по левому полю. Чтобы задать взаимное расположение текста и рисунка, следует выделить рисунок и выбрать соответствующую команду в меню **Формат** или на панели инструментов **Рисование** (рис. 8.17).

Рис. 8.17. Вкладка **Положение** окна **Рисунок** для задания положения рисунка относительно текста

На рис. 8.18 показан пример использования изображения на создаваемой Web-странице. Для этого примера будет сгенерирован следующий фрагмент кода:

```
<IMG SRC="Image28.gif" WIDTH=213 HEIGHT=356 ALIGN="RIGHT" HSPACE=12>
<FONT FACE="Times New Roman" SIZE=5 COLOR="#000000"><P>Пример вставки
изображения в текст HTML-документа. Этот рисунок взят из подборки
Microsoft Clip Gallery и располагается на странице с выравниванием
по правому краю.</P></FONT>
```

Заметим, что Word автоматически нумерует рисунки, которые конвертируются из других форматов. В данном примере изображение, взятое из сборника рисунков Clip Gallery, было преобразовано в формат GIF и сохранено с именем `Image28.gif`.

Примечание

Изображения в файле сохраняются в их исходном размере. При изменении размеров изображения при компоновке Web-страницы для встроенного изображения изменяются лишь значения параметров `WIDTH` и `HEIGHT` тэга `<IMG>`. Таким образом, преобразование изображения к нужному размеру будет выполняться браузером при отображении страницы.

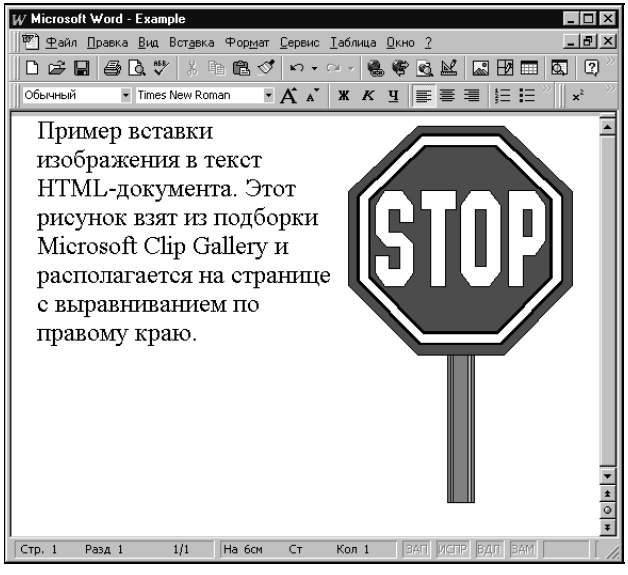


Рис. 8.18. Вставка изображений в HTML-документ

Создание ссылок в документе

Программа Word 97 позволяет включать в редактируемый документ ссылки внутри документа, а также ссылки на другие страницы и документы. Имеется несколько способов включения ссылок в состав документа. Наиболее простым способом является использование режима автоформатирования, когда текст, представляющий собой адрес ссылки, автоматически преобразуется в нужную ссылку. Для включения режима автоформатирования ссылок необходимо выполнить следующие действия: выбрать пункт **Автозамена** из меню **Сервис**. На вкладке **Автоформат при вводе** в разделе **Заменять при вводе** включить режим **адреса Интернета и сетевые пути гиперссылками**. Тогда при вводе текста, представляющего собой соответствующие адреса, они будут заменяться гиперссылками. Например, при вводе текста:

```
www.anywhere.ru
Sergeev@mail.ifmo.ru
```

будет сгенерирован следующий код:

```
<A HREF="http://www.anywhere.ru/">www.anywhere.ru</A>
<A HREF="mailto:Sergeev@mail.ifmo.ru">Sergeev@mail.ifmo.ru</A>
```

Другим способом включения в текст ссылки является использование команды **Гиперссылка** из меню **Вставка**. После вызова этой команды будет открыто диалоговое окно **Добавить гиперссылку**, в котором нужно указать требуемый адрес и другие параметры ссылок (рис. 8.19).

Рис. 8.19. Диалоговое окно **Добавить гиперссылку**

Создание форм на Web-страницах

Редактор Word 97 имеет набор инструментов для создания форм и задания свойств их элементов. Образцы форм, например формы для опроса, предоставляются мастером Web-страниц. С помощью мастера можно создать основные формы, а затем отредактировать их в соответствии с конкретными задачами. Если нужной формы нет в мастере, ее можно создать, вводя необходимые элементы управления. Для этого необходимо перейти в режим конструктора, воспользовавшись соответствующим режимом меню **Вид**. Затем нужно выбрать необходимые элементы из появившейся панели **Элементы управления** и разместить их в форме (рис. 8.20).

Рис. 8.20. Панель **Элементы управления**

Например, если разместить в форме строку ввода текста и две кнопки **Submit** и **Reset**, то будет создан следующий код:

```
<FORM>
<P>
<INPUT TYPE="TEXT" WIDTH="28">
</P>
<P>
<INPUT TYPE="SUBMIT">
</P>
<P>
```

```
<INPUT TYPE="RESET">
</P>
</FORM>
```

Заметим, что свойства элементов управления можно изменять, нажимая кнопку **Свойства** или выполняя двойной щелчок мышью на выбранном элементе.

Сохранение существующего документа Word в формате HTML

Сохранение документа в формате HTML представляет собой простейший способ создания Web-страниц. Для его реализации не требуется никаких знаний о формате HTML, а лишь необходимо уметь работать в среде Word. После открытия существующего или создания нового документа Word можно выполнить его сохранение в виде HTML-документа. Это выполняется в специальном пункте меню **Файл** с названием **Сохранить в формате HTML** (File/Save as HTML). Документ Word закрывается, а затем вновь открывается в формате HTML. На рис. 8.21 приведено содержимое меню **Файл**, используемое при работе в режиме документа Word.

Если перед преобразованием документ Word не был сохранен, то будет выдано предупреждающее сообщение (рис. 8.22) о том, что ряд элементов форматирования Word-документа будет потерян.

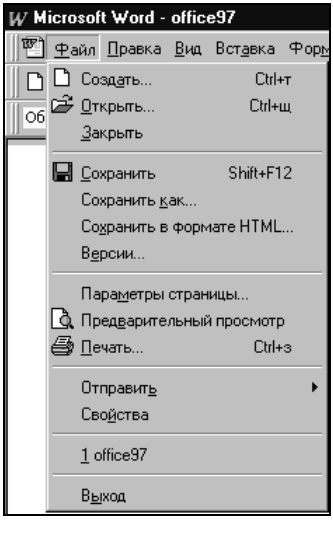


Рис. 8.21. Меню **Файл** с командой **Сохранить** в формате HTML

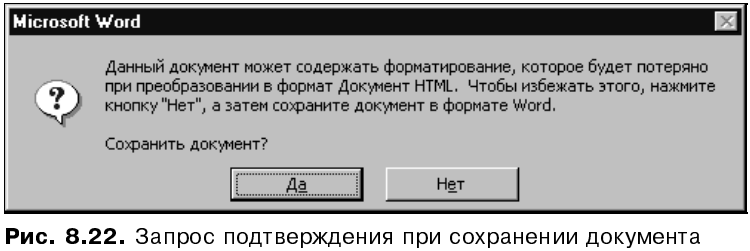


Рис. 8.22. Запрос подтверждения при сохранении документа Word в формате HTML

Поэтому лучше всего предварительно сохранять документ в формате Word, а лишь затем выполнять его преобразование. После открытия документа в формате HTML полученная Web-страница изображается практически в том виде, который она имела бы при просмотре ее с помощью Web-браузера. При необходимости в Web-страницу можно внести нужные изменения и вновь сохранить ее. Заметим, что при выполнении преобразования Word переходит в режим работы с документом формата HTML. Формально это отражается на содержимом пунктов меню и некотором изменении возможностей редактора. На рис. 8.23 приведено содержимое меню **Файл**, используемое при работе в режиме HTML-документа. Сравните пункты меню, приведенные на рис. 8.21 и 8.23.

Примечание

В справочной системе русифицированной версии Word 97 не используется термин "браузер". Вместо него применяется термин "средство просмотра Web".

Приведенных сведений вполне достаточно для выполнения преобразования простейших документов Word, не содержащих сложных элементов форматирования. Читателям, разрабатывающим документы Word с использованием элементарного набора приемов, можно на этом месте завершить ознакомление с данной главой и начать практиковаться в преобразовании своих документов. Выполнив преобразование и убедившись, что полученная Web-страница содержит все необходимые данные и выглядит практически так же, как и исходный документ Word, можно считать задачу создания HTML-документа выполненной.

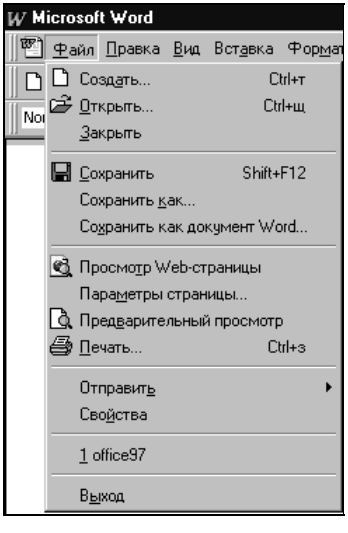


Рис. 8.23. Меню **Файл** при работе в режиме HTML-документа

Однако, отдельные элементы форматирования, не поддерживаемые языком HTML, могут быть потеряны или изменены. Приведем таблицу элементов документа Word, которые удаляются или изменяются в процессе преобразования (табл. 8.1). Эта таблица полностью заимствована из справочной системы Word 97 с незначительными редакционными изменениями (рис. 8.24).

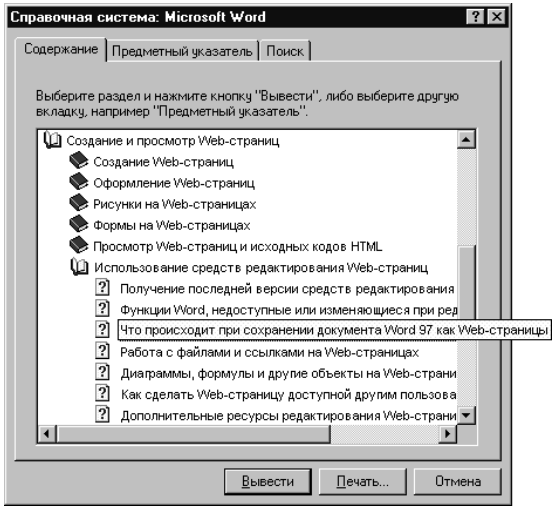


Рис. 8.24. Справочная система редактора Word по созданию и редактированию HTML-документов

Таблица 8.1. *Элементы документа Word, которые удаляются или изменяются при преобразовании в формат HTML*

Элемент документа Word	**Word —> HTML**	**Подробности преобразования**
Примечания	См. подробности	Примечание, введенное с помощью команды **Примечание** (меню **Вставка**), удаляется. Однако после сохранения документа в формате HTML примечания можно снова ввести и присвоить им стиль **Примечание**. Примечания не изображаются при просмотре Web-страницы с помощью средства просмотра Web
Размер шрифтов	См. подробности	Шрифт документа заменяется шрифтом HTML соответствующего размера. Размеры шрифтов HTML изменяются от 1 до 7. Эти цифры не являются собственно размером шрифта, а служат указанием на размер шрифта для средств просмотра Web. В Word изображаются шрифты размером от 9 до 36 пунктов
Текстовые эффекты: приподнятый, с тенью, утопленный, все прописные, малые прописные, двойное зачеркивание и контур (меню **Формат**, команда **Шрифт**, вкладка **Шрифт**)	Нет	Текстовые эффекты не сохраняются, но текст остается

Таблица 8.1 *(продолжение)*

Элемент документа Word	Word —> HTML	Подробности преобразования
Начертания: полужирное, курсив, подчеркивание и зачеркивание	Да	Некоторые особые эффекты подчеркивания, например подчеркивание пунктиром, преобразуются в подчеркивание сплошной чертой, а другие — нет
Анимация текста (меню **Формат**, команда **Шрифт**, вкладка **Анимация**)	См. подробности	Анимация не сохраняется, но текст остается. Вместо эффектов анимации с помощью среды редактирования Web-страниц на страницу следует поместить бегущую строку
Графика	См. подробности	Изображения, например рисунки и картинки, преобразуются в формат GIF, если рисунки не сохранены уже в формате JPEG. Графические объекты, например, надписи и фигуры, не преобразуются. Линии преобразуются в горизонтальные линии
Табуляция	Да	Символы табуляции преобразуются в символы табуляции HTML, представляемые в исходных кодах как &#9. В некоторых браузерах символы табуляции заменяются пробелами, поэтому вместо них лучше использовать отступы или таблицы
Поля	См. подробности	Значения полей преобразуются в текст; коды полей удаляются. Например, если в документ вставлено поле DATE, текст даты преобразуется, но сама дата не будет в дальнейшем обновляться
Оглавления и указатели	См. подробности	Информация преобразуется, но указатели и оглавления после преобразования не будут обновляться автоматически, так как они базируются на кодах полей. В оглавлении вместо номеров страниц изображаются звездочки; эти звездочки являются гиперссылками, позволяющими пользователю перемещаться по Web-странице. Звездочки можно заменить текстом, который будет указывать на гиперссылки
Буквицы	Нет	Буквицы удаляются. В среде создания и редактирования Web-страницы размер одной буквы можно увеличить, выделив ее и нажав кнопку **Увеличить размер**. Если для буквы имеется графическое изображение, его можно вставить перед текстом

Таблица 8.1 *(продолжение)*

Элемент документа Word	Word —> HTML	Подробности преобразования
Графические объекты: автофигуры, фигурный текст, надписи и тени	Нет	Графические объекты не сохраняются. В среде редактирования и создания Web-страницы можно использовать графические средства, вставляя объекты **Рисунок Word**. Объекты преобразуются в формат GIF
Формулы, диаграммы и другие объекты OLE	См. подробности	Объекты преобразуются в изображения формата GIF. Их внешний вид сохраняется, но они не могут быть изменены
Таблицы	Да	Таблицы преобразуются, однако параметры, не поддерживаемые средой редактирования Web-страниц, не сохраняются. Цветные границы и границы переменной ширины не сохраняются
Ширина таблиц	См. подробности	По умолчанию после преобразования таблицы имеют фиксированную ширину. Чтобы после преобразования таблица получила относительную ширину (при этом размер таблицы составляет определенную долю от размера окна средства просмотра), установите параметр PercentageTableWidth=1 в следующем поле системного реестра Windows 95:HKEY_LOCAL_MACHINE\Software \Microsoft\Shared Tools\Text Converters \Export\HTML\Options
Выделение цветом	Нет	Выделение цветом не сохраняется
Записанные исправления	Нет	Внесенные изменения сохраняются, однако пометки исправлений удаляются
Нумерация страниц	Нет	Так как документ HTML считается одной Web-страницей, независимо от его длины, нумерация страниц не сохраняется
Поля страниц	Нет	Чтобы сохранить разметку страницы, можно использовать таблицу
Границы вокруг абзацев и слов	Нет	Границы таблицы сохраняются. Для разделения частей и выделения важных мест Web-страницы можно использовать горизонтальные линии

Таблица 8.1 *(окончание)*

Элемент документа Word	Word —> HTML	Подробности преобразования
Границы страниц	Нет	В HTML нет соответствия границам страниц. Страницу можно сделать более привлекательной с помощью фона. Фон выбирается командой **Фон** в меню **Формат**. Таблицу можно снабдить границами. Для разделения частей и выделения важных мест Web-страницы можно использовать горизонтальные линии
Верхние и нижние колонтитулы	Нет	В HTML отсутствуют аналоги колонтитулов
Обычные и концевые сноски	Нет	
Колонки	Нет	Для создания колонок используйте таблицы
Стили	См. подробности	Определенные пользователем стили преобразуются в прямое форматирование, если оно поддерживается HTML. Например, при преобразовании стиля, содержащего полужирное начертание и затенение текста, полужирное начертание преобразуется в прямое форматирование, а затенение не сохраняется

Основываясь на сведениях, приведенных в таблице, можно понять причины изменения вида документа при преобразовании и принять соответствующие меры. Для большинства случаев вполне достаточно воспользоваться рекомендациями, приведенными в таблице, однако некоторые случаи заслуживают особого разговора. Прокомментируем ниже отдельные проблемы, возникающие при преобразовании документов в Web-страницу, а заодно приведем пример выполнения преобразования.

Проблемы преобразования полей

Рассмотрим на примере некоторые проблемы, возникающие при преобразовании документа Word в HTML-документ. Пусть имеется документ, показанный на рис. 8.25.

В нем имеется таблица, содержащая поля с формулами, а также некоторая другая информация, содержащая обновляемые поля. В первом столбце располагаются названия продуктов, второй и третий столбцы содержат соответ-

ствующие значения, которые должны быть введены пользователем. Каждая ячейка третьего столбца содержит поле с формулой, определяющей перемножение значений ячеек второго и третьего столбца. В конце таблицы задана формула, подводящая итоговое значение. Такая таблица может с успехом служить шаблоном для оперативных подсчетов затрат. Достаточно заполнить соответствующие ячейки в первых трех колонках, обновить значения полей, содержащих формулы, тогда последняя колонка и итоговая графа примут нужные значения. Конечно, задачи обработки электронных таблиц логичнее решать в Microsoft Excel, который специально предназначен для этого, однако некоторые элементарные возможности вполне допустимо реализовывать и в Word. Здесь специально выбран пример таблицы с формулами. Отметим также еще одну особенность таблицы — наличие протяженной ячейки. В последней строке объединены вместе три ячейки, в которой записано слово "Итого:". Такого же визуального эффекта можно было бы добиться и без объединения ячеек. Например, можно было для этих трех ячеек оставить только прорисовку общей рамки, а линии между этими ячейками отменить. Такое решение допустимо для документа Word, однако неосуществимо в формате HTML. Обратите внимание на то, как протяженная ячейка будет реализована в кодах HTML, приводимых ниже.

Примечание

Microsoft Word 97 предоставляет также возможность объединения нескольких ячеек таблицы по вертикали. Этой возможности не было в предыдущих версиях Word.

Тот же самый документ на рис. 8.26 показан в режиме отображения кодов полей. В таблице используется формула умножения (PRODUCT), а также формула для суммирования всех элементов данного столбца (SUM).

ZATRAT.DOC

Затраты на покупку продовольственных товаров

Продукт	Количество, кг	Цена за 1 кг	Стоимость, руб.
Хлеб	1	4,5	4,5
Сахар	2	7,3	14,6
Картофель	5	5,0	25
Огурцы	0,5	11	5,5
Помидоры	1	8	8
Итого:			57,6

Дата составления: вторник, 21 Сентября 1999 г.
Составил: Сергеев А.О.

Рис. 8.25. Документ Word, содержащий поля с формулами

ZATRAT.DOC

Затраты на покупку продовольственных товаров

Продукт	Количество, кг	Цена за 1 кг	Стоимость, руб.
Хлеб	1	4,5	{ =PRODUCT(B2;C2) }
Сахар	2	7,3	{ =PRODUCT(B3;C3) }
Картофель	5	5,0	{ =PRODUCT(B4;C4) }
Огурцы	0,5	11	{ =PRODUCT(B5;C5) }
Помидоры	1	8	{ =PRODUCT(B6;C6) }
Итого:			{ =SUM(ABOVE) }

Дата составления: { TIME \@ "dddd, d MMMM yyyy 'г.'" }
Составил: { USERNAME * MERGEFORMAT }

Рис. 8.26. Документ Word, показанный в режиме отображения кодов полей

Примечание

Напомним, что коды полей можно просмотреть при работе с документом в формате Word, используя горячую клавишу переключения режима просмотра полей (View Fields, <Alt>+<F9>). Также допустимо переключать отображение отдельного поля (Toggle Field, <Shift>+<F9>).

Ниже таблицы располагаются поля с текущей датой и поле со сведениями о пользователе. При обновлении этих полей берется дата, установленная на компьютере и текст, заданный в графе имени пользователя.

Примечание

Напомним, что обновление выделенного поля выполняется клавишей <F9> (Update Field). Для одновременного обновления всех полей можно выполнить пункт **Выделить все** меню **Правка**, а затем нажать клавишу <F9>. Автоматическое обновление полей при изменении данных не производится (в отличие от Excel).

Выполним сохранение данного документа в формате HTML. Как указано в табл. 8.1, при преобразовании документа в формат HTML коды полей утрачиваются и заменяются соответствующим текстом. В действительности не все коды полей при преобразовании будут заменяться соответствующим текстом. Для данного примера поле, содержащее дату, будет заменено текстом (как это и указано в качестве примера в таблице), однако все остальные коды полей просто исчезнут и не будут заменены ничем. Преобразованный документ в формате HTML будет выглядеть следующим образом (рис. 8.27).

Трудно анализировать причины такого преобразования. Бессмысленно также проводить анализ, какие поля будут преобразовываться, а какие нет. Возможно это связано с особенностями русифицированной версии Word 97,

а может быть просто с наличием ошибок, которые будут исправлены в следующих версиях. Задача, поставленная в данном примере, довольно типична, поэтому предложим путь решения, который позволит обойти описанные проблемы.

Затраты на покупку продовольственных товаров

Затраты на покупку продовольственных товаров

Продукт	Количество, кг	Цена за 1 кг	Стоимость, руб.
Хлеб	1	4,5	
Сахар	2	7,3	
Картофель	5	5,0	
Огурцы	0,5	11	
Помидоры	1	8	
Итого:			

Дата составления: вторник, 21 Сентября 1999 г.

Составил:

Рис. 8.27. HTML-документ, полученный в результате преобразования из документа Word

Для достижения поставленной цели можно рекомендовать до проведения преобразования в формат HTML осуществить процедуру разрыва связи с полями (Unlink Field). Иначе эту процедуру называют отключением полей. После разрыва связи с полем текущее значение поля становится обычным текстом. Код поля исчезает, поэтому, если в дальнейшем потребуется обновить сведения, придется вставлять это поле снова. Следует порекомендовать сохранять документ с кодами полей, а процедуру разрыва связи с полями осуществлять с копией документа, которая может быть удалена после преобразования в формат HTML. Это обеспечит возможность возврата к документу Word с обновляемыми полями. Для разрыва связи с полем необходимо выделить его и нажать клавиши <Ctrl>+<Shift>+<F9>. Для разрыва связей со всеми полями документа можно выделить весь документ и нажать те же клавиши. Заметим, что при разрыве связи с полем не происходит его обновления. Например, если вы изменили какие-то значения во втором или третьем столбце примера и разорвали связь с полями без их обновления, то в итоговой колонке сохранятся прежние значения. После выполнения всех этих действий в документе не останется обновляемых полей, а будет присутствовать лишь обычный текст, преобразование которого в HTML будет выполнено без проблем.

Сведем все указанные рекомендации в единый порядок действий, выполняя которые можно избежать потери значений полей при преобразовании в HTML-формат:

1. Откройте документ Word, подлежащий преобразованию в HTML-формат.
2. Если необходимо, то обновите поля документа. Для этого выделите весь документ или его фрагмент и нажмите клавишу <F9>.
3. Выполните разрыв связи с полями. Для этого выделите весь документ или его фрагмент и нажмите совокупность клавиш <Ctrl>+<Shift>+<F9>. Коды полей будут потеряны, поэтому предварительно сохраните копию документа.
4. Выполните сохранение файла в HTML-формате.

Примечание

Обратим внимание на некоторые возможности Word, незнание которых может повергнуть в шок даже довольно опытных пользователей. Любое из полей документа может быть подвергнуто блокировке, что означает запрет на его обновление. Блокировка поля (Lock Field) выполняется клавишами <Ctrl>+<F11>. Снятие блокировки (Unlock Field) — клавишами <Ctrl>+<Shift>+<F11>. Если какое-либо поле заблокировано, то запрос на его обновление будет проигнорирован и при этом не будет выдано никаких текстовых сообщений. Лишь те пользователи, компьютер которых оснащен звуковой картой, при включенном звуке услышат сигнал, означающий невозможность выполнения требуемой операции. Визуально различить заблокированное поле от незаблокированного невозможно, так как они отображаются абсолютно одинаково. Определить наличие заблокированного поля можно лишь косвенно по звуковому сигналу или увидев, что пункт **Обновить поле** в контекстном меню, выдаваемом по правой кнопке мыши, недоступен.

Приведем HTML-код, полученный в результате преобразования документа из примера:

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-1251">
<META NAME="Generator" CONTENT="Microsoft Word 97">
<TITLE>Затраты на покупку продовольственных товаров</TITLE>
<META NAME="Template" CONTENT="C:\PROGRAM FILES\MICROSOFT
OFFICE\OFFICE\html.dot">
</HEAD>
<BODY LINK="#0000ff" VLINK="#800080">

<B><FONT FACE="Arial" SIZE=5><P>Затраты на покупку продовольственных
товаров</P></B></FONT>
<TABLE BORDER CELLSPACING=1 CELLPADDING=7 WIDTH=557>
<TR><TD WIDTH="29%" VALIGN="TOP">
<P><FONT FACE="Arial" SIZE=5>Продукт</FONT></TD>
```

```
<TD WIDTH="27%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=5><P ALIGN="CENTER">Количество, кг</FONT></TD>
<TD WIDTH="19%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=5><P ALIGN="CENTER">Цена за</P>
<P ALIGN="CENTER">1 кг</FONT></TD>
<TD WIDTH="25%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=5><P ALIGN="CENTER">Стоимость, руб.</FONT></TD>
</TR>
<TR><TD WIDTH="29%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=4><P>Хлеб</FONT></TD>
<TD WIDTH="27%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=4><P ALIGN="CENTER">1</FONT></TD>
<TD WIDTH="19%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=4><P ALIGN="CENTER">4,5</FONT></TD>
<TD WIDTH="25%" VALIGN="TOP"> </TD>
</TR>
<TR><TD WIDTH="29%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=4><P>Сахар</FONT></TD>
<TD WIDTH="27%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=4><P ALIGN="CENTER">2</FONT></TD>
<TD WIDTH="19%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=4><P ALIGN="CENTER">7,3</FONT></TD>
<TD WIDTH="25%" VALIGN="TOP"> </TD>
</TR>
<TR><TD WIDTH="29%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=4><P>Картофель</FONT></TD>
<TD WIDTH="27%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=4><P ALIGN="CENTER">5</FONT></TD>
<TD WIDTH="19%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=4><P ALIGN="CENTER">5,0</FONT></TD>
<TD WIDTH="25%" VALIGN="TOP"> </TD>
</TR>
<TR><TD WIDTH="29%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=4><P>Огурцы</FONT></TD>
<TD WIDTH="27%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=4><P ALIGN="CENTER">0,5</FONT></TD>
<TD WIDTH="19%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=4><P ALIGN="CENTER">11</FONT></TD>
<TD WIDTH="25%" VALIGN="TOP"> </TD>
</TR>
<TR><TD WIDTH="29%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=4><P>Помидоры</FONT></TD>
<TD WIDTH="27%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=4><P ALIGN="CENTER">1</FONT></TD>
<TD WIDTH="19%" VALIGN="TOP">
<FONT FACE="Arial" SIZE=4><P ALIGN="CENTER">8</FONT></TD>
```

```
<TD WIDTH="25%" VALIGN="TOP"> </TD>
</TR>
<TR><TD WIDTH="75%" VALIGN="TOP" COLSPAN=3>
<FONT FACE="Arial" SIZE=5><P ALIGN="CENTER">Итого:</FONT></TD>
<TD WIDTH="25%" VALIGN="TOP"> </TD>
</TR>
</TABLE>

<FONT FACE="Arial" SIZE=4><P>Дата составления: вторник,
21 Сентября 1999 г.</P>
<P>Составил: </P></FONT></BODY>
</HTML>
```

Использование примечаний в документе

Напомним, что примечания часто используются при работе нескольких пользователей с одним документом, каждый из которых вносит свои ремарки. Добавление примечаний в документ выполняется в пункте **Примечание** меню **Вставка** (Insert/Annotation). Как правило, в окончательный документ примечания не вносятся, а используются в качестве рабочих записей отдельных разработчиков или рецензентов документа. Примечания похожи на сноски (Footnote) тем, что при вставке в документ примечания в то место, к которому оно относится, ставится специальный символ, а сам текст примечания, как и текст сноски, вводится в отдельном окне. Примечания нумеруются последовательно, но в отличие от сносок, в символ примечания добавляются инициалы автора данного примечания. Word позволяет также вставлять звуковые примечания, если компьютер оснащен звуковой платой и микрофоном.

Все примечания и сноски, вставленные в документе Word, при преобразовании в формат HTML теряются. Это указано в приведенной выше таблице, где также дается рекомендация по вставке примечаний в HTML-документ. Для этого достаточно выделенный текст пометить стилем **Примечание** (Comment). Этот совет компании Microsoft нуждается в комментариях.

Использование этого стиля приводит к тому, что весь текст, помеченный данным стилем, обрамляется парой тэгов `<COMMENT>` и `</COMMENT>`. Этот тэг служит для отметки примечаний (часто их называют комментариями), которые игнорируются браузерами и часто служат лишь для заметок автора HTML-документа. Такой подход к реализации комментариев в HTML имеет существенный недостаток. Дело в том, что тэг `<COMMENT>` не распознается некоторыми браузерами. Он не входил в спецификацию HTML 3.2 и был введен лишь в спецификации HTML 4.0. До сих пор этот тэг не распознается браузером Netscape (вплоть до версии 4.7). Впервые этот тэг был использован в браузере Mosaic и вскоре был включен в Microsoft Internet Explorer.

В настоящее время стало общепринятым записывать комментарии, обрамляя их совокупностью символов <!-- и -->. Этот вариант записи комментариев распознается всеми ведущими браузерами и давно входит в спецификацию HTML. Ясно, что компания Microsoft мало заботится о совместимости с другими браузерами, однако на сегодняшний день забывать о значительной роли браузера Netscape не следует. Браузер Netscape проигнорирует тэг <COMMENT> при отображении документа на экране и весь текст комментария появится на экране, что вряд ли соответствует замыслам автора документа. Исходя из приведенных соображений рискнем дать следующий совет, входящий в противоречие с рекомендациями разработчиков Word 97.

Совет

Не используйте стиль **Примечание** (Comment) при создании HTML-документа в среде Word 97. Если вы все-таки воспользовались этим стилем, то в полученном HTML-коде выполните замену сгенерированных тэгов <COMMENT> и </COMMENT> соответственно на символы <!-- и -->.

Версии и реализации Microsoft Word

Пакет Microsoft Office так же, как и подавляющее большинство пакетов программного обеспечения, в ходе своего развития претерпевал изменения, что отражалось в выпуске новых версий программ и исправлении недочетов. В результате оказывается, что пользователи пакета работают с его различными версиями и при этом не всегда представляют себе особенности и возможности. Опыт общения со многими пользователями, зачастую далекими от компьютерных профессионалов, свидетельствует о достаточном консерватизме в использовании программного обеспечения. Это связано как с недостатком времени и возможностей пользователей постоянно заботиться об обновлении программных средств, так и с иногда принципиальной невозможностью установки более мощных программ без соответствующей модернизации технической части компьютерного парка. Данные рассуждения особенно касаются такого пакета, как Microsoft Word, с которым работает очень широкий круг людей, не имеющих специального компьютерного образования. Исходя из сегодняшнего опыта и экстраполируя ситуацию на будущее, можно утверждать, что в ближайшее время будут использоваться самые различные версии Microsoft Word. В частности, до сих пор встречается MS Windows 3.1, работающая на стареньких по сегодняшним меркам компьютерах. Наступающая эра Windows 2000 и, соответственно, Office 2000 далеко не сразу охватит весь парк компьютеров.

Приведем здесь лишь краткую информацию об используемых версиях Microsoft Word и некоторых их особенностях.

Пакет Microsoft Word имеет давнюю историю. Ранние версии пакета работали еще под MS-DOS, что сегодня уже практически не встречается. Версия

пакета для Windows 3.1 называлась Microsoft Word 6.0. Для операционной системы Windows 95 был выпущен Microsoft Word 7.0, который чаще стали называть коротко — Microsoft Word 95. В своей основе эта программа представляет собой Microsoft Word 6.0, перенесенный в среду Windows 95 с добавлением некоторых новых возможностей. Заметим, что эти версии программы имеют одинаковую структуру файлов, хранящих подготовленные документы. Здесь проблема совместимости и переноса файлов еще не возникала.

Следующей версией стала версия Microsoft Word 97, которая появилась в начале 1997 года.

Примечание

Разработчики, видимо, хотели сохранить преемственность в нумерации версий (6.0 → 7.0 → 8.0), однако прижилось лишь название Word 97, а номер версии 8.0 можно увидеть, лишь заглянув внутрь кодов любого файла, сохраненного редактором Microsoft Word 97.

Версия обогатилась новыми возможностями, одной из которых является способность сохранять, просматривать и редактировать документы в формате HTML. Предыдущие версии не обладали такой возможностью, однако для них были разработаны специальные программы, при установке которых подобная возможность появлялась. Такие программы для Microsoft Office получили название Internet Assistant. В частности, для Microsoft Word 6.0/95 имеется программа Wordia.exe (размер этого установочного файла — 1091 Кб). Пользователям, не желающим переходить на более поздние версии Microsoft Word, можно рекомендовать установку этой программы.

Версия Microsoft Word 97 не нуждается в программах типа Internet Assistant, так как соответствующие возможности уже заложены в этот пакет. Заметим лишь, что для использования возможностей редактирования HTML-документов в Microsoft Word 97 необходимо указать это на этапе инсталляции пакета, поскольку типичный вариант инсталляции, который обычно выбирают многие пользователи по умолчанию, не предполагает установку данного компонента (рис. 8.28).

Microsoft Word 97 имеет другую структуру файла документа, что вызывает проблемы переносимости. Microsoft Word 97 может читать любые файлы формата Word 6.0/95, может записывать по требованию пользователя в формате Word 6.0/95 (с возможной потерей некоторых нюансов форматирования документа) для обратной совместимости. Однако документы, сохраненные в формате Word 97, впрямую не могут читаться программами Microsoft Word 6.0/95. Для обеспечения совместимости (пусть неполной) существует специальный конвертер, устанавливаемый в среду Microsoft Word 6.0/95, который предоставляет возможность чтения файлов Microsoft Word 97. Этот конвертер обычно поставляется в виде файла с именем Wrd97cnv.exe размером 952 Кб.

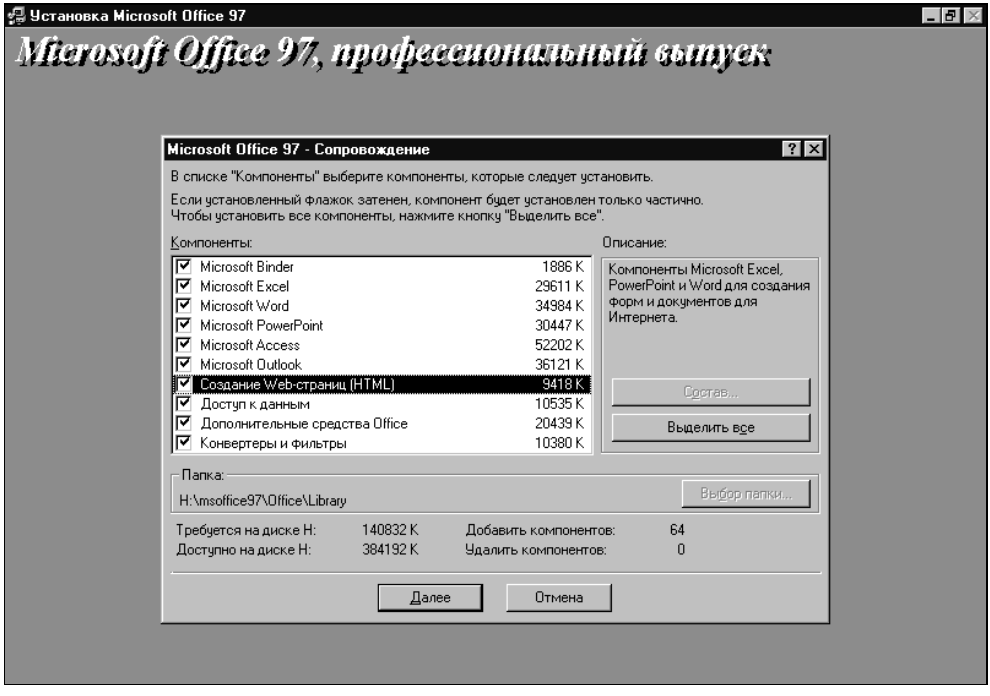


Рис. 8.28. Включение компоненты **Создание WEB-страниц (HTML)** при установке пакета Microsoft Office

Как и у многих программ, с течением времени обнаруживаются ошибки и недочеты, исправляемые специальными пакетами обновления (patch). На текущий момент для Microsoft Word 97 существует уже два таких пакета, получивших названия SR-1 (Service Release-1) и SR-2. Пакет SR-1 существует в виде файла SR1off97.exe размером 8022 Кб. Пакет SR-2 — в виде файла SR2aof97.exe размером 22448 Кб. Установка этих пакетов может быть рекомендована пользователям исходной версии Microsoft Word 97 (рис. 8.29).

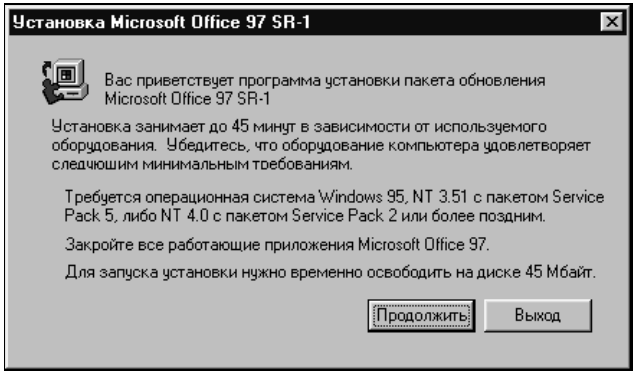


Рис. 8.29. Окно установки пакета обновления SR-1

Заметим, что установка SR-1 занимает несколько десятков минут (в зависимости от параметров компьютера и наличия установленных компонент Microsoft Office 97). Установка же SR-2 выполняется в течение десятков секунд. Уточнить версию Microsoft Word 97, с которой работает пользователь всегда можно, воспользовавшись пунктом меню **О программе** (About) из меню **Справка** (рис. 8.30).

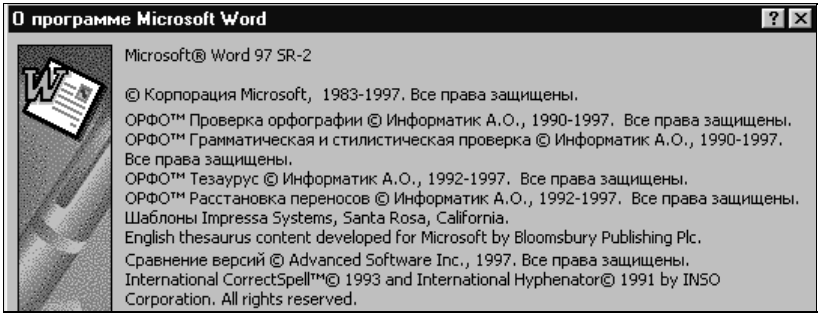


Рис. 8.30. Окно, открывающееся по команде **Справка/О программе**

Кроме того, появилась специальная небольшая программа, единственное назначение которой — проконтролировать номер версии пакета Microsoft Office. Имя программы SR2chk.exe (123 Кб). Ее работа показана на рис. 8.31.

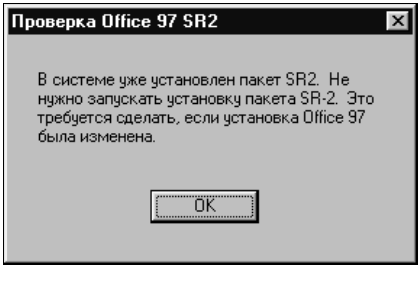


Рис. 8.31. Информационное окно, генерируемое программой проверки версий пакета Microsoft Office

Появилась и уже работает у ряда пользователей и следующая версия — Microsoft Office 2000. Ее нюансы и особенности лишь только начинают обнаруживаться и их обсуждение — дело ближайшего будущего.

ЧАСТЬ II

9. Выполняемые сценарии
10. Динамический HTML
11. Встраиваемые компоненты

Интерактивные Web-документы

Стандартный язык разметки HTML позволяет легко и быстро создавать Web-страницы, передаваемые по сети Интернет. Это достаточно удобный инструмент работы в сети, однако загружаемые в окно браузера страницы являются статичными. Пользователь не может изменять их содержимое, не может взаимодействовать с ними. Мы живем в динамичном, меняющемся мире, и, естественно, хотели бы видеть то же самое и в рукотворном мире — мире Интернета.

Для придания динамичности HTML-страницам был предложен и реализован ряд технологий, "оживляющих" и создающих "реагирующие" на действия пользователя HTML-документы. Одной из первых технологий в этом ряду стоит технология, основанная на CGI-сценарии — программе, инициализируемой на сервере при передаче на него информации из полей форм HTML, создаваемых тэгом `<FORM>`. Ее недостатком является реализация простейшего сценария вида: "Я вам послал сообщение, а вы мне на него ответили". Более того, подобный сценарий существенно влияет на загрузку сети: любой запрос и ответ занимает ресурсы сети. А если для выполнения некоторого действия на сервере переданы неправильные данные? Пользователь получит вместо ожидаемого (возможно достаточно длительное время) ответа всего лишь сообщение о неверно введенных данных.

Чтобы избежать подобных ситуаций, фирмой Netscape был разработан специальный язык сценариев JavaScript. Программы, написанные на этом язы-

ке, встраиваются в документ HTML и интерпретируются браузером, используемым для его просмотра. Подобная технология снимает нагрузку на сеть, избавляя пользователя от ненужных пересылок недостоверной информации, ибо теперь можно написать выполняемый на стороне клиента код для проверки введенных данных. Фирма Microsoft, браузер которой Internet Explorer конкурирует с браузером Netscape Navigator фирмы Netscape, разработала и активно продвигает собственный язык сценариев — VBScript, являющийся подмножеством широко используемого для разработки Windows-приложений языка Visual Basic.

Языки сценариев действительно делают HTML-страницы интерактивными. Содержание страницы может зависеть от желания пользователя, однако для его изменения все равно необходимо производить загрузку новой страницы либо во фрейм отображаемой страницы, либо в новое окно браузера. Причем эти изменения связаны с необходимостью взаимодействия с элементами управления: кнопка, поле ввода, переключатель и т. п. Хотелось бы, чтобы страницы действительно стали динамичными, как наш меняющийся мир: за поворотом мы видим открывающийся новый вид, не взаимодействуя ни с какими "элементами управления". Двигаясь по странице и перемещая по ней курсор мыши, хочется увидеть новую информацию, скрытую до тех пор, пока курсор не пройдет, например, над определенным словом или изображением, расположенным на странице. Поместив курсор мыши на некоторый элемент списка, хотелось бы увидеть уточняющую информацию, скрытую для пользователя до этого момента. Или, указав просто на слово "дальше", увидеть на странице очередную картину любимого художника. Все это реализуется с помощью так называемого динамического языка разметки страниц (Dynamic HTML), который, по существу, состоит из трех компонентов — каскадной таблицы стилей (CSS), JavaScript и HTML, соединенных объектной моделью документа (DOM).

Перечисленные технологии являются мощными инструментами создания интерактивных Web-страниц, однако они ограничены возможностями соответствующих языков сценариев, которые напрямую связаны с располагаемыми на странице элементами, задаваемыми тэгами языка HTML. Апплеты Java и элементы управления ActiveX, создаваемые с помощью современных языков программирования, поддерживающих объектно-ориентированные технологии, позволяют внедрять в документ программируемые объекты и взаимодействовать с ними, меняя их свойства и вызывая их методы посредством языков сценариев.

Перечисленным трем технологиям создания интерактивных Web-документов и посвящена данная часть книги. Приводится достаточное число примеров, позволяющих оценить их эффективность, а также научиться применять эти технологии для разработки собственных интерактивных документов.

ГЛАВА 9

Выполняемые сценарии

Основы объектно-ориентированных технологий

Данный раздел посвящен основам объектно-ориентированных технологий (ООТ). Без знания основополагающих понятий ООТ невозможно понимание программирования на языках сценариев, которые реализуют простую объектную модель, непосредственно связанную с документом HTML. Не имея никакого представления о событийных приложениях, трудно создать интерактивные, встроенные в документ сценарии.

Что такое программный объект

Реальный мир, в котором мы живем, "населен" разнообразными объектами. Нам не надо определять, что это такое: от момента рождения до смерти мы видим и взаимодействуем с объектами. Одни объекты статичны (дом, в котором мы живем), другие объекты могут перемещаться при определенном воздействии на них (машина, в которой мы ездим), третьи могут развиваться сами по себе (мы сами, животные, растения). Все объекты непохожи друг на друга. Не найти двух совершенно одинаковых домов, деревьев даже одного вида, не говоря уже о животных и людях. Однако все объекты похожи друг на друга в том, что каждый объект обладает некоторым внешним видом, поведением (что он умеет делать), а также возможностью взаимодействия с другими объектами. Конечно, такая грубая абстракция не может являться реальной моделью внешнего мира, но она оказывается полезной при моделировании программным способом реально существующих объектов внешнего мира.

Программный объект также обладает некоторым внешним видом или *свойствами*, отражаемыми в значениях его переменных, и поведением или *методами*, задаваемыми в виде его процедур. Причем свойства и методы не существуют отдельно друг от друга, а объединены вместе, образуя единый

объект с новым качеством. Методы "окружают" свойства объекта, не позволяя напрямую обращаться к ним или менять их значения. Говорят, что свойства заключены в некую "капсулу", *инкапсулированы* в объект. Доступ к ним предоставляют методы объекта, которые решают, можно ли изменять значения свойств, или можно получить только значение некоторого свойства, установленного разработчиком программного объекта. Это и есть фундаментальное свойство программного объекта: свойства инкапсулированы в объект и доступ к ним осуществляется только посредством методов, предоставляемых объектом. Программный объект схематически можно представить так, как на рис. 9.1.

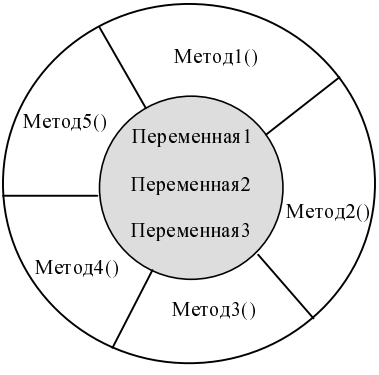


Рис. 9.1. Схематическое представление программного объекта

Как видим, методы предоставляют программный интерфейс для доступа к значениям свойств, заключенных в ядре объекта, и являются некоторой внешней оболочкой, защищающей их от несанкционированного или случайного воздействия.

Инкапсуляцией называется расположение переменных объекта под защитой методов. Она используется для скрытия деталей реализации от других объектов. Не надо знать биологический механизм роста дерева, для практики достаточно знания, что дерево растет и, например, через столько-то лет оно достигнет состояния, при котором его можно использовать в качестве деловой древесины. Водителю автомашины не обязательно представлять детали реализации коробки передач, ему достаточно знать, что, установив ручку переключения скорости в такое-то положение, автомобиль сможет двигаться с заданной скоростью. Главное знать, какие методы может выполнять объект, а как это реализовано — не столь важно. Таким образом, можно всегда изменить внутреннюю реализацию объекта, не меняя части программы, где используется этот объект.

Существование программных объектов самих по себе не имеет никакого смысла. Они дадут преимущества при программировании тогда, когда можно организовать их взаимодействие. И это является следующей основной концепцией объектно-ориентированных технологий. Объекты могут *взаимо-*

действовать друг с другом, посылая сообщения с просьбой выполнить некоторый подходящий метод или выполняя метод в ответ на запрос системы. Взаимодействуя, объекты образуют единое целое — программу.

Любое сообщение состоит из трех компонентов:

1. Имя объекта, которому оно адресовано.
2. Имя метода, который объект-адресат должен выполнить.
3. Необязательные параметры, необходимые для выполнения метода.

Этих трех компонентов достаточно, чтобы объект точно выполнил, что от него хотят.

Здесь мы не будем углубляться в понятия класса, перекрытие методов, механизмов наследования и полиморфизма, которые реализуются объектно-ориентированными системами программирования, например, Java, Visual C++, Delphi. Языки программирования сценариев, как отмечалось выше, реализуют простую объектно-ориентированную модель, и для понимания функционирования и создания сценариев вполне достаточно приведенной информации о программных объектах.

Обратиться к свойству или методу объекта можно с использованием стандартного для объектно-ориентированных языков синтаксиса — так называемой *точечной нотации*. Разные объекты могут иметь свойства и методы с одинаковыми названиями. Чтобы указать, метод какого объекта вызывается, перед именем метода указывается имя объекта, отделенного точкой. Например, если необходимо вызвать метод `рост` объекта `липа`, это можно сделать с помощью следующего оператора:

```
липа.рост(. . .)
```

где в скобках задаются необходимые для выполнения метода параметры.

Событийные приложения

До появления операционных систем с графическим интерфейсом (типа Windows) было распространено так называемое "процедурное" программирование, суть которого заключается в том, что программа жестко определяет, когда и в какой последовательности вызываются те или иные процедуры, в совокупности составляющие программу. Программист должен был заранее разработать и реализовать алгоритм выполнения своей программы. При ее запуске она жестко следовала инструкциям вызова соответствующих процедур.

По-иному выглядит работа приложения с графическим интерфейсом. Оно должно реагировать на действия пользователя: нажал ли он кнопку, а если нажал — то какую, хочет ли он выполнить команду какого-либо меню и т. п. В этом случае выполняемая программа не должна следовать один раз и навсегда заданному алгоритму выполнения. Она должна иметь возможность

запускать на выполнение процедуры, реализующие действие, которое желает выполнить в данный момент пользователь приложения.

Такая технология реализуется с помощью концепции *события*, которое представляет собой действие пользователя (например, нажатие кнопки) или сообщение, генерируемое операционной системой (открылось закрытое окном другого приложения окно нашего приложения). В ООТ любое событие представляется объектом, обратившись к свойствам которого можно получить некоторые параметры сгенерированного события.

События и сообщения системы тесно связаны: любое событие является причиной посылки некоторого сообщения операционной системе. И в то же время некоторые сообщения операционной системы, адресованные приложению, представляются в нем в виде некоторых объектов. Таким образом, можно организовать перехват событий в приложении и написать собственную *процедуру обработки события*, выполняющей все необходимые действия, которые, например, должны быть ассоциированы с нажатием некоторой кнопки графического интерфейса приложения.

Объектные модели языков сценариев

Объектные модели языков сценариев тесно связаны с тэгами HTML. При загрузке страницы HTML в браузер интерпретатор языка создает объекты со свойствами, определяемыми значениями параметров тэгов страницы. Говорят, что браузер отражает HTML-страницу в свойствах объектов, и иногда этот процесс называют *отражением* (reflection). Созданные объекты существуют в виде иерархической структуры, отражающей структуру самой HTML-страницы. На верхнем уровне расположен объект `window`, представляющий собой активное окно браузера. Далее вниз по иерархической лестнице следуют объекты `frame`, `document`, `location` и `history`, представляющие соответственно фрейм, непосредственно сам документ, адрес загружаемого документа и список ранее загружавшихся документов, и т. д. Значения свойств объектов отражают значения соответствующих параметров тэгов страницы или установленных системных параметров. Более подробно модель объектов JavaScript для клиента рассматривается ниже в этой главе в разделе *"Объекты клиента и обработка событий"*.

Для правильного использования объектных моделей следует четко понимать, как браузер компонует страницы и, тем самым, создает иерархию объектов. При загрузке страницы просматриваются сверху вниз, тем самым последовательно происходит компоновка страницы и ее отображение в окне браузера. А это означает, что и объектная модель страницы также формируется последовательно, по мере ее обработки. Поэтому невозможно обратиться из сценария, расположенного ранее какой-либо формы на странице, к элементам этой формы. Всегда следует помнить о том, что браузер последовательно сверху вниз интерпретирует содержимое HTML-страницы.

Еще один аспект работы с объектами языков сценариев заключается в том, что нельзя изменить свойства объектов. Браузер обрабатывает страницу *только* один раз, компонуя и отображая ее. Поэтому попытка в сценарии изменить свойство отображенного элемента страницы обречена на провал. Только повторная загрузка страницы приведет к желаемому результату.

Примечание

Можно динамически изменять содержимое полей ввода элементов форм HTML. Это осуществляется установкой свойства `value` соответствующего элемента в сценарии, встроенном в HTML-страницу.

Язык создания сценариев JavaScript

Язык программирования JavaScript разработан фирмой Netscape для создания интерактивных HTML-документов. Это объектно-ориентированный язык разработки встраиваемых приложений, выполняющихся как на стороне клиента, так и на стороне сервера. Синтаксис языка очень похож на синтаксис языка Java — поэтому его часто называют Java-подобным. Клиентские приложения выполняются браузером просмотра Web-документов на машине пользователя, серверные приложения выполняются на сервере.

Примечание

Языки Java и JavaScript являются совершенно разными языками, ориентированными на выполнение разных задач в сети, но дополняющими друг друга при создании сложных сетевых приложений.

При разработке обоих типов приложений используется общий компонент языка, называемый ядром и включающий определения стандартных объектов и конструкций (переменные, функции, основные объекты и средство LiveConnect взаимодействия с Java-апплетами), и соответствующие компоненты дополнений языка, содержащие специфические для каждого типа приложений определения объектов. На рис. 9.2 схематически представлено взаимодействие компонентов JavaScript при создании клиентских и серверных приложений.

Клиентские приложения непосредственно встраиваются в HTML-страницы и интерпретируются браузером по мере отображения частей документа в его окне. Серверные приложения для увеличения производительности предварительно компилируются в промежуточный байт-код. В данном разделе описывается ядро JavaScript версии 1.3 и дополнительные возможности разработки клиентских приложений.

Примечание

Фирма Microsoft разработала свой интерпретатор языка JavaScript, который она называет JScript.

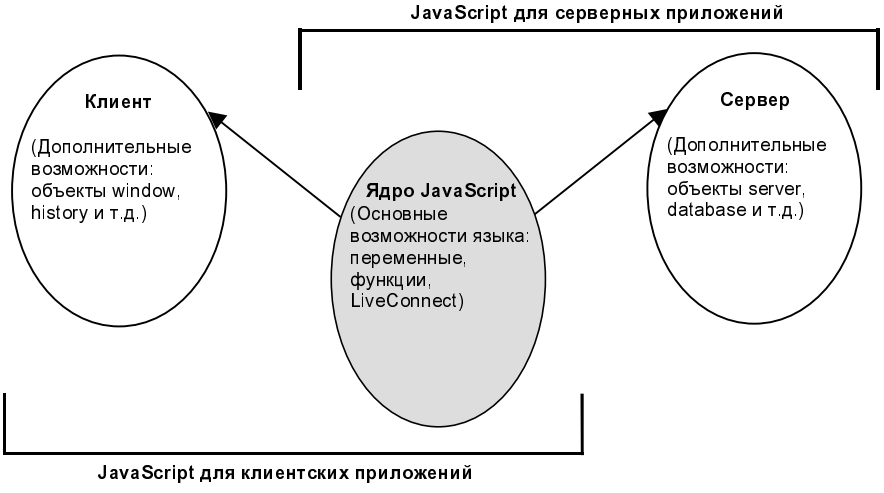


Рис. 9.2. Компоненты JavaScript

Прежде чем переходить к обзору языка JavaScript, перечислим основные области его использования при создании интерактивных HTML-страниц:

- Динамическое создание документа с помощью сценария
- Оперативная проверка достоверности заполняемых пользователем полей форм HTML *до* передачи их на сервер
- Создание динамических HTML-страниц совместно с каскадными таблицами стилей и объектной моделью документа
- Взаимодействие с пользователем при решении "локальных" задач, решаемых приложением JavaScript, встроенном в HTML-страницу

Общий обзор языка

В настоящее время трудно найти в Интернете HTML-страницу, не содержащую ни одного оператора языка JavaScript. Любой Web-мастер или создатель собственной страницы в Интернете заботится о том, чтобы как можно большее число потенциальных посетителей разработанного сайта или личной страницы увидели в окне своего браузера именно то, что задумывалось разработчиком страницы. Дело в том, что наиболее популярные на настоящий момент браузеры могут поддерживать не все существующие технологии, реализованные в HTML, или использовать их несколько отлично друг от друга. Поэтому практически любая страница содержит определение и вызов функции языка JavaScript для идентификации используемого пользователем браузера, а также его версии.

Синтаксис языка

Приложение JavaScript представляет набор операторов языка, последовательно обрабатываемых встроенным в браузер интерпретатором. Каждый оператор можно располагать в отдельной строке. В этом случае разделитель (`;`), отделяющий один оператор от другого, не обязателен. Его используют только в случае задания нескольких операторов на одной строке. Любой оператор можно расположить в нескольких строках без всякого символа продолжения. Например, следующие два вызова функции `alert` эквивалентны:

```
alert("Подсказка");
alert(
"Подсказка"
);
```

Строковый литерал обязательно должен располагаться на одной строке. Если необходимо задать его в нескольких строках, то следует разбить его на более мелкие строковые литералы и использовать оператор конкатенации строк для соединения полученных мелких строк в одну длинную. В этом случае каждый литерал можно располагать на отдельной строке.

Для удобства чтения кода приложения в нем можно расположить комментарии. Любая последовательность символов, следующая за двумя косыми чертами (`//`), рассматривается как комментарий. Этот прием позволяет задать комментарий, расположенный на одной строке. Для задания многострочных комментариев используется синтаксис, заимствованный из языков Java и C. Любая последовательность символов, заключенная между символами (`/*`) и (`*/`), интерпретируется как комментарий.

Язык JavaScript чувствителен к регистру. Это означает, что строчные и прописные буквы алфавита считаются разными символами.

Размещение операторов языка на странице

Встроить сценарий JavaScript в HTML-страницу можно несколькими способами:

- Задать операторы языка внутри тэга `<SCRIPT>` языка HTML
- Указать файл с кодом JavaScript в параметре `SRC` тэга `<SCRIPT>`
- Использовать выражения JavaScript в качестве значений параметров тэгов HTML
- Определить обработчик событий в тэге HTML

Использование тэга *<SCRIPT>*

Для внедрения в HTML-страницу сценария JavaScript в спецификацию языка HTML был введен тэг-контейнер `<SCRIPT>...</SCRIPT>`, внутри которого

могут располагаться операторы языка JavaScript. Обычно браузеры, не поддерживающие какие-нибудь тэги HTML, просто их игнорируют, анализируя, однако, содержимое пропускаемых тэгов с точки зрения синтаксиса языка HTML, что может приводить к ошибкам при отображении страницы. Во избежание подобной ситуации следует помещать операторы языка JavaScript в контейнер комментария `<!-- ... -->`, как показано ниже

```
<SCRIPT [LANGUAGE="JavaScript"]>
<!--
       операторы JavaScript
//-->
</SCRIPT>
```

Параметр `LANGUAGE` задает используемый язык сценариев. В случае языка JavaScript его значение задавать не обязательно, так как этот язык используется браузерами по умолчанию. Для языка сценариев VBScript необходимо явно задать значение этого параметра в виде строки `"VBScript"`.

Примечание

Символы (`//`) перед закрывающим тэгом комментария `-->` являются оператором комментария JavaScript. Он необходим для правильной работы интерпретатора.

Документ может содержать несколько тэгов `<SCRIPT>`, расположенных в любом месте документа. Все они последовательно обрабатываются интерпретатором JavaScript по мере отображения частей документа в окне браузера. В связи с этим ссылка на переменную, определенную в сценарии, размещенном в конце документа, может привести к генерации ошибки интерпретатора при обращении к такой переменной из сценария в начале документа. Поэтому рекомендуется размещать сценарии с глобальными функциями и переменными в разделе `<HEAD>` документа. В этом случае все определения обрабатываются интерпретатором в начале загрузки документа и хранятся в памяти с первых моментов отображения документа в окне браузера.

Задание файла с кодом JavaScript

Тэг `<SCRIPT>` имеет параметр `SRC`, позволяющий связать встраиваемый сценарий с внешним файлом, содержащим программный код на языке JavaScript. В качестве значения параметра задается полный или относительный URL-адрес ресурса. Задание закрывающего тэга `</SCRIPT>` обязательно, независимо от того, заданы или нет операторы внутри тэга. Следующий фрагмент кода связывает документ HTML с файлом-источником, содержащим некоторый набор функций:

```
<SCRIPT SRC="http://home.bhv.com/functions/jsfuncs.js">
       [операторы JavaScript]
</SCRIPT>
```

Примечание

Связываемый внешний файл не должен содержать тэгов HTML и должен иметь расширение JS.

Операторы внутри тэга `<SCRIPT>` игнорируются браузером, если только не произошла ошибка при включении файла в страницу, например, файл не найден. Можно разместить внутри тэга операторы, выводящие сообщение об ошибке загрузки файла.

Следующий сценарий использует метод `alert` объекта `document` для вывода диалогового окна с сообщением:

```
<SCRIPT SRC="http://home.bhv.com/functions/jsfuncs.js">
<!—
       document.alert("Не загрузился файл сценария!")
//-->
</SCRIPT>
```

Элементы JavaScript в параметрах тэгов HTML

Переменные и выражения JavaScript можно использовать в качестве значений параметров тэгов HTML. Эта процедура аналогична процедуре встраивания числовых или символьных примитивов HTML. Элементы JavaScript также располагаются между амперсандом (`&`) и точкой с запятой (`;`), но должны заключаться в фигурные скобки `{}` и использоваться *только* в качестве значений параметров тэгов HTML.

Пусть определена переменная `barWidth`, и ей присвоено значение 75. Следующий тэг нарисует горизонтальную линию длиной в 75% от горизонтального размера окна браузера:

```
<HR WIDTH="&{barWidth};%" ALIGN="LEFT">
```

Предупреждение

Нельзя использовать элементы JavaScript в тексте HTML. Они интерпретируются только тогда, когда расположены справа от параметра и задают его значение. Например, попытка использовать значение переменной `myVar` в следующем фрагменте

```
<H4> &{myVar}; </H4>
```

обречена на провал. Вместо ожидаемого значения переменной `myVar` браузер отобразит строку `myVar`.

Обработчики событий

Для совместимости с языками сценариев в некоторые тэги HTML были введены специальные параметры обработки возникающих событий. Значения-

ми этих параметров могут быть операторы языка JavaScript. Обычно в качестве значения задается имя функции, которая вызывается, когда происходит соответствующее событие, определяемое параметром обработки события. Имя параметра начинается с приставки `on`, за которым следует имя самого события. Например, параметр обработки события `Click` (щелчок кнопкой мыши) будет иметь имя `onClick`.

События в основном связаны с действиями, производимыми пользователем с элементами форм HTML. Поэтому чаще всего перехват и обработка событий задается в параметрах элементов форм, что позволяет проверить введенную информацию *перед* ее отправкой на обработку CGI-сценарием.

Здесь кратко остановимся на функциях JavaScript. *Функция* или *процедура* — это именованная последовательность операторов, которая выполняет определенную задачу и может возвращать некоторое значение. Функция определяется оператором `function`, имеющем следующий синтаксис:

```
function имя_функции([параметры]) {
        [операторы JavaScript]
        [return значение]
}
```

Параметры, передаваемые функции, разделяются запятыми. Необязательный оператор `return` в теле функции (блок операторов, заключенный в фигурные скобки), определяет возвращаемое функцией значение.

Следует четко понимать различие между объявлением функции и ее вызовом. Объявление функции только задает ее имя и определяет, что она будет делать при ее вызове. Непосредственное выполнение функции осуществляется, когда в сценарии вызывают ее и передают действительные параметры.

Определение необходимых функций следует осуществлять в тэге `<HEAD>`, так как все определенные в нем операторы сценария интерпретируются до отображения страницы, и, таким образом, будут известны в процессе отображения всей страницы.

Следующий пример демонстрирует задание функции и ее вызов в процессе формирования страницы документа.

Пример 9.1. Задание функции и ее вызов

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- //Скрыть сценарий от браузеров: не поддерживающих JavaScript
function square(number) {
alert("Мне надо вычислить функцию и потом сформировать документ!");
return number * number;
}
```

```
//-->
</SCRIPT>
</HEAD>
<BODY>
<P>Начинается отображение страницы, в которую внедрен сценарий вычисления
функции</P>
<SCRIPT>
<!--
document.write("Значение, которое вычислялось, равно ", square(5), ".");
//-->
</SCRIPT>
<P> Теперь формирование страницы закончено.
</BODY>
```

В тэге `<HEAD>` задано описание функции `square()`, которая возвращает квадрат значения своего параметра, а также отображает окно сообщения. Вызов функции осуществляется в сценарии, размещенном в теле HTML-документа. В этом сценарии используется метод `write` объекта `document` для формирования вывода в HTML-страницу.

При загрузке этого документа в браузер Internet Explorer отображается первый абзац, а затем выводится окно сообщения, как на рис. 9.3.

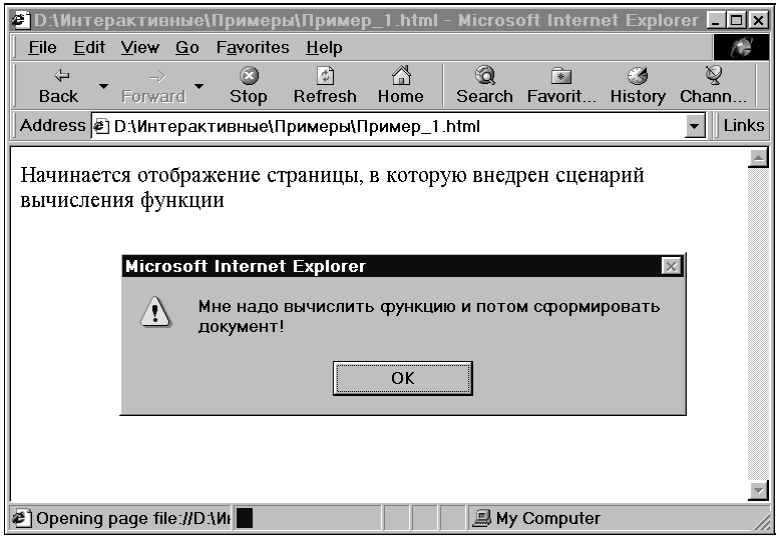


Рис. 9.3. Отображение окна сообщения

Нажатие на кнопку **OK** закрывает окно сообщения, и продолжает отображение страницы. Окончательный вид полученной страницы можно видеть на рис. 9.4.

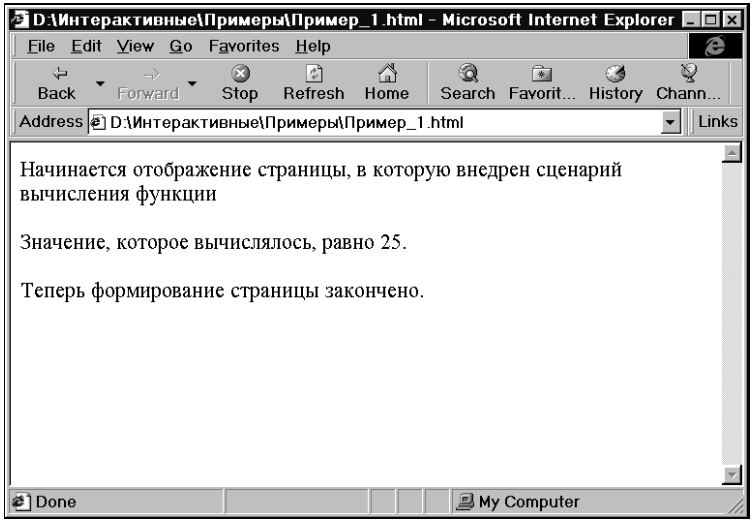


Рис. 9.4. Динамически сформированная страница

Следующий пример демонстрирует прием явного вызова функции из сценария. Стилем хорошего программирования, однако, является неявный вызов функций через параметры обработки событий элементов форм.

Пример 9.2. Прием явного вызова функции из сценария

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- //Скрыть сценарий от браузеров: не поддерживающих JavaScript
function validate(form) {
       if(form.value >=30) {
              alert("Да Вы уже взрослый человек!");
              }
       else {
              alert("Вы еще повзрослеете!");
              }
}
//-->
</SCRIPT>
</HEAD>
<BODY>
<FORM name="Form_1">
Ваш возраст: <INPUT type="text" size=5 name="age"><HR>
<INPUT type="button" value="Подтвердите"
       onClick="validate(this.form.age)>
</FORM>
</BODY>
```

В этом примере обработчик события `onClick` кнопки формы связан с вызовом функции `validate`, которой передается значение, введенное в текстовое поле `age`. Имя поля задается параметром `name`. При нажатии на кнопку вызывается функция `validate`, отображающая сообщение в зависимости от введенного значения.

Параметр `name` элемента формы задает символическое имя элемента, которое можно использовать в операторах сценария для ссылки на соответствующий элемент. Передаваемый функции параметр `this.form.age` использует синтаксис объектно-ориентированных языков, обозначающий элемент с именем `age` (текстовое поле) формы. Ключевое слово `this` языка JavaScript означает в данном случае ссылку на текущую форму. В функции используется свойство `value` элемента Текстовое поле формы для анализа введенного пользователем значения.

На рис. 9.5 показан результат взаимодействия с загруженной в браузер страницей.

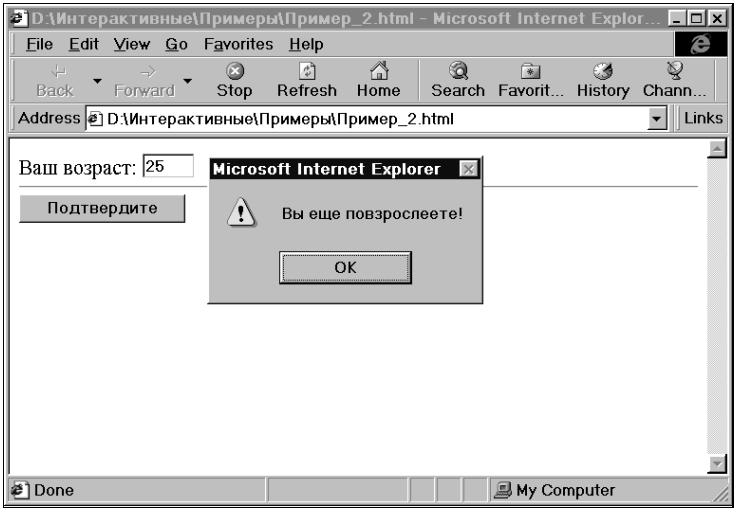


Рис. 9.5. Результат взаимодействия с кнопкой формы HTML

Язык ядра JavaScript

Этот раздел посвящен элементам языка, общим для клиентской и серверной частей JavaScript. Здесь вводятся основные понятия языка, операторы и стандартные объекты и функции.

Переменные и литералы

Как и любой другой язык программирования, JavaScript использует переменные для хранения данных определенного типа. Реализация JavaScript

является примером языка свободного использования типов. В нем не обязательно задавать тип переменной. Ее тип зависит от типа хранимых в ней данных, причем при изменении типа данных меняется и тип переменной.

JavaScript поддерживает четыре простых *типа данных*:

- Целый
- Вещественный
- Строковый
- Булевый, или логический

Для присваивания переменным значений основных типов применяются *литералы* — буквальные значения данных соответствующих типов.

Целые литералы являются последовательностью цифр и представляют обычные целые числа со знаком или без знака:

```
 123   // целое положительное число
-123   // целое отрицательное число
+123   // целое положительное число
```

Для задания *вещественных* литералов используется синтаксис чисел с десятичной точкой, отделяющей дробную часть числа от целой, или запись вещественных чисел в научной нотации с указанием после символа "e" или "E" порядка числа. Пример правильных вещественных чисел:

```
1.25   0.125e01      12.5E-1       0.0125E+2
```

Строковый литерал — последовательность алфавитно-цифровых символов, заключенная в одинарные (') или двойные кавычки ("), например: "Анна", 'АННА'.

Примечание

При задании строковых переменных нельзя смешивать одинарные и двойные кавычки. Недопустимо задавать строку, например, в виде "Анна'.

Предупреждение

Двойные кавычки — это один самостоятельный символ, а не последовательность двух символов одинарных кавычек.

Если в строке нужно использовать символы кавычек, то строковый литерал необходимо заключать в кавычки противоположного вида:

```
"It's a string"       //Значение строки равно It's a string
```

Булевы литералы имеют два значения: `true` и `false`, и используются для обработки ситуаций да/нет в операторах сравнения.

Каждая переменная имеет имя, которое должно начинаться с буквы латинского алфавита, либо символа подчеркивания "_", за которыми следует лю-

бая комбинация алфавитно-цифровых символов или символов подчеркивания. Следующие имена являются допустимыми именами переменных

```
Temp1
MyFunction
_my_Method
```

Определить переменную можно двумя способами:

- Оператором `var`
- Оператором присваивания (=)

Оператор `var` используется как для задания, так и для инициализации переменной и имеет синтаксис:

```
var имя_переменной [= начальное_значение];
```

Необязательный оператор присваивания задает данные, которые содержит переменная. Их тип определяет и тип переменной. Например, следующий оператор

```
var weekDay = "Пятница";
```

задает переменную `weekDay`, присваивает ей строковое значение `"Пятница"`, и тем самым определяет ее тип как строковый.

Если при определении переменной ей не присвоено никакого значения, то ее тип не определен. Ее тип будет определен только после того, как ей будет присвоено некоторое значение оператором присваивания `=`.

Этот оператор можно использовать в любом месте программы, меняя тем самым тип переменной. Это обстоятельство отличает язык JavaScript от строго типизированных языков программирования (например, Java или C++), в которых тип переменной должен быть определен до ее использования. В следующем фрагменте кода переменная `weekDay` меняет свой тип:

```
var weekDay           // Переменная определена, но тип не известен
.       .       .
weekDay = "Пятница"; // Строковый тип
.       .       .
weekDay = 5;          // Целый тип
```

Выражения и операторы

Выражение — это комбинация переменных, литералов и операторов, в результате вычисления которой получается одно единственное значение, которое может быть числовым (целым или вещественным), строковым или булевым.

Переменные в выражениях должны быть инициализированы либо в операторе `var`, либо оператором присваивания. Если при вычислении выражения

встречается неинициализированная или вообще не определенная переменная, то интерпретатор генерирует ошибку "`undefined variable`" ("`переменная не определена`"), указывая ее местоположение на странице HTML.

Примечание

В JavaScript определен специальный литерал `null` для обозначения неопределенного значения. Если переменной присвоено значение `null`, то она считается инициализированной, и при вычислении выражения с такой переменной ошибка не генерируется.

Оператор присваивания рассматривается как выражение присваивания, которое вычисляется равным выражению правой части, и в то же время он присваивает вычисленное значение выражения переменной, заданной в левой части оператора.

Кроме выражения присваивания в JavaScript существует три типа сложных выражений:

- Арифметическое (вычисляемым значением является число)
- Строковое (вычисляемым значением является строка)
- Логическое (вычисляемое значение равно `true` или `false`)

Для построения выражений применяются операторы, соответствующие типу выражения.

Арифметические выражения создаются *арифметическими* операторами (табл. 9.1).

Таблица 9.1. *Арифметические операторы*

Оператор	Название
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Остаток от деления целых чисел
++	Увеличение значения переменной на единицу
--	Уменьшение значения переменной на единицу

Ниже представлены арифметические выражения в операторах присваивания:

```
speed = 5.5;
time = 4;
distance = speed * time;
distance = (speed ++)*time;
```

В последнем операторе скорость (переменная `speed`) увеличивается на единицу и вычисляется пройденное расстояние.

Операторы в выражении вычисляются слева направо с учетом общепринятого старшинства арифметических операций. Скобками можно изменить порядок выполнения арифметических операций.

Кроме простого оператора присваивания (=) существуют сокращенные формы операторов присваивания, совмещенных с арифметическими операторами, в которых производятся арифметические действия над левым и правым операндами и результат присваивается переменной, заданной левым операндом. Все они перечислены в табл. 9.2.

Таблица 9.2. *Сокращенные операторы присваивания*

Оператор	Значение
`x *= y`	`x = x * y`
`x /= y`	`x = x / y`
`x += y`	`x = x + y`
`x -= y`	`x = x - y`
`x %= y`	`x = x % y`

Для создания *логических* выражений используются операторы *сравнения* и *логические* операторы, применяемые к переменным любого типа.

Операторы *сравнения* аналогичны таковым в других языках программирования. Их список представлен в табл. 9.3.

Таблица 9.3. *Операторы сравнения*

Оператор	Название
`==`	Равно
`!=`	Не равно
`>=`	Больше или равно
`<=`	Меньше или равно
`>`	Строго больше
`<`	Строго меньше

При использовании этих операторов в выражении оно вычисляется равным `true`, если соответствующее сравнение верно, в противном случае значение выражения равно `false`.

Логические операторы представлены в табл. 9.4. В примерах предполагается, что переменная `var1 = 'Кит'`, `var2 = 'Кот'`, `var3 = false`.

Таблица 9.4. *Логические операторы*

Оператор	Синтаксис	Описание	Пример
`&&` (логическое И)	`выраж1 && выраж2`	Возвращает `выраж1`, если оно преобразуется или равно `false`, иначе `выраж2`	`var1 && var2` (равно `'Кот'`) `var2 && var3` (равно `false`)
`||` (логическое ИЛИ)	`выраж1 || выраж2`	Возвращает `выраж1`, если оно преобразуется или равно `true`, иначе `выраж2`	`var1 || var2` (равно `'Кит'`) `var3 || var1` (равно `'Кит'`) `var3 || false` (равно `false`)
`!` (логическое НЕ)	`!выраж`	Если `выраж` равно `true`, возвращает `false`; если `выраж` равно `false`, возвращает `true`	`!var1` (равно `false`) `!var3` (равно `true`)

Логические операторы и операторы сравнения используются в операторах цикла и условия для проверки завершения цикла или выполнения определенной группы операторов.

Строковые операторы используются для создания *строковых* выражений. В JavaScript, собственно говоря, существует только один строковый оператор — оператор конкатенации (соединения) строк (+), если не считать сокращенной формы оператора присваивания со сложением (+=). Этот оператор присоединяет к строковому значению первого операнда строковое значение второго, получая результат, равный соединению строк:

```
string = "Моя" + "строка"; // Значение переменной string равно "Моястрока"
```

Примечание

Оператор `+` может использоваться со смешанными типами операндов. Все операнды приводятся к строковому типу, если хотя бы один из операндов содержит строковый литерал. Например, выражение `"Май" + 1.999e3` будет вычислено равным строке `"Май1999"`, а `"Май" + t` будет равно `"Майtrue"`, если переменная `t` содержит булево значение `true`.

Условный оператор является единственным оператором, использующим три операнда. Его значением является один из двух операндов, определяемый из условия истинности третьего. Синтаксис его таков:

```
(условие) ? знач1 : знач2;
```

Если операнд `условие` имеет значение `true`, то результатом вычисления условного оператора будет `знач1`, в противном случае — `знач2`. Например, оператор

```
range = (mark <= 2) ? "Пересдача" : "Зачтено";
```

присваивает переменной `range` значение `"Пересдача"`, если переменная `mark` меньше либо равно `2`, в противном случае ей присваивается значение `"Зачтено"`.

Кроме перечисленных выше операторов в JavaScript существует большая группа операторов для *поразрядных действий* с данными. В них содержимое каждого оператора рассматривается как последовательность битов, а не как данные строкового, числового или булевого типов. Их описание можно найти в любом учебнике по языку JavaScript или в технических описаниях на сервере фирмы Netscape по адресу: **http://developer.netscape.com/**.

Стандартные объекты и функции

В ядре JavaScript определены объекты и функции, которые можно использовать вне контекста загруженной страницы. Они доступны как для сценариев на стороне клиента, так и для сценариев на стороне сервера.

Объект *Array*

В JavaScript нет типа данных массив, но с помощью объекта `Array` можно создавать массивы в приложениях и манипулировать ими. Методы этого объекта позволяют сортировать, объединять, записывать в обратном порядке содержимое массивов и выполнять многие другие действия. Наиболее часто используется свойство объекта `Array`, позволяющее определять количество элементов, содержащихся в массиве.

Массив — упорядоченный набор однородных данных, к элементам которого можно обращаться по имени и индексу. Массив создается оператором `new` и *конструктором* массива — системной функцией `Array`, инициализирующей элементы массива. Создать массив можно одной из следующих конструкций:

```
имя_массива = new Array([элемент0, элемент1, . . ., элементN]);
имя_массива = new Array([длина_массива]);
```

При использовании первого синтаксиса конструктору массива в качестве параметров передаются значения элементов массива, во второй конструкции задается длина массива. Можно использовать конструктор без параметров, но в этом случае только определяется, что переменная с данным именем используется в качестве массива. Элементы самого массива не заданы, и поэтому к ним нельзя обратиться, пока в сценарии им явно не будут присвоены значения.

Для получения значения элемента массива необходимо в квадратных скобках рядом с именем массива указать порядковый номер элемента. Элемент

массива можно использовать в выражениях и в левой части оператора присваивания:

```
a[0] = "1"; a[1] = 2;
c = b[2]*c[3];
```

Примечание

В JavaScript нумерация элементов массивов начинается с нуля. Поэтому первый элемент имеет индекс 0, второй — 1 и т. д.

Примечание

В JavaScript версии 1.2 и выше вторая форма задания массива с указанием числа элементов в конструкторе создает массив из одного элемента с целым значением. В предыдущих версиях языка создается массив с заданным числом элементов, значения которых не определены.

Массив, являясь объектом, обладает методами, которые вызываются с использованием обычной для объектно-ориентированных языков точечной нотации. В табл. 9.5 перечислены методы объекта **Массив**.

Таблица 9.5. *Методы объекта* ***Массив***

Метод	Действие
concat	Объединяет два массива в один
join	Соединяет все элементы массива в одну строку
pop	Удаляет последний элемент из массива и возвращает его значение
push	Добавляет один или несколько элементов в конец массива и возвращает последний добавленный элемент
reverse	Переставляет элементы массива в обратном порядке: первый элемент становится последним, а последний первым
shift	Удаляет первый элемент массива и возвращает его значение
slice	Создает сечение массива в виде нового массива
splice	Добавляет и/или удаляет элементы из массива
sort	Сортирует элементы массива
unshift	Добавляет один или более элементов в начало массива и возвращает новую длину массива

Предположим, что определены два массива

```
array1 = new Array("Первый","Второй","Третий");
array2 = new Array("Один","Два","Три");
```

Метод `array1.join()` возвратит строку "Первый, Второй, Третий"; метод `array1.sort()` упорядочит элементы массива `array1` (переставив их местами) в алфавитном порядке, а оператор `array1.concat(array2).sort()` объединит два массива в один новый и отсортирует его.

Примечание

Последний оператор является типичным примером точечной нотации в объектно-ориентированных языках. Метод `array1.concat(array2)` возвращает массив, который является объектом. Следовательно, можно выполнить любой метод этого объекта, в данном случае `sort()`.

Присвоить значение элементу массива можно в любом месте программы. Добавление элементов в конец массив можно осуществлять простым присваиванием значения новому элементу, а не только методом `push()`. Для определения длины массива используется свойство `length`. Ниже представлен фрагмент кода, определяющий пятый элемент массива `array1`, и тем самым изменяющий значение его свойства `length` с 3 на 5:

```
array1 = new Array("Первый","Второй","Третий");
l1 = array1.length;
array1[4] = "1";
l2 = array1.length;
document.write(l1); // Напечатает 3
document.write(l2); // Напечатает 5
```

Для задания массивов нескольких размерностей следует значениям элементов массивов присваивать массивы. Подобная техника иллюстрируется следующим фрагментом, в котором создается двумерный массив:

```
a = new Array()
for (i=0; i < 4; i++) {
   a[i] = new Array()
   for (j=0; j < 4; j++) {
      a[i][j] = "["+i+","+j+"]"
   }
}
```

Здесь в первом операторе определяется массив `a`. Далее в цикле элементы этого одномерного массива сами определяются как массивы, элементам которых присваиваются значения. Таким образом создается двумерный массив.

Объект *Date*

Для представления дат в программе JavaScript используется объект `Date`. Он создается, как и любой объект в JavaScript, оператором `new` с помощью конструктора `Date`. Синтаксис оператора создания даты следующий:

```
имя_объекта_дата = new Date([параметры]);
```

В JavaScript дата хранится в виде числа миллисекунд, прошедших от 1 января 1970 года. Если в конструкторе даты отсутствуют параметры, то значением объекта будет текущая дата. Параметром конструктора может быть строка вида "месяц день, год часы:минуты:секунды". Если опустить значения часов, минут и секунд, то по умолчанию они будут иметь значения 0. Можно задать список параметров, задающих год, месяц и день или год, месяц, день, часы, минуты и секунды. Ниже представлены все три способа инициализации объекта `Date`:

```
today = new Date()
Xmas = new Date("January 7, 1999 12:00:00")
Xmas = new Date(99, 1, 7)
Xmas = new Date(99, 1, 7, 12, 0, 0)
```

Методами объекта `Date` можно получать и устанавливать отдельно значения года, месяца, дня недели, часов, минут и секунд. Например, метод `getYear()` возвращает год, метод `setYear()` устанавливает значение года объекта `Date`.

Метод `getTime()` возвращает число миллисекунд, прошедшее с момента времени 1 января 1970 года 00:00:00, метод `setTime()` устанавливает соответствующее значение даты в миллисекундах, заданных в качестве параметра.

Объект *Math*

В свойствах объекта `Math` хранятся основные математические константы, а его методы вычисляют основные математические функции. При обращении к свойствам и методам этого объекта создавать его не надо, но следует явно указывать его имя `Math`. Например, в свойстве `PI` хранится значение числа π и использовать его в программе можно в виде `Math.PI`.

Методы этого объекта включают процедуры вычисления тригонометрических, экспоненциальных, логарифмических и других математических функций. В табл. 9.6 собраны все методы объекта `Math`.

Таблица 9.6. *Методы объекта* `Math`

Метод	Описание
`abs`	Абсолютное значение
`sin, cos, tan`	Стандартные тригонометрические функции; аргумент задается в радианах
`acos, asin, atan`	Обратные тригонометрические функции
`exp, log`	Экспоненциальная функция и функция натурального логарифма
`ceil`	Наименьшее целое, большее или равное значению аргумента

Таблица 9.6 *(окончание)*

Метод	Описание
`floor`	Наибольшее целое, меньшее или равное значению аргумента
`min, max`	Наибольшее или наименьшее значение двух аргументов
`pow`	Показательная функция: `pow(x,y)`=x^y
`round`	Округление аргумента до ближайшего целого
`sqrt`	Квадратный корень

Объект *String*

Когда переменной присваивается строковый литерал, она становится, как указывалось выше в разделе *"Переменные и литералы"*, строковой переменной. На самом деле JavaScript не поддерживает строковых типов, а создает стандартный объект `String`. Таким образом, любая строковая переменная или строковый литерал является объектом `String`, к которому могут быть применены соответствующие методы этого объекта.

Можно явно создать строковый объект, используя ключевое слово `new` и конструктор `String`, как показано ниже:

```
имя_объекта = new String(строка);
```

Параметром конструктора является любая допустимая строка. Например:

```
myString = new String("Строка");
```

Объект `String` имеет единственное свойство `length`, хранящее длину строки, содержащейся в строковом объекте. Так, и `"Строка".length`, и `myString.length` возвращают одинаковые значения 6, равные в первом случае длине строкового литерала, а во втором случае длине строки, содержащейся в строковом объекте.

Объект `String` имеет два типа методов: первые непосредственно влияют на саму строку, например метод `substring`, а вторые возвращают отформатированный HTML вариант строки, например метод `bold`.

Некоторым методам необходимы параметры. Так, метод получения подстроки требует задания двух целых чисел, определяющих позиции начала и конца подстроки, например `substring(2,7)`.

Методы, возвращающие HTML-отформатированные варианты строк, соответствуют тэгам форматирования HTML. Например, следующий оператор вставляет в страницу HTML связь с ресурсом, расположенным по адресу, задаваемому параметром метода `link`:

```
document.write(s.link("http://www.bhv.com"));
```

В документе отобразится содержимое строкового объекта `s`, представленное как связь с соответствующим ресурсом.

В табл. 9.7 перечислены методы строковых объектов.

Таблица 9.7. *Методы объекта* `String`

Метод	Действие
`anchor`	Создает именованную ссылку
`big, blink, bold, fixed, italics, small, strike, sub, sup`	Создает строку в формате HTML
`charAt, charCodeAt`	Возвращает символ или код символа, параметр определяет позицию символа в строке
`indexOf, lastIndexOf`	Возвращает позицию начала или конца в строке заданной подстроки
`link`	Создает гиперсвязь
`concat`	Конкатенация двух строк
`split`	Преобразует строковый объект в массив строк, разбивая строку на подстроки
`slice`	Получает сечение строки
`substring, substr`	Возвращают подмножество строки, заданное либо началом и концом, либо началом и числом символов
`match, replace, search`	Используются для работы с регулярными выражениями
`toLowerCase, toUpperCase`	Переводят содержимое строк в верхний или в нижний регистр соответственно

Примечание

Подробную информацию о работе с методами строкового объекта можно найти в любом учебнике по JavaScript или в Интернете по адресу **http://developer.netscape.com**.

Стандартные функции верхнего уровня

В добавление к стандартным объектам существует несколько функций, для вызова которых не надо создавать никакого объекта. Они находятся вне иерархии объектов JavaScript на так называемом "верхнем уровне".

Полезными при разработке приложений могут оказаться две функции, производящие "синтаксический" анализ своих аргументов: `parseFloat` и `parseInt`.

Функция `parseFloat(параметр)` анализирует значение переданного ей строкового параметра на соответствие представлению вещественного числа в JavaScript. Если в строке при последовательном просмотре обнаруживается символ, отличный от символов, применяемых для формирования вещественных литералов (знаки `+` и `-`, десятичные цифры, точка и символы (`e`) или (`E`)), то она игнорирует оставшуюся часть строки и возвращает то числовое значение, которое ею обнаружено до неправильного символа. Если первый символ в строке не является цифрой, она возвращает значение `"NaN"` (Not a Number — не число).

Аналогично функция `parseInt(строка,[основание])` пытается возвратить целое число по заданному вторым параметром основанию. Если первый символ в строке не является цифрой, она также возвращает значение `"NaN"`.

Эти функции полезны при анализе введенных пользователем данных в полях формы до передачи их на сервер.

Функции `Number(объект)` и `String(объект)` преобразуют объект, заданный в качестве его параметра в число или строку.

Функция `isNaN(параметр)` тестирует значение своего параметра на соответствие нечисловому значению. Если ее параметр действительно оказывается не числом, она возвращает `true`, в противном случае — `false`.

Примечание

Подробную информацию о функциях верхнего уровня можно найти в любом учебнике по JavaScript или в Интернете по адресу **http://developer.netscape.com**.

Операторы управления

Весь набор операторов управления языка можно разбить на три группы:

- Операторы выбора, или условные
- Операторы цикла
- Операторы манипулирования с объектами

В этом разделе кратко обсуждается каждая из указанных групп операторов JavaScript, в совокупности позволяющая создавать высоко интерактивные приложения.

Операторы выбора

К этой группе операторов относятся операторы, которые выполняют определенные блоки операторов в зависимости от истинности некоторого булевского выражения. Это оператор условия `if...else` и переключатель `switch`.

Оператор условия `if` применяется, если необходимо вычислить некоторый блок операторов в зависимости от истинности заданного условия, и имеет следующий синтаксис:

```
if(условие) {
       операторы1
}
[else {
       операторы2
} ]
```

Первая группа операторов `операторы1` выполняется при условии истинности выражения `условие`. Необязательный блок `else` задает группу операторов `операторы2`, которая будет выполнена в случае ложности условия, заданного в блоке `if`.

Примечание

Фигурные скобки, отмечающие группу выполняемых операторов в блоках `if` и `else`, необязательны, если группа состоит из одного оператора.

Совет

Лучше всегда задавать группы выполняемых операторов в блоках `if` и `else` заключенными в фигурные скобки. Программа в этом случае легче читается, и проще производить ее модификацию.

Внутри группы выполняемых операторов могут использоваться любые операторы JavaScript, в том числе и операторы условия. Таким образом, можно создавать группу вложенных операторов условия и реализовывать сложные алгоритмы проверки.

В операторе `switch` вычисляется одно выражение и сравнивается со значениями, заданными в блоках `case`. В случае совпадения выполняются операторы соответствующего блока `case`. Синтаксис этого оператора следующий:

```
switch (выражение){
   case значение1 :
      [операторы1]
      break;
   case значение2 :
      [операторы2]
      break;
   ...
   default :
      [операторы]
}
```

Если значение выражения в блоке `switch` равно `значение1`, то выполняется группа операторов `операторы1`, если равно `значение2`, то выполняется группа операторов `операторы2` и т. д. Если значение выражения не равняется ни одному из значений, заданных в блоках `case`, то вычисляется группа операто-

ров блока default, если этот блок задан, иначе происходит выход из оператора switch.

Необязательный оператор break, задаваемый в каждом из блоков case, выполняет безусловный выход из оператора switch. Если он не задан, то продолжается выполнение операторов в следующих блоках case до первого оператора break или до конца тела оператора switch.

Операторы цикла

Оператор цикла повторно выполняет последовательность операторов JavaScript, определенных в его теле, пока не выполнится некоторое заданное условие. В языке существует два оператора цикла: for и while. Они отличаются механизмом организации цикла.

Оператор цикла for позволяет организовать выполнение блока операторов заданное число раз. Он определяет переменную, называемую *переменной цикла*, которая изменяет свое значение во время выполнения цикла. Условие завершения цикла зависит от значения этой переменной. Оператор имеет следующий синтаксис:

```
for([инициал_выражение];[условие];[изменяющее_выражение]) {
        [операторы]
}
```

Параметром инициал_выражение задается и инициализируется переменная цикла. Это выражение вычисляется один раз в начале выполнения цикла. После этого проверяется истинность выражения условие. Если оно истинно, то выполняется блок операторов тела цикла, ограниченного фигурными скобками; вычисляется изменяющее_выражение, содержащее переменную цикла, и снова проверяется истинность выражения условие. Если оно истинно, то повторяется цикл вычислений, если нет, то оператор цикла завершает свое выполнение.

Примечание

Инициализирующее выражение, изменяющее выражение и условие окончания цикла не являются обязательными, и любое из них может быть опущено. Однако следует оставлять разделитель (;) для правильной интерпретации оставшихся выражений. Например, в случае отсутствия условия завершения цикла заголовок цикла for будет выглядеть следующим образом:

```
for(var i=0;;i+2).
```

Следующий цикл выводит в документ значения степеней двойки:

```
for(var i=0; i<=5; i++){
document.write("<p>2<sup>", i, "</sup> = ", Math.pow(2,i))
}
```

Результат работы этого оператора цикла представлен ниже:

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$

Цикл `while` выполняется пока истинно выражение, задающее условие выполнения цикла. Его синтаксис следующий:

```
while(условие) {
       [операторы]
}
```

Сначала проверяется истинность условия, заданного в заголовке цикла, а затем выполняются (или не выполняются) операторы тела цикла. Проверка истинности условия осуществляется на каждом шаге цикла. Использование этого цикла предполагает, что условное выражение окончания цикла меняется в зависимости от вычисленных значений переменных и выражений в теле цикла. Например, следующий фрагмент кода

```
s = "Example"; i = 0;
while(i<s.length) {
document.write(s.substr(i,1),"<BR>")
i++;
}
```

выведет в документ слово "Example" по одной букве в строке.

Иногда необходимо завершить цикл не по условию, задаваемому в заголовке цикла, а в результате вычисления некоторого условия в теле цикла. Для этой цели в JavaScript существуют операторы `break` и `continue`.

Оператор `break` завершает выполнение цикла и передает управление оператору, непосредственно следующим за оператором цикла. Оператор `continue` прекращает выполнение текущей итерации и начинает выполнение следующей, т. е. в цикле `while` он передает управление на проверку выражения `условие` цикла, а в цикле `for` — на вычисление выражения `изменяющее_выражение`.

Манипулирование объектами

Четыре оператора JavaScript предназначены для работы с объектами. Это оператор `new`, создающий новый объект *(см. выше раздел "Стандартные объекты и функции")*, операторы `for...in` и `with` и ключевое слово `this`.

Оператор `for...in` позволяет организовать цикл по свойствам объекта JavaScript. Синтаксис его следующий:

```
for( переменная_цикла in объект) {
       [операторы]
}
```

Этот цикл производит перебор свойств объекта. В переменной цикла на каждой итерации сохраняется значение свойства объекта. Количество итераций равно количеству свойств, существующих у заданного в заголовке цикла объекта.

В следующем примере функция `properties()`, в качестве параметров которой передаются объект и его имя, используется для отображения в HTML документе всех свойств объекта `Флажок`, созданного на странице тэгом `<INPUT>`.

Пример 9.3. Определение свойств объекта `Флажок`

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- //Скрыть сценарий от браузеров, не поддерживающих JavaScript
function properties(obj,obj_name) {
     var result = ""
     for (var i in obj) {
         result += obj_name + "." + i + " = " + obj[i] + "<BR>"
     }
     result += "<HR>"
     return result
}
-->
</SCRIPT>
</HEAD>
<BODY>
<INPUT TYPE="checkbox" NAME="check1" value="Флажок">
Флажок
<p>
<SCRIPT>
document.write(properties(check1, check1.value))
</SCRIPT>
</BODY>
```

Сама функция описывается в разделе `<HEAD>`, а ее вызов происходит в теле документа. Для связи элементов управления форм со сценарием применяется параметр `NAME` тэга `<INPUT>`. Его значение равно переменной, которую используют в сценарии для ссылки на соответствующий элемент управления.

В качестве имени объекта в функцию передается значение свойства `value` объекта `checkbox`.

Предупреждение

Вызов функции осуществляется после создания в документе элемента управления `checkbox`. Попытка вызвать функцию `properties()` до создания в документе флажка приведет к ошибке интерпретатора, так как объект еще не будет существовать *(см. выше раздел "Объектные модели языков сценариев")*.

Оператор `with` задает объект по умолчанию для блока операторов, определенных в его теле. Это означает, что все встречаемые в операторах этого блока свойства и методы, являются свойствами и методами указанного объекта. Применение данного оператора избавляет от необходимости указывать иерархию принадлежности объекта и сокращает исходный текст программы.

Текст приведенного выше сценария с использованием оператора `with` упростится следующим образом:

```
<SCRIPT>
with( check1) {
document.write(properties(check1, value))
}
</SCRIPT>
```

Здесь свойство `value` относится к объекту `check1`, который указан в заголовке оператора `with`.

Полезно использовать этот оператор для объекта `Math`. Тогда обращение к его свойствам и методам осуществляется без явного указания префикса `Math`. Например:

```
with( Math) {
r = sin(2.0) // Вычисление синуса
l = 2*PI*r // Вычисление длины окружности
}
```

Объекты клиента и обработка событий

Как отмечалось выше, при интерпретации страницы HTML браузером создаются объекты JavaScript, свойства которых представляют значения параметров тэгов языка HTML. Объекты хранятся в виде иерархической структуры, отражая структуру документа. Некоторые тэги HTML являются контейнерами, в которых могут размещаться другие тэги. Например, тэг формы `<FORM>` содержит элементы управления, задаваемые тэгами `<INPUT>`. Эта подчиненность одних тэгов другим и образует структуру документа, отражаясь в иерархической структуре объектов, соответствующих тэгам HTML.

Примечание

Не все свойства объектов могут соответствовать значениям параметров тэгов HTML. Некоторые свойства могут получать значения от установленных свойств самого браузера. Например, свойство `fgColor` объекта `document` отражает установку цвета отображения текста на странице в диалоговом окне предпочтений браузера Netscape Navigator.

Иерархия объектов

На самом верхнем уровне иерархии находится объект `window`, представляющий окно браузера и являющийся "родителем" всех остальных объектов. Расположенные ниже в иерархии объекты могут иметь свои подчиненные объекты. На рис. 9.6 показана структура объектов клиента (браузера).

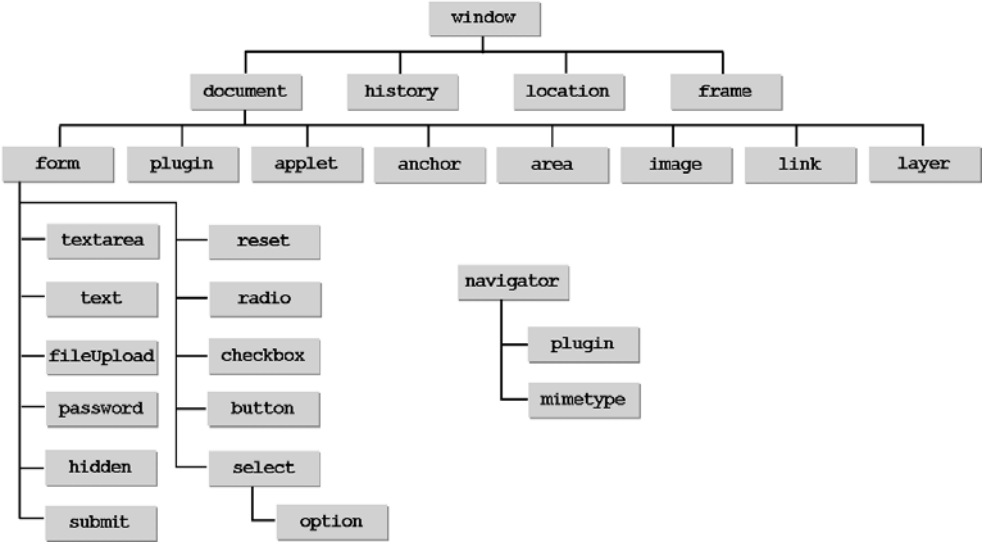


Рис. 9.6. Иерархия объектов JavaScript на стороне клиента

Особняком стоит объект `navigator` с двумя дочерними (подчиненными) объектами. Он относится к самому браузеру, и его свойства позволяют определить характеристики программы просмотра. Свойство `appName` содержит имя браузера (для Internet Explorer, например, его значение равно "`Microsoft Internet Explorer`"), а свойство `appVersion` содержит информацию о версии браузера (например, для Internet Explorer версии 4.01 его значение равно "`4.0 (compatible; MSIE 4.01; Windows 95)`").

Каждая страница в добавление к объекту `navigator` обязательно имеет еще четыре объекта:

- `window` — объект верхнего уровня, свойства которого применяются ко всему окну, в котором отображается документ.

- `document` — свойства которого определяются содержимым самого документа: связи, цвет фона, формы и т. д.
- `location` — свойства которого связаны с URL-адресом отображаемого документа.
- `history` — представляет адреса ранее загружавшихся HTML-страниц.

Кроме указанных объектов страница может иметь дополнительные объекты, зависящие от ее содержимого, которые являются дочерними объектами объекта `document`. Если на странице расположена форма, то все ее элементы являются дочерними объектами этой формы. Для задания точного имени объекта используется точечная нотация с полным указанием всей цепочки наследования объекта. Это возможно, так как объект верхнего уровня имеет свойство, значением которого является объект нижнего уровня. Ссылка на объект осуществляется по имени, заданному параметром `NAME` тэга HTML. Например, пусть в документе задана форма с элементами:

```
<FORM NAME="form1">
Фамилия: <INPUT TYPE = "text" name = "studentName" size = 20>
Курс: <INPUT TYPE = "text" name = "course" size = 2>
</FORM>
```

Для получения фамилии студента, введенного в первом поле ввода, в программе JavaScript следует использовать ссылку `document.form1.studentName.value`, а чтобы определить курс, на котором обучается студент, необходимо использовать ссылку `document.form1.course.value`.

Примечание

При ссылке на формы и их элементы необязательно указывать объект верхнего уровня `document`. В приведенном примере сослаться на значение первого поля ввода можно и так `form1.studentName.value`.

Совет

Для получения свойств объектов можно воспользоваться сценарием примера 9.3.

Свойства и методы ключевых объектов

Не все объекты иерархии интенсивно используются в сценариях JavaScript. В данном разделе перечислены свойства и методы наиболее часто используемых объектов.

Объекты *window* и *Frame*

Объект `window` создается автоматически при запуске браузера, так как для отображения документа необходимо окно. В меню **Файл** (**File**) любого браузера есть команда **Создать** (**New**), позволяющая открыть новое окно и ото-

бразить в нем новый документ, и команда **Закрыть** (**Close**) закрытия окна. Эти действия можно осуществлять программно из приложения JavaScript, применяя методы `open()` и `close()` объекта `window`.

Новое окно создается методом `open()`, который имеет следующий синтаксис:

```
имя_перемен_окна=window.open([имя_файла],[имя_ссылки_окна],[параметры])
```

Здесь:

`имя_перемен_окна` — имя для ссылки на новое окно в операторах JavaScript,

`имя_файла` — полный или относительный URL-адрес открываемого в окне документа,

`имя_ссылки_окна` — имя, указываемое в качестве цели в гипертекстовой ссылке на это окно из другого документа HTML, `параметры` — задают значения параметров окна (ширина, высота, наличие панелей инструментов, полос прокрутки и т. п.).

Примечание

Все три параметра задаются в виде текстовых литералов или переменных и не являются обязательными. Если они все отсутствуют, то открывается новое окно браузера с параметрами по умолчанию.

Например, следующий оператор:

```
winExample=window.open
           ("http://www.bhv.ru/library/index.html",
                "linkWin","toolbar=no,scrollbars=yes")
```

открывает новое окно без панелей инструментов, но с полосами прокрутки, на которое можно ссылаться в сценарии по имени `winExample`, и загружает в него документ, расположенный по адресу **http://www.bhv.ru/library/index.html**. На это окно можно ссылаться из другого документа по имени `"linkWin"`.

Вывод во вновь открытое окно осуществляется методом `write()` объекта `document` этого окна. Например, в примере 9.3 можно отобразить свойства объекта в новом окне с помощью следующего сценария:

```
<SCRIPT>
msgWindow=window.open("","displayWindow","toolbar=no,scrollbars=yes")
msgWindow.document.write(properties(check1, check1.value))
</SCRIPT>
```

Результат выполнения сценария показан на рис. 9.7.

Закрывается окно методом `close()` без параметров. Окно документа, в котором находится сценарий, закрывается одним из следующих операторов:

```
window.close()
self.close()
close()
```

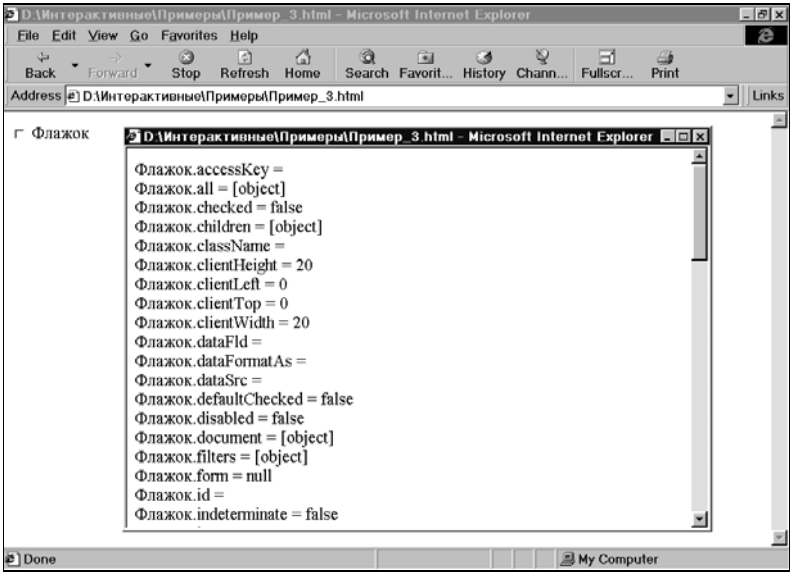


Рис. 9.7. Открытие нового окна и отображение в нем свойств объекта

Для закрытия окна, открытого методом `open()`, необходимо явно указывать имя переменной этого окна:

```
msgWindow.close()
```

Метод `alert()` объекта `window` отображает диалоговое окно с текстом, передаваемым в качестве параметра этому методу. Например, следующий сценарий выводит диалоговое окно с информацией для пользователя, если некоторая переменная меньше нуля:

```
if( myVar < 0) alert("Переменная 'myVar' стала отрицательной!")
```

Эта функция полезна в сценариях проверки правильности заполнения полей формы перед отправкой формы на сервер.

В HTML тэг-контейнер `<FRAMESET>...</FRAMESET>` задает специальный тип окна, называемый набором фреймов. Это окно может отображать несколько независимых, каждый со своими полосами прокрутки фреймов на одном экране. Каждый фрейм, в свою очередь, может отображать определенный документ, расположенный по адресу, указанному в параметре `SRC` тэга `<FRAME>`. Набор фреймов образует страницу, поэтому не надо задавать тэг `<BODY>`.

Тэг-контейнер `<FRAMESET>` может содержать, кроме тэгов `<FRAME>`, определяющих фреймы, другие тэги `<FRAMESET>`, образуя, таким образом, вложенные наборы фреймов. Пример HTML-страницы с вложенными наборами фреймов представлен ниже:

```
<FRAMESET COLS="30%,70%">
   <FRAMESET ROWS="30%,70%">
      <FRAME SRC="./Часы.html" NAME="clockFrame">
      <FRAME SRC="menu.html" NAME="menuFrame">
   </FRAMESET>
   <FRAME SRC="" NAME="contentFrame">
</FRAMESET>
```

Значение параметра `NAME` задает имя, по которому можно ссылаться на соответствующий фрейм в иерархии объектов документа. Отображение страницы с фреймами можно увидеть на рис. 9.8.

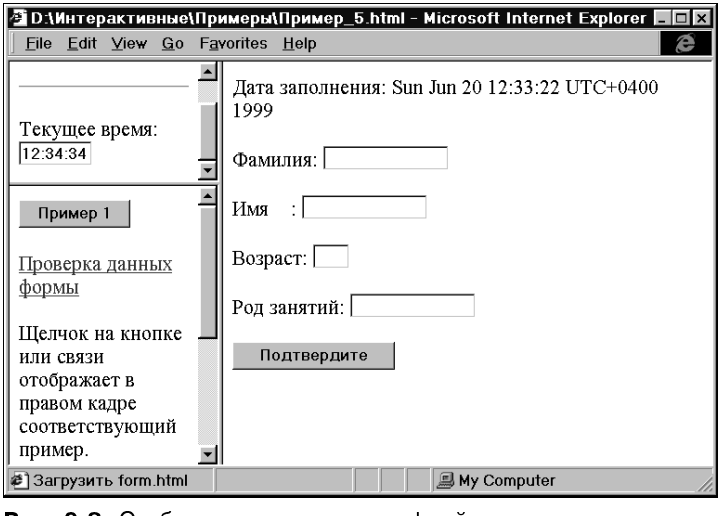


Рис. 9.8. Отображение страницы с фреймами

Эта страница содержит три фрейма, которые образуют иерархическую модель объектов `Frame`. На верхнем уровне расположен объект `top`, являющийся родителем всех трех фреймов на странице (`clockFrame`, `menuFrame` и `contentFrame`). Для ссылки на фреймы страницы можно использовать либо символические имена, либо свойство-массив `frames` объекта `top`, в котором содержатся ссылки на все фреймы страницы. Так, на первый фрейм можно сослаться двумя способами:

```
top.clockFrame
```

или

```
top.frames[0]
```

Свойство location объекта Frame содержит адрес загруженного во фрейм документа. Изменение значения этого свойства приведет к загрузке нового документа в соответствующий фрейм. Следующий фрагмент кода HTML задает на странице menu.html кнопку с именем "Пример 1":

```
<INPUT TYPE="button" VALUE="Пример 1"
       onClick="javascript: top.contentFrame.location='Пример_1.html'">
</FORM>
```

При щелчке на кнопке Пример 1 выполняется код JavaScript, заданный в качестве значения параметра обработки события onClick. Этот код меняет значение свойства location фрейма contentFrame, что приводит к отображению в нем нового документа Пример_1.html.

Если загружаемый во фрейм документ сам содержит набор фреймов, то объектная модель немного усложняется. В этом случае все фреймы, отображаемого в заданном фрейме документа, являются подчиненными этому фрейму, который, в свою очередь, порождается от объекта верхнего уровня top. Если, например, в предыдущем примере файл menu.html будет содержать следующий код HTML:

```
<FRAMESET ROWS="30%,70%">
      <FRAME SRC="file1.html" NAME="Frame1">
      <FRAME SRC="file2.html" NAME="Frame2">
</FRAMESET>
```

то фреймы Frame1 и Frame2 будут порождаться фреймом menuFrame, и для ссылки на один из этих фреймов можно воспользоваться любой из следующих конструкций:

```
top.menuFrame.Frame1
top.frames[2].frames[0]
```

В последней конструкции используется свойство-массив frames любого объекта Frame.

Объект *document*

Этот объект содержит информацию о текущей загруженной странице. Все элементы страницы HTML представляются свойствами объекта document (см. рис. 9.6). Для каждой страницы создается один объект document, некоторые свойства которого соответствуют параметрам тэга-контейнера <BODY>: bgColor, fgColor, linkColor, alinkColor и vlinkColor. При работе с этим объектом полезно его свойство URL, содержащее адрес загруженного документа.

Этот объект используется наиболее часто в связи с его двумя полезными методами write() и writeln(), которые записывают в документ информацию и, тем самым, позволяют динамически его создавать.

Объекты *location* и *history*

Объект `location` связан с текущим URL-адресом. Его свойства позволяют получить информацию о хост-машине, с которой в данный момент связан браузер. Например, свойство `hostname` содержит имя хоста, свойство `port` — номер порта, к которому подсоединен браузер на хост-машине.

Два метода объекта `location` связаны с загрузкой HTML-страниц. Метод `reload()` перезагружает в браузер текущую страницу, а метод `replace()` загружает в окно браузера страницу, адрес которой задан в качестве его параметра.

Объект `history` содержит список адресов HTML-документов, ранее загружавшихся в браузер. Получить адреса текущей, следующей и предыдущей страницы можно с помощью соответственно свойств `current`, `next` и `previous` этого объекта.

Метод `go()` этого объекта загружает страницу из списка посещенных. Текущая страница имеет индекс 0, предыдущие по отношению к текущей страницы индексируются отрицательными целыми числами, а последующие — положительными целыми числами. Например, следующий оператор

```
history.go(-3)
```

загрузит страницу, расположенную на три пункта назад по отношению к текущей в списке посещенных страниц.

Объекты *Form*

Каждая форма в документе, определенная тэгом `<FORM>`, создает объект `Form`, порождаемый объектом `document`. Ссылка на этот объект осуществляется с помощью переменной, определенной в параметре `NAME` тэга `<FORM>`. В документе может быть несколько форм, поэтому для удобства ссылок и обработки в объект `Form` введено свойство-массив `forms`, в котором содержатся ссылки на все формы документа. Ссылка на первую форму задается как `document.forms[0]`, на вторую — `document.forms[1]` и т. д. Вместо индекса в массиве `forms` можно указывать строку, значение которой равно имени переменной для формы. Например, если создана одна форма

```
<FORM NAME="form1">
Информация пользователя:
    <INPUT TYPE="text" NAME="text1" value="Введите что-нибудь"><HR>
    <INPUT TYPE="submit" NAME="subButton">
</FORM>
```

то любой из следующих операторов JavaScript содержит ссылку на эту форму:

```
document.forms[0]
document.forms["form1"]
document.form1
```

Последний оператор возможен в силу того, что объект `document` порождает объект `Form`, и ссылку на него можно осуществлять по обычным правилам наследования языка JavaScript.

Все элементы формы порождают соответствующие объекты, подчиненные объекту родительской формы (см. рис. 9.6). Таким образом, для ссылки на объект `Text` формы `form1` следует пользоваться любым из нижеприведенных операторов:

```
document.forms[0].text1
document.forms["form1"].text1
document.form1.text1
```

Каждый объект `Form` имеет также свойство-массив `elements`, содержащий ссылки на все подчиненные форме элементы в том порядке, как они определены в документе HTML.

Элементы формы, точнее их объекты, имеют свойство `name`, значение которого равно значению параметра `NAME` тэга `<INPUT>`, а также свойство `value`, значение которого определяется смыслом параметра `VALUE` элемента формы. Например, для элементов `text` и `textarea` значением этого свойства будет строка содержимого полей ввода этих элементов, для кнопки подтверждения — надпись на кнопке и т. д.

Свойства-массивы объектов

Некоторые объекты имеют свойства, которые являются массивами. Они используются для хранения информации о подчиненных объектах, когда их количество заранее не известно (например, массив `forms` объекта `document` или массив `elements` объекта `Form`). В табл. 9.8 перечислены все объекты, обладающие свойствами-массивами.

Таблица 9.8. *Свойства-массивы объектов JavaScript*

Объект	Свойство	Описание
`document`	`anchors`	Отражает тэги `<A>` в порядке их появления в документе
	`applets`	Отражает тэги `<APPLET>` в порядке их появления в документе
	`embeds`	Отражает тэги `<EMBED>` в порядке их появления в документе
	`forms`	Отражает тэги `<FORM>` в порядке их появления в документе
	`images`	Отражает тэги `<IMG>` в порядке их появления в документе

Таблица 9.8 *(окончание)*

Объект	Свойство	Описание
document	links	Отражает тэги <AREA HREF="..."> и <A HREF="">, а также объекты link, созданные методом link() в порядке их появления в документе
function	arguments	Отражает параметры функции
form	elements	Отражает элементы формы в порядке их перечисления в тэге <FORM>
select	options	Отражает опции объекта select (тэг <OPTION>) в порядке их появления
window	frames	Отражает тэги <FRAME> в окне, содержащем тэг <FRAMESET>, в порядке их появления в документе
	History	Отражает элементы объекта history
Navigator	MimeTypes	Отражает все типы MIME, поддерживаемые браузером
	Plugins	Отражает все установленные дополнительные приложения для браузера

Работа с перечисленными массивами аналогична работе с массивом forms, описанным выше в разделе *"Объекты Form"*.

Обработчики событий

Интерактивные страницы должны реагировать на действия пользователя. Например, при нажатии на кнопку появляется диалоговое окно с сообщением, или выполняется проверка правильности введенных пользователем данных в полях формы.

В JavaScript подобная интерактивность реализована возможностью перехвата и обработки событий, возникающих в результате действий пользователя. Для этого в тэги некоторых элементов (объектов с точки зрения JavaScript) введены *параметры обработки событий*, задающие действия, выполняемые при возникновении события, связанного с элементом. Имя параметра обработки события начинается с приставки on, за которой следует название события. Если событием является, например, щелчок кнопкой мыши Click, то соответствующий параметр обработки этого события называется onClick; если обрабатываемым событием является нажатие кнопки мыши MouseDown, то параметр называется onMouseDown.

В табл. 9.9 представлены возможные события, и в каких элементах документа HTML они могут инициализироваться.

Таблица 9.9. *События JavaScript*

Событие	Применяется к объектам	Когда происходит событие	Обработчик события
Abort	image	Пользователь отказывается от загрузки изображения	onAbort
Blur	window и все объекты формы	Потеря объектом фокуса	onBlur
Change	text, textarea, select	Пользователь изменяет значение элемента	onChange
Click	button, radio, checkbox, submit, reset, link	Щелчок на элементе формы или связи	onClick
DragDrop	window	Пользователь перетаскивает мышью объект в окно браузера, например файл	onDragDrop
Error	image, window	Загрузка документа или изображения вызывает ошибку	onError
Focus	Window и все объекты формы	Окно или элемент формы получает фокус	onFocus
KeyDown	document, image, link, textarea	Пользователь нажимает клавишу клавиатуры	onKeyDown
KeyPress	document, image, link, textarea	Пользователь удерживает нажатой клавишу клавиатуры	onKeyPress
KeyUp	document, image, link, textarea	Пользователь отпускает клавишу клавиатуры	onKeyUp
Load	Тело документа	Загружается документ в браузер	onLoad
MouseDown	document, button, link	Пользователь нажимает кнопку мыши	onMouseDown
MouseMove	Никакой	Пользователь перемещает курсор мыши	onMouseMove
MouseOut	area, link	Пользователь перемещает курсор из области изображения или со связи	onMouseOut
MouseOver	link	Пользователь перемещает курсор над связью	onMouseOver
MouseUp	document, button, link	Пользователь отпускает кнопку мыши	onMouseUp
Move	window	Пользователь или сценарий перемещает окно	onMove

Таблица 9.9 *(окончание)*

Событие	Применяется к объектам	Когда происходит событие	Обработчик события
`Reset`	`form`	Пользователь нажимает кнопку `Reset` формы	`onReset`
`Resize`	`window`	Пользователь или сценарий изменяет размеры окна	`onResize`
`Select`	`text, textarea`	Пользователь выбирает поле ввода элемента формы	`onSelect`
`Submit`	`form`	Пользователь нажимает кнопку `Submit` формы	`onSubmit`
`Unload`	Тело документа	Пользователь закрывает документ	`onUnload`

Хорошим стилем программирования является оформление действий, выполняемых при обработке событий, в виде процедур.

Процедуры JavaScript

Процедура, или *функция*, — это именованная последовательность операторов, которая инициализируется и выполняется простой ссылкой на имя функции.

Процедура задается оператором `function`, имеющим следующий синтаксис:

```
function имя_функции([параметры]) {
            [операторы]
}
```

где `имя_функции` — любое правильное имя языка JavaScript, `параметры` — список передаваемых в процедуру параметров, элементы которого отделяются запятыми.

Оператор `function` только определяет процедуру, но не выполняет ее. Для вызова процедуры достаточно указать ее имя с заданными в скобках параметрами.

Примечание

Если в процедуре параметры отсутствуют, наличие скобок без параметров в операторе вызова процедуры обязательно.

Процедура может возвращать некоторое вычисляемое в ней значение. В этом случае обычно она называется функцией, и в операторах, определяющих последовательность выполняемых ею действий, обязательно должен присутствовать оператор `return`, задающий возвращаемое функцией значе-

ние. Вызов функции осуществляется аналогично вызову процедуры, но ее можно использовать в выражениях JavaScript.

Обычно все определения процедур и функций задаются в разделе `<HEAD>` документа. Это обеспечивает интерпретацию и сохранение в памяти всех процедур при загрузке документа в браузер.

С процедурами и обработчиками событий тесно связано ключевое слово `this`.

Ключевое слово *this*

Ключевое слово `this` используется для ссылки на текущий объект и обычно применяется для ссылки на объект при вызове процедуры обработки событий в обработчике событий элемента.

Предположим, что для проверки правильности ввода в текстовое поле курса, на котором учится студент, написана процедура `validate`:

```
function validate(obj){
     if( obj.value < 1 || obj.value > 6) {
          alert("Правильно введите Ваш курс")
     }
}
```

Вызов этой процедуры в обработчике событий элемента формы осуществляется следующим образом:

```
<FORM NAME="form1">
<B>Курс, на котором учитесь:</B>
<INPUT TYPE="text" NAME="text1" SIZE=3
       onChange="validate(this.form.text1)>
</FORM>
```

Ключевое слово `this` в данном контексте ссылается на элемент Текстовое поле ввода `text1`. Свойство `form` этого объекта содержит ссылку на текущую форму. Таким образом, получается правильная ссылка на поле ввода в контексте объектной модели документа: `document.form1.text1`.

Объект *event*

В JavaScript каждое событие порождает ассоциированный с ним объект `event`. Этот объект содержит всю информацию о событии и его можно передать процедуре обработки события.

Информация о событии зависит от конкретного произошедшего события. Например, объект `event` события `MouseDown` содержит информацию о типе события (свойство `type`), какая кнопки мыши была нажата (свойство `which`), какая клавиша (`Alt`, `Shift` или `Ctrl`) удерживалась при щелчке кнопкой мыши (свойство `modifiers`), и значения координат курсора мыши в момент возникновения события (свойства `screenX` и `screenY`).

Объекты event совместно с обработчиками событий позволяют проводить достаточно тонкую обработку события.

Вызов процедуры обработки события

Вызов процедуры обработки события можно осуществить двумя способами:

- явно — назначив ссылку на процедуру обработки события в соответствующем свойстве объекта;
- неявно — в параметре обработки события тэга соответствующего элемента.

Каждый объект JavaScript, создаваемый для элементов HTML-документа, имеет свойства, ассоциированные с возможными событиями, которые могут быть сгенерированы для этого элемента. Присвоив этому свойству в качестве значения ссылку на процедуру обработки события, мы, тем самым, определим процедуру, которая будет вызываться при возникновении соответствующего события. Например, следующий код определяет процедуру showEventType как процедуру обработки события MouseDown кнопки формы:

```
<FORM NAME="form1">
<INPUT TYPE="button" NAME="button1" VALUE="Узнай событие">
<SCRIPT>
   document.form1.button.onmousedown = showEventType
</SCRIPT>
</FORM>
```

Сама процедура задается следующим кодом:

```
function showEventType(e) {
  alert("Произошло событие: " + e.type)
}
```

Обратим внимание на то, что, присваивая ссылку на процедуру, мы не задаем в скобках никаких параметров. Это связано с тем, что любая процедура в JavaScript представляется как объект. Поэтому указание имени процедуры (объекта) однозначно определяет ссылку на нее.

В объявлении функции showEventType присутствует параметр e, свойство type которого выводится в диалоговом окне. При явном вызове процедуры обработки события объект event передается ей по умолчанию, поэтому в данном случае печатается значение свойства type объекта event, т. е. тип события — MouseDown.

Второй, неявный вызов процедуры обработки события требует задания обращения к ней в параметре onMouseDown тэга <INPUT>. При этом необходимо явно указывать параметр event, как показано в следующем фрагменте кода:

```
<FORM NAME="form1">
<INPUT TYPE="button" NAME="button1" VALUE="Узнай событие"
```

```
 onMouseDown = "showEventType(event) >
</FORM>
```

Практические примеры

Примеры этого раздела демонстрируют технику рекурсивного обращения к функциям JavaScript, создание простого меню, а также работу с фреймами.

Часы JavaScript

Как отмечалось выше, нельзя изменить содержимое HTML-страницы после ее загрузки в окно браузера. Однако можно динамически с помощью функций JavaScript менять содержимое полей форм, что и используется при отображении текущего времени на странице.

Прежде всего создадим HTML-страницу с текстовым полем, в которое будем выводить текущее время:

```
<HEAD>
</HEAD>
<BODY bgcolor="#FFCC00" onLoad="JSClock()>
<FORM NAME="clockForm">
  <INPUT TYPE="text" NAME="digits" SIZE=8 VALUE="">
</FORM>
</BODY>
```

При загрузке документа вызывается функция `JSClock()`, которая определяет и отображает в поле `digits` формы `clockForm` текущее время. Определение самой функции разместим в разделе `<HEAD>` документа:

```
<SCRIPT LANGUAGE="JavaScript">
<!-- //Скрыть сценарий от браузеров, не поддерживающих JavaScript
function JSClock() {
   var time = new Date()
   var hour = time.getHours()
   var minute = time.getMinutes()
   var second = time.getSeconds()
   var temp = "" + hour
   temp += ((minute < 10) ? ":0" : ":") + minute
   temp += ((second < 10) ? ":0" : ":") + second
   document.clockForm.digits.value = temp
   id = setTimeout("JSClock(),1000)
}
//-->
</SCRIPT>
```

При вызове функции переменная `time` содержит ссылку на объект `Date`, в котором хранится текущее время. Методы `getHours()`, `getMinutes()` и `getSeconds()` объекта `Date` определяют часы, минуты и секунды, задаваемые системными часами. В переменной `temp` формируется строка `"часы:минуты:секунды"` для отображения в поле `digits` формы `clockForm`. Наиболее интересный оператор этой процедуры — оператор вызова функции `setTimeout()`. Эта функция выполняет указанные в первом параметре действия по истечении интервала времени (в миллисекундах), задаваемого вторым параметром. В нашем случае через одну секунду снова вызовется функция `JSClock()`, отобразит в поле формы текущее время, и процесс повторится снова.

Простое меню

На HTML-страницах часто можно встретить меню на фоне привлекательной картинки. При выборе команды такого меню изменяется цветовая гамма ее фона. Подобное меню легко создается с помощью любого языка сценария, в том числе JavaScript.

Прежде всего необходимо выбрать рисунок для фона меню и разрезать его на составные части, представляющие обрамление и команды меню, как показано на рис. 9.9, и сохранить их в отдельных графических файлах.

Добавить названия команд меню в соответствующие графические файлы. Теперь остается только создать новые картинки, которые будут отображаться вместо исходных картинок при расположении курсора мыши над соответствующей командой. Эти картинки должны отличаться только цветом фона. Заготовки графических файлов для одной из команд меню примера показаны на рис. 9.10.

Примечание

Фон графического файла для команды соответствует фону исходной картинки для меню (желтая цветовая гамма); фон графического файла для выбранной команды реализован в синих тонах.

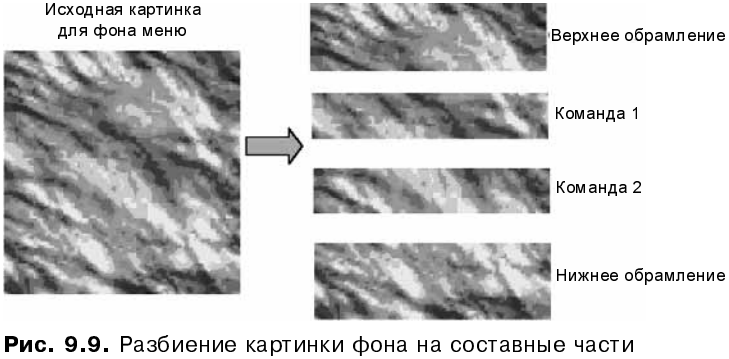


Рис. 9.9. Разбиение картинки фона на составные части

Команда

Выбранная команда

Рис. 9.10. Графические файлы команды и выбранной команды меню

Примечание

Графические картинки подобного типа достаточно просто можно сделать с помощью, например, программы Adobe Photoshop.

В HTML-документе меню создается в виде таблицы, в ячейках которой располагаются графические файлы, причем файлы с командами меню располагаются в тэге `<A>`, содержащем ссылки на загружаемые документы:

```
<body bgcolor="#FFCC00" leftmargin="0" text="#FFFF00" link="#FFFF00"
      vlink="#FFFF00" topmargin="0">
<div align="left">
<table border="0" width="152" cellspacing="0" cellpadding="0">
  <tr>
    <td align="left" width="152"> <img src="M-Up.jpg" border="0"
       WIDTH="152" HEIGHT="50"></td>
  </tr>
  <tr>
    <td align="left" width="152"> <a href="artist.htm"
        onMouseOver="enter('i0') onMouseOut="out('i0')><img name="i0"
        src="m0.gif" border="0" WIDTH="152" HEIGHT="26"></a></td>
  </tr>
  <tr>
    <td align="left" width="152"> <a href="poet.htm"
        onMouseOver="enter('i1') onMouseOut="out('i1')><img name="i1"
        src="m1.gif" border="0" " WIDTH="152" HEIGHT="22"></a></td>
  </tr>
  <tr>
    <td align="left" width="152"><img src="M-Down.jpg" border="0"
        WIDTH="152" HEIGHT="50"></td>
  </tr>
</table>
</div>
</body>
```

Обработчики событий `onMouseOver` и `onMouseOut` вызывают процедуры, которые загружают либо графический образ невыбранной команды, либо графический образ выбранной команды. Эти и другие необходимые процедуры располагаются в разделе `<HEAD>` документа и представлены ниже:

```
<title>MENU</title>
<script LANGUAGE="JavaScript">
```

```
<!--
var MAX = 1;
var img = new MakeArray(MAX);
function MakeArray(n){
  for( var i=0; i<n; i++){
    this['i'+i]=0;
  }
  this.maxlen=n;
  this.len=0;
  return this;
}
function Images(org, swap) {
 if(document.images){
   this.org = new Image();
   this.org.src = org;
   this.swap = new Image();
   this.swap.src = swap;
 }
}
function Add(name, id){
  img[id] = new Images(name+".gif", name+"_m.gif");
}
function enter(id) {
  if(document.images){
    document.images[id].src = img[id].swap.src;
  }
}
function out(id) {
  if(document.images){
    document.images[id].src = img[id].org.src;
  }
}
Add("m0","i0");
Add("m1","i1");
//  -->
</script>
```

Сначала создается массив `img` (с помощью конструктора `MakeArray()`), к элементам которого можно ссылаться по индексу с именем объекта, который в нем хранится. В нашем случае в этом массиве будут храниться графические образы невыбранной и выбранной команды меню. Конструктор `Images()` как раз и предназначен для создания объекта, хранящего подобные графические объекты. Функция `Add()` добавляет в массив `img` объекты, содержащие по два графических файла для каждой команды меню. Назначение функций `enter()` и `out()` — отобразить соответствующий графический файл

при расположении курсора мыши над командой и при выходе курсора из области команды. Результаты работы меню можно видеть на рис. 9.11.

Обычное меню

Меню с выбранной командой

Рис. 9.11. Меню, созданное в примере

Примечание

При просмотре созданного меню в браузере рисунок фона обычного меню реализован в желто-коричневых тонах, рисунок фона выбранной команды — в синих тонах, названия команд реализованы в белом цвете.

Документ с фреймами

В этом примере мы объединим два предыдущих и создадим полноценную HTML-страницу, в одном из фреймов которой будет отображаться текущее время, в другом располагаться меню, а в третий — загружаться соответствующие документы при выборе команд меню.

Текст основной страницы представлен ниже и не требует пояснений:

```
<HEAD>
</HEAD>
<FRAMESET COLS="152,*" border="0">
   <FRAMESET ROWS="70,*">
      <FRAME SRC="Часы.html" NAME="clockFrame">
      <FRAME SRC="menu.html" NAME="menuFrame">
   </FRAMESET>
   <FRAME SRC="" NAME="contentFrame">
</FRAMESET>
```

Каждый фрейм имеет имя, заданное в параметре `NAME`, по которому можно ссылаться на него в тексте JavaScript. Файл `Часы.html` является файлом первого примера данного раздела, файл `menu.html` содержит код документа, реализующего меню с небольшим дополнением: в раздел `<HEAD>` необходимо внести дополнение для указания интерпретатору, что все HTML-страницы, на которые имеются ссылки, должны загружаться во фрейм с именем `contentFrame`. Для этой цели необходимо добавить следующий тэг:

```
<BASE TARGET="contentFrame">
```

Естественно, необходимо создать все HTML-документы, на которые имеется ссылка в меню страницы. Результат отображения страницы показан на рис. 9.12.

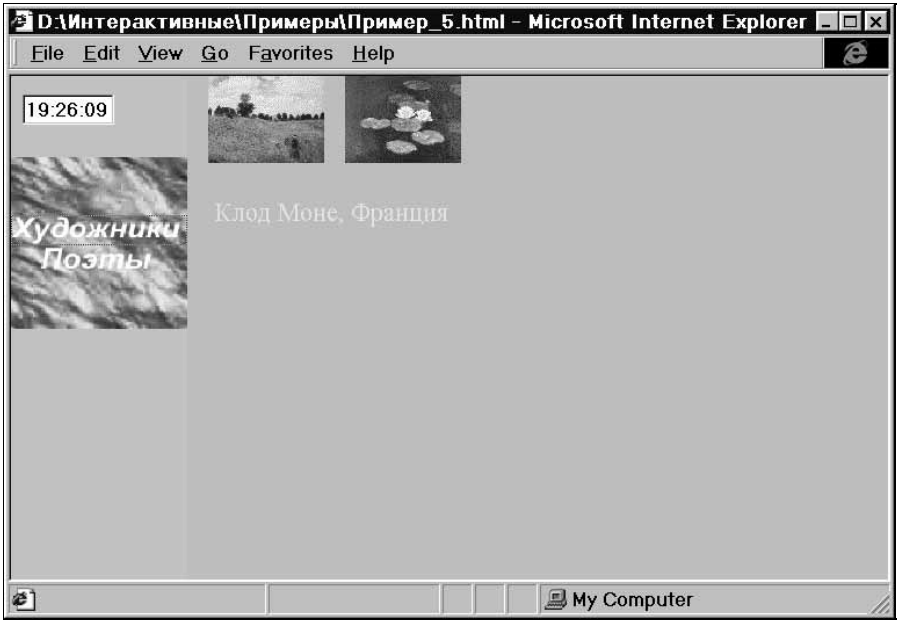


Рис. 9.12. Отображение в браузере страницы примера

Язык VBScript

Язык создания сценариев VBScript разработан фирмой Microsoft и является подмножеством достаточно распространенного в среде программистов языка Visual Basic разработки прикладных Windows-приложений. Как и его родитель, язык VBScript достаточно прост и легок в изучении.

Преимуществом его применения для создания сценариев является возможность использования, с небольшими корректировками, ранее написанных процедур на языках Visual Basic и Visual Basic for Application.

Функциональные возможности сценариев, написанных на VBScript, ничем не отличаются от возможностей сценариев JavaScript: динамическое создание документа или его частей, перехват и обработка событий и т. д.

VBScript используется для написания сценариев клиента (в этом случае браузер должен иметь встроенный интерпретатор этого языка), а также для написания сценариев на сервере (в этом случае сервер должен поддерживать язык VBScript). Для создания сценариев клиента используется набор объек-

тов, аналогичный набору объектов JavaScript *(см. выше раздел "Язык создания сценариев JavaScript")*. Объекты клиента и сервера отличаются друг от друга, но существует общая часть (ядро) объектов, используемых при разработке как сценариев клиента, так и сценариев сервера.

Основные понятия

В этом разделе вводятся основные понятия языка VBScript, необходимые для написания кода сценариев.

Типы данных

В VBScript определен только один тип данных — `Variant`. Это универсальный тип, в котором можно хранить информацию, представленную другими типами данных, применяемыми в программировании, начиная от простейшего целого и заканчивая объектами.

В своем простейшем использовании тип `Variant` содержит либо числовые данные, либо символьные строки — типы данных, наиболее часто встречаемые при написании сценария. Реально содержащиеся в вариантном типе данные могут быть одного из типов, называемых *подтипами* типа `Variant`, представленных в табл. 9.10. Собственно говоря, эти подтипы охватывают все возможные типы данных, которые используются в языках Visual Basic и Visual Basic for Application.

Таблица 9.10. *Подтипы данных, хранящихся в типе* `Variant`

Подтип	Описание	Функция преобразования
`Empty`	Переменная не инициализирована	
`Null`	Переменная не содержит никаких допустимых данных	
`Error`	Содержит номер ошибки	
`Boolean`	Содержит значения либо `True`, либо `False`	`CBool`
`Byte`	Содержит целые числа в диапазоне от 0 до 255	`CByte`
`Integer`	Содержит целые числа в диапазоне от –32 768 до 32 767	`CInt`
`Currency`	Значения в диапазоне от –922 337 203 685 477.5808 до 922 337 203 685 477.5807	`CCur`
`Long`	Содержит целые числа в диапазоне от –2 147 483 648 до 2 147 483 647	`CLng`

Таблица 9.10 *(окончание)*

Подтип	Описание	Функция преобразования
`Single`	Содержит вещественные числа с плавающей точкой одинарной точности в диапазоне от –3.402823E38 до –1.401298E-45 для отрицательных значений и от 1.401298E-45 до 3.402823E38 для положительных значений	`CSng`
`Double`	Содержит вещественные числа с плавающей точкой удвоенной точности в диапазоне от –1.79769313486232E308 до –4.94065645841247E-324 для отрицательных значений и от 4.94065645841247E-324 до 1.79769313486232E308 для положительных значений	`CDbl`
`Date(Time)`	Содержит число, которое представляет дату в диапазоне от 1 января 100 года до 31 декабря 9999 года	`CDate`
`String`	Содержит строку переменной длины (до 2 миллионов символов)	`CStr`
`Object`	Содержит ссылку на объект	

Первые три подтипа, собственно говоря, не являются подтипами, а представляют значения, которые может принимать вариантный тип.

Значение `Empty` имеет переменная, которая была объявлена в операторе `Dim` (см. ниже), но ей еще не присваивали никакого значения. Это значение считается равным 0 в математических операциях и равным пустой строке (`""`) в операциях со строковыми значениями.

Значение `Null` означает, что переменная не содержит данных. Его не следует путать со значением `Empty`. Вариантная переменная может получить значение `Null` в результате выполнения некоторых операций над ней. Это значение можно присвоить переменной, тогда как значение `Empty` — нельзя.

Значение `Error` — это специальное значение, которое используется для указания возникновения ошибки в процедуре.

Каждый подтип данных задается с помощью литералов (символьных констант). Числовые литералы представляют собой целые числа, действительные числа с плавающей или фиксированной точкой. Примеры числовых литералов приведены ниже:

```
 23          ' Целое число
-23.78       ' Действительное число с фиксированной точкой
-237.8E-1    ' Действительное число с плавающей точкой
```

Строковые литералы задаются в виде последовательности символов, заключенных в двойные кавычки ("):

```
"Это строковый литерал".
```

Литералы даты и времени заключаются между символами числовых знаков (#). VBScript поддерживает большое число форматов даты и времени. Следующие примеры показывают правильные литералы даты и времени, соответствующие дате 10 июня 1999 года:

```
#10-6-99 22:20#
#10/6/99#
#10/6/99 10:20pm#
```

Внутренне литералы даты и времени представляются в виде действительных чисел удвоенной точности. Целая часть представляет количество дней, прошедших от даты 30 декабря 1899 года, а дробная часть — время суток.

Булевы литералы `True` и `False` являются константами целого типа, принимающими соответственно значения `1` и `0`. Любое числовое значение, не равное нулю, преобразуется функцией `CBool` в `True`, а нулевое значение (целое или действительное) — в `False`.

Вариантный тип данных при использовании в выражениях в качестве операндов разнообразных операторов языка обрабатывается в зависимости от подтипа содержащихся в нем данных. Например, при использовании переменных этого типа данных в операторе сложения (+) результат зависит от того, какие подтипы данных в них содержатся. Если хотя бы один из операндов содержит число, то результатом будет сумма значений двух переменных (содержимое второго операнда преобразуется к числовому подтипу), если оба операнда содержат строковые данные, то результатом будет конкатенация строк.

Вариантный тип данных предоставляет программисту более эффективный способ обработки и хранения данных, не заботясь о типе хранимых данных. Если, например, при вычислениях первоначально в переменной вариантного типа хранилось значение типа `Byte` (число в диапазоне от 0 до 255), и в результате выполнения некоторых действий это значение стало отрицательным, то просто изменится представление этого числа в переменной (оно станет типа `Integer`) и не возникнет никакой ошибки. Правда, за это удобство приходится платить используемой памятью: для вариантного типа данных вне зависимости от хранимого подтипа нужно 16 байт памяти.

Иногда в некоторых вычислениях необходимо явно преобразовать содержащийся в переменной подтип в другой. Для этого в VBScript имеется ряд функций преобразования в соответствующие типы. В табл. 9.10 последний столбец содержит имена функций преобразования в соответствующий под-

тип. Эти функции в качестве параметра принимают литералы, переменные и выражения.

Переменные, массивы и константы

Переменные используются для хранения данных приложения. Прежде чем переменную можно будет использовать, ее необходимо объявить. Это можно осуществить явным способом с помощью оператора `Dim`, или неявным — просто использовать имя переменной в операторе присваивания. Синтаксис оператора явного объявления переменной следующий:

```
Dim имя_переменной
```

Параметр `имя_переменной` — имя объявляемой переменной. Оно должно начинаться с буквы, не содержать пробелов, точку (`.`), восклицательный знак (`!`), а также символов (`@`), (`&`), (`$`), (`#`) и не превышать длину в 255 символов.

Язык VBScript не чувствителен к регистру. Это означает, что в нем не различаются строчные и прописные буквы. Поэтому, например, и `m`, и `M` будут ссылаться на одну и ту же переменную, если используются в качестве идентификатора переменной.

В одном операторе `Dim` можно объявлять несколько переменных, которые в списке параметров задаются через запятую.

Так как в VBScript определен только один тип данных `Variant`, то и все, используемые в приложении переменные, также имеют тип `Variant`.

Примечание

Вместо оператора `Dim` можно использовать операторы `Private` и `Public` для объявления переменной. Эти операторы, наследуемые VBScript от Visual Basic и задающие там локальные и открытые переменные соответственно, в VBScript выполняют функции, аналогичные оператору `Dim`.

Иногда возникает необходимость хранить данные в массивах, обращаясь к ним с помощью индекса. VBScript поддерживает одномерные и многомерные массивы двух типов:

- Статические массивы
- Динамические массивы

Любой тип массива определяется оператором `Dim`. Отличие от объявления переменной заключается в том, что после имени массива в круглых скобках указывается его размерность. Синтаксис задания массива следующий:

```
Dim имя_массива([индексы])
```

Параметром `индексы` задается число размерностей массива и протяженность каждой размерности. Если этот параметр представляет одно целое число, то он задает одномерный массив, содержащий число элементов на единицу

больше значения параметра. Это связано с тем, что первый элемент массива имеет индекс нуль. Например, следующий оператор

```
Dim M(9)
```

задает массив, состоящий из 10 элементов: первый элемент — `M(0)`, второй — `M(1)` и т. д.

Если параметр `индексы` задан в виде последовательности целых чисел, разделенных запятыми, то оператор `Dim` задает многомерный массив. Количество чисел в списке определяет число размерностей массива, а значение каждого индекса равно количеству элементов массива, соответствующих этой размерности, минус единица. Следующий оператор задает двумерный массив, или таблицу из 10 строк и 4 столбцов:

```
Dim B(9,3)
```

Для получения значения элемента массива следует указать имя массива и индексы элемента. Например, чтобы получить значение элемента, находящегося на пересечении второй строки и третьего столбца массива `B`, следует использовать запись

```
B(1,2)
```

Число размерностей можно задавать до 60, хотя следует учитывать, что с увеличением числа размерностей катастрофически увеличивается требуемое количество памяти для размещения массива.

Задание параметра `индексы` определяет так называемые статические массивы, т. е. массивы, количество элементов и размерности которого известны и не могут быть изменены в процессе выполнения приложения.

Если при задании массива отсутствует параметр `индексы`, то такая конструкция определяет динамический массив, размерность и протяженность каждой размерности которого могут меняться в процессе выполнения программы. Для задания числа размерностей и их протяженности применяется оператор `ReDim`, синтаксис которого аналогичен синтаксису оператора `Dim` при задании статических массивов:

```
Dim Array()
   . . .
ReDim Array(3)
   . . .
```

В этом фрагменте оператор `Dim` задает динамический массив `Array`, а в операторе `ReDim` назначается его размерность.

В программе можно неограниченное число раз изменять размерности динамических массивов. Следует только учитывать, что при новом переопределении размерности массива, его предыдущие элементы могут быть потеряны.

Во избежание потери старых значений элементов динамического массива при новом его распределении, следует использовать в операторе `ReDim` ключевое слово `Preserve`, как показано в следующем примере:

```
Dim Array()
   . . .
ReDim Array(3)
   . . .
ReDim Preserve Array(Ubound(Array)+1)
   . . .
```

В этом фрагменте используется функция `Ubound` для определения верхней границы изменения индекса массива `Array`, затем его протяженность увеличивается на единицу с сохранением предыдущих значений элементов.

Примечание

Для многомерных массивов можно изменять протяженность только последней размерности массива. Любая попытка изменения протяженности не последней размерности приведет к ошибке интерпретатора.

Иногда в программе необходимо задавать переменные, значения которых нельзя изменять. Такие переменные называются *именованными константами*. В VBScript для задания констант существует оператор `Const`, имеющий следующий синтаксис:

```
Const имя_константы = значение
```

Параметр `значение` представляет собой любой допустимый литерал. Например, следующий фрагмент кода определяет несколько констант разных подтипов:

```
Const conName = "Александр"  ' Строковая константа
Const conPi = 3.1416         ' Числовая константа
Const conBirthDay = #1-8-53# ' Константа даты
```

Совет

Чтобы отличать в программе константы от переменных, следует выбрать схему именования констант и придерживаться ее на протяжении разработки всех сценариев. Например, можно использовать для всех констант префикс `con`.

Операторы

При вычислении выражений необходимо производить разнообразные действия с переменными и литералами. Для этих целей в VBScript предусмотрен ряд встроенных операторов, выполняющих арифметические операции, операции сравнения, конкатенацию (соединение) строк и логические операции над данными, хранящимися в переменных, или представленными литералами.

В VBScript каждый оператор размещается на отдельной строке и не завершается никаким разделителем. Однако, если возникает необходимость задания нескольких операторов в одной строке, то они разделяются двоеточием (:).

Если оператор достаточно длинный, или из соображений удобства чтения исходного текста необходимо расположить его в нескольких строках, то следует использовать символы продолжения — пробел со знаком подчеркивания (_).

Комментарии в языке VBScript вводятся в текст программы одинарной кавычкой (').Любой текст, расположенный в строке за одинарной кавычкой, трактуется интерпретатором как комментарий, и, естественно, не обрабатывается им.

Большую группу представляют арифметические операторы, выполняющие основные арифметические действия над числовыми данными: возведение в степень (^), умножение (*), деление (/), целочисленное деление (\), сложение (+) и вычитание (-). Они подчиняются принятым в математике правилам старшинства операций: сначала выполняется возведение в степень, затем умножение или деление, далее сложение или вычитание. Скобки изменяют последовательность вычисления операций.

Примечание

Так как в VBScript существует только один тип данных `Variant`, то интерпретатор языка преобразует хранящиеся в вариантных переменных не числовые подтипы данных в числовые, и производит указанные в выражении арифметические операции. Если подтип данных не может быть конвертирован в правильное число, то генерируется ошибка.

Предупреждение

Оператор сложения (+), примененный к операндам, содержащим строковые данные, выполняет операцию конкатенации строк, а не сложение преобразованного в числовые данные содержимого обоих операндов. Для выполнения операции конкатенации в VBScript существует специальный оператор (&), которым и следует пользоваться для соединения строковых данных.

Для сравнения данных используются операторы сравнения: равенство (=), неравенство (<>), меньше (<), больше (>), меньше или равно (<=), больше или равно (>=). Объекты сравниваются с помощью специального оператора `Is`.

Примечание

В VBScript нет специального знака для операции присваивания. Один и тот же знак равенства используется как для операции присваивания значения переменной (см. выше), так и для операции сравнения на равенство. Смысл операции, представляемой этим символом, зависит от контекста, в котором она применена.

Существует ряд операторов, выполняющих действия над логическими (булевыми) данными: отрицание (`Not`), конъюнкция (`And`), дизъюнкция (`Or`), исключающее ИЛИ (`Xor`), эквивалентность (`Eqv`) и импликация (`Imp`). Эти операции используются в операторах условия для вычисления выражений, истинность (или ложность) которых позволяет изменить поток выполнения операторов языка.

Оператор (`&`) производит конкатенацию (соединение) двух строк. При его выполнении данные, содержащиеся в операндах, преобразуются при необходимости к строковому подтипу, и осуществляется сцепление двух строк.

Операторы условия и цикла

Операторы сценария выполняются последовательно в том порядке, как они записаны. Изменить порядок выполнения операторов в VBScript можно операторами условия и цикла.

Операторы принятия решения (условные операторы) выполняют определенные блоки операторов в зависимости от результатов проверки некоторого выражения или выражений. VBScript поддерживает следующие конструкции операторов принятия решения:

- `If...Then`
- `If...Then...Else`
- `Select Case`

Конструкция `If...Then` применяется, когда необходимо выполнить группу операторов или один оператор в зависимости от значения выражения, задаваемого в качестве параметра условия конструкции.

Ее первая форма

```
If условие Then оператор
```

позволяет вычислить указанный `оператор`, если истинно заданное `условие`.

Вторая форма этой конструкции позволяет вычислить группу операторов, заданных в нескольких строках кода, и имеет следующий синтаксис:

```
If условие Then
       операторы
End If
```

Примечание

Во всех условных операторах проверяемое на истинность `условие` не обязательно должно быть сравнением. Оно может быть любым вычисляемым выражением, которое интерпретируется как `Ложь`, в случае нулевого значения, и как `Истина` — в противном случае.

Наиболее общий синтаксис конструкции `If...Then...Else` следующий:

```
If условие1 Then
       [группа-операторов-1]
[ElseIf условие2 Then
       [группа-операторов-2]]...
[Else
       [группа-операторов-n]]
End If
```

Сначала проверяется `условие1`. Если оно ложно, то проверяется `условие2`. Если и оно ложно, то проверяется следующее условие из группы `ElseIf` до тех пор, пока не будет найдено истинное условие, операторы которого и выполняются. После чего управление передается оператору, непосредственно следующему за оператором `End If`.

Если не найдено ни одно истинное условие, то выполняется группа операторов из блока `Else`, если он присутствует в конструкции. В противном случае управление передается оператору, следующему за оператором `End If`.

Примечание

Блоков `ElseIf` в конструкции `If...Then...Else` может быть сколько угодно, тогда как блок `Else` всегда один, если он задан.

Если в предыдущей конструкции принятия решения проверяется равенство *одного* выражения *разным* условиям, она становится не достаточно эффективной как с точки зрения ее выполнения, так и с точки зрения легкости восприятия текста. В этом случае следует использовать конструкцию `Select Case`:

```
Select Case тестируемое_выражение
      [Case список_значений1
          [группа-операторов-1]]
      [Case список_значений2
          [группа-операторов-1]]
          .
          .
          .
      [Case Else
          [группа-операторов-n]]
End Select
```

Вычисляется единственное выражение `тестируемое_выражение` и последовательно сравнивается со значениями из списков значений блоков `Case`. Если значение выражения совпадает со значением, заданным в списке какого-либо блока `Case`, то выполняется группа операторов данного блока, и после

этого управление передается оператору, непосредственно следующему за оператором `End Select`.

Если не найдено ни одного соответствия значения тестируемого выражения со значениями из списков значений, то выполняется группа операторов блока `Case Else` (в случае его наличия).

Список значений блока `Case` может состоять из одного или нескольких значений. В случае нескольких значений они разделяются запятыми.

Примечание

Если вычисленное значение тестируемого выражения совпадает со значениями из нескольких блоков `Case`, то выполняется группа операторов, заданная в первом из блоков `Case`, в котором найдено соответствие.

Для повторного выполнения несколько раз группы операторов VBScript, как и любой другой язык программирования, предоставляет разнообразные типы операторов цикла:

- `Do...Loop`
- `For...Next`
- `For Each...Next`

Конструкция `Do...Loop` применяется для выполнения группы операторов, пока некоторое выражение ложно или истинно. Она имеет несколько разновидностей, отличающихся моментом проверки условия завершения цикла (до начала выполнения группы операторов или после) и тем, истинно или ложно это условие.

Цикл `Do While` выполняется до тех пор, пока истинно условие окончания цикла:

```
Do While условие_окончания
      группа-операторов
Loop
```

Перед выполнением операторов цикла проверяется, истинно ли выражение `условие_окончания`. Если оно истинно, то выполняется `группа-операторов` (в ней изменяются значения переменных, входящих в выражение `условие_окончания`). После этого снова проверяется условие окончания цикла и, в случае его истинности, выполняется группа операторов тела цикла. Процедура повторяется до тех пор, пока выражение `условие_окончания` не станет ложным.

Примечание

Если выражение, определяющее условие завершения цикла и первоначально имеющее значение `True`, не изменяется в теле цикла, то цикл будет повторять-

ся бесконечное число раз. Для выхода из бесконечного цикла следует использовать оператор `Exit Do` (см. ниже).

Цикл `Do While` не будет выполнен ни разу, если при первой проверке `условие_окончания` ложно.

Другая разновидность цикла `Do While` сначала выполняет группу операторов, а потом проверяет условие окончания цикла:

```
Do
      группа-операторов
Loop While условие_окончания
```

Этот цикл обязательно выполнит один раз группу операторов, определенных в теле цикла.

Цикл `Do Until` аналогичен первой разновидности цикла `Do While`, за исключением того, что он выполняется, пока значение выражения `условие_окончания` ложно:

```
Do Until условие_окончания
      группа-операторов
Loop
```

Этот цикл также может не выполниться ни одного раза, если при первой же проверке условия завершения цикла, оно оказывается истинным.

Во второй разновидности цикла `Do Until` условие окончания завершения цикла проверяется после выполнения группы операторов, и, таким образом, он обязательно выполнится хотя бы один раз:

```
Do
      группа-операторов
Loop Until условие_окончания
```

Примечание

Выражение `условие_окончания` не обязательно должно быть выражением сравнения, принимающим значение `Истина` или `Ложь`. Значение любого вычисляемого выражения трактуется как `Ложь`, если оно равно нулю, и `Истина` — в противном случае.

В циклах `Do...Loop` заранее не известно количество итераций повторения группы операторов, но иногда требуется выполнить точно заданное число повторений цикла. Такую возможность предоставляет цикл `For...Next`.

В этом цикле задается переменная, называемая счетчиком цикла, которая увеличивается (или уменьшается) на заданную величину после выполнения группы операторов. Цикл завершает свои итерации, когда значение счетчика превысит (или станет меньше) заданной величины. Синтаксис такой конструкции цикла следующий:

```
For счетчик = нач_значение To кон_значение [Step приращение]
        операторы
Next
```

В начале выполнения этого цикла переменной `счетчик` присваивается значение, заданное параметром `нач_значение`. Выполняются операторы цикла, и значение переменной цикла увеличивается или уменьшается (в зависимости от знака) на величину `приращение`. Осуществляется проверка, не превысило ли (или не стало меньше) новое значение счетчика значение параметра `кон_значение`. Если нет, то итерации повторяются, если да, то цикл завершает свое выполнение.

Параметр `приращение` цикла `For...Next` является необязательным. Если он не задан, то по умолчанию переменная цикла увеличивается на 1.

Примечание

Следует аккуратно задавать все три параметра цикла `For...Next`. Они должны быть согласованы: если приращение положительно, то начальное значение должно быть меньше или равно конечному значению; если приращение отрицательно, то конечное значение должно быть меньше или равно начальному значению. Если это не так, то цикл `For...Next` не выполняется ни одного раза.

Конструкция `For Each...Next` позволяет организовать цикл по элементам массива или по объектам некоторого набора (семейства) объектов (*см. ниже в разделе*), не зная заранее число элементов в массиве или число объектов в наборе. Синтаксис этой конструкции следующий:

```
For Each элемент In группа
        операторы
Next
```

Параметр `группа` задает имя массива или имя набора объектов. Переменная `элемент` на каждом шаге цикла будет содержать ссылку на элемент массива или объект набора. Цикл завершает свое выполнение, как только завершится последовательный перебор всех элементов массива или объектов набора.

Следующий фрагмент кода подсчитывает количество элементов в массиве:

```
Dim Ar(10)
        . . .
Number = 0
For Each j In Ar
     Number = Number + 1
Next
Document.Write "Число элементов массива равно " & Number
```

Результатом работы данного сценария будет строка в документе:

```
Число элементов массива равно 11
```

В следующем примере используется набор `forms` для организации цикла по числу форм в документе. Внутренний цикл по элементам набора `elements` формы печатает в документ значения свойства `value` всех элементов формы.

Пример 9.4. Отображение всех элементов форм документа

```
<BODY>
<FORM name="frm1">
<INPUT TYPE="checkbox" NAME="check1" value="Флажок"> Флажок
</FORM>
<FORM name="frm2">
<INPUT TYPE="checkbox" NAME="check2" value="Флажок1"> Флажок1
</FORM>
<p>
<SCRIPT language="VBScript">
For Each i In document.forms
 document.write "<p> Форма " & i.name & ":"
 For Each j In i.elements
   document.write "<p> Элемент: " & j.value
 Next
 document.write "<HR>"
Next
</SCRIPT>
</BODY>
```

На каждом шаге внешнего цикла переменная `i` содержит ссылку на форму, созданную тэгом `<FORM>`. Во внутреннем цикле осуществляется перебор элементов формы. Ссылка на набор `elements` формы указывается в качестве параметра `i.elements` цикла `For Each`. Переменная `j` внутреннего цикла последовательно ссылается на элементы формы, поэтому конструкция `j.value` дает значение параметра `value` элемента формы. Результат отображения страницы в окне браузера показан на рис. 9.13.

Обычно любой оператор цикла выполняется столько раз, сколько определено его параметрами, но иногда необходимо прервать выполнение цикла, например, в результате выполнения какого-либо условия. Для этих целей в VBScript существует несколько операторов `Exit` безусловного выхода из конструкций цикла:

- `Exit Do` осуществляет безусловный выход из операторов группы `Do...Loop`
- `Exit For` осуществляет безусловный выход из операторов `For...Next` и `For Each...Next`

Их можно использовать в любом месте в теле цикла, но обычно они используются совместно с операторами условиями.

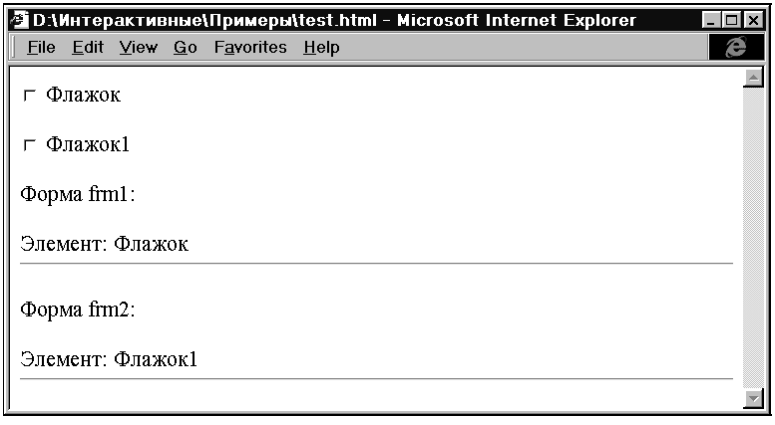


Рис. 9.13. Отображение информации по элементам форм документа

Процедуры

Часто приходится выполнять одну и ту же последовательность операторов, реализующих какое-нибудь действие в программе, возможно с разными переменными или разными значениями. Например, отображение подсказки пользователю в диалоговом окне можно рассматривать как одну и ту же последовательность действий (отобразить на экране диалоговое окно) с разными значениями (содержание подсказки). Такая последовательность действий обычно оформляется в виде процедуры (с параметрами или без параметров), инициализация выполнения которой осуществляется простым указанием имени этой процедуры и передаче ей необходимых параметров.

VBScript предусматривает создание двух типов процедур:

- Процедура `Sub`
- Процедура `Function` (или функция)

Процедура `Sub` выполняет последовательность действий, но не возвращает никакого значения, ассоциированного с ее именем. Она имеет следующий синтаксис:

```
Sub имя_процедуры ([список-параметров])
       операторы
End Sub
```

Через необязательный список параметров можно передать в процедуру внешние данные или, наоборот, получить некоторые вычисленные ею значения.

Вызов процедуры `Sub` осуществляется оператором `Call`, после которого указывается имя процедуры и в круглых скобках параметры. Процедуру можно вызвать и простым указанием ее имени, но в этом случае передаваемые ей

параметры задаются без скобок. Следующие два способа вызова процедуры эквивалентны:

```
Call MyProc(firstarg, secondarg)
MyProc firstarg, secondarg
```

Функция также выполняет определенную последовательность операторов и ей можно передать внешние данные через параметры процедуры, но, в отличие от процедуры `Sub`, она возвращает значение, присваиваемое ее имени, и может быть использована в выражениях VBScript. Она имеет следующий синтаксис:

```
Function имя_процедуры ([список-параметров])
       операторы
       имя_процедуры = значение
End Function
```

В операторах процедуры обязательно должен присутствовать оператор присвоения имени процедуры некоторого значения. Следующая функция пересчитывает переданное ей в качестве параметра значение в дюймы или сантиметры; второй параметр этой процедуры определяет, в каких единицах измерения задано исходное значение параметра.

```
Function InchMeter( value, index)
   Select Case index
          Case 0      ' Задано значение в сантиметрах
              InchMeter = value / 2.54
          Case 1      ' Задано значение в дюймах
              InchMeter = value * 2.54
          Case Else
              InchMeter = -1
              MsgBox "InchMeter: неправильный параметр index"
   End Select
End Function
       . . .
leng = InchMeter(1.5,0)
       . . .
```

В этом фрагменте кода функция `InchMeter()` вызывается в правой части оператора присваивания. Вызов функции в выражениях осуществляется указанием ее имени и списком параметров в круглых скобках. Функции `Function` можно вызывать, используя синтаксис вызова процедур `Sub`, но в этом случае VBScript игнорирует возвращаемое ею значение. Следующие два оператора вызывают одну и ту же процедуру `InchMeter()`, но возвращаемые значения не доступны для дальнейшего использования:

```
Call InchMeter(1,0)
InchMeter 1,1
```

VBScript предоставляет два способа передачи параметров в процедуры:

- По ссылке
- По значению

Способ передачи параметра по ссылке, применяемый по умолчанию, передает фактический адрес переменной, используемой в качестве параметра. Это позволяет изменить содержимое соответствующего адреса памяти, а тем самым, и значение переменной.

Передача параметров по значению предполагает передачу в процедуру копии переменной, а не адреса самой переменной. Поэтому любые изменения параметра внутри процедуры воздействуют на копию, а не на саму переменную, и, следовательно, значение переменной, переданной в качестве параметра, изменяться не будет. Для указания интерпретатору, что параметр передается в процедуру по значению, используется ключевое слово `ByVal`, задаваемое перед параметром в описании процедуры. При вызове следующей процедуры значение переменной `l`, переданной в качестве параметра, не изменяется, хотя процедура изменяет значение параметра:

```
Sub proc(ByVal m)
        m = 3
End Sub
l = 4
proc l
document.write l // Напечатает 4, а не 3
```

Объектная модель и взаимодействие с элементами документа

Как уже говорилось (в разделе *"Объектные модели языков сценариев"* этой главы), при интерпретации HTML-страницы браузер создает объекты, соответствующие элементам отображаемой страницы. Эти объекты доступны для приложения VBScript и любой сценарий может обратиться или установить значения свойств объектов, выполнить их методы.

Язык VBScript, как и другой распространенный язык сценариев, может манипулировать стандартными объектами, встроенными в ядро VBScript, и объектами, предоставляемыми браузером. При описании объектов JavaScript мы рассматривали объекты браузера Netscape Navigator фирмы Netscape. Теперь остановимся на объектах браузера Microsoft Internet Explorer фирмы Microsoft.

Функции и объекты ядра VBScript

Ядро VBScript предоставляет программисту большой набор встроенных функций для работы с датами, строковыми подтипами данных, вычисления ма-

тематических функций и отображения информационных диалоговых окон и окон ввода информации пользователем.

Для вычисления стандартных математических функций используются процедуры-функции, представленные в табл. 9.11. В качестве параметров этих процедур могут выступать числовые литералы, переменные, содержащие числовые значения, или числовые выражения.

Таблица 9.11. *Математические функции VBScript*

Функция	Действие
`Abs`	Вычисляет абсолютное значение
`Atn`	Вычисляет значение арктангенса своего параметра
`Cos`	Вычисляет значение косинуса угла, заданного в радианах
`Exp`	Экспоненциальная функция: e^x
`Int`	Целая часть числа
`Log`	Вычисляет натуральный логарифм своего параметра
`Sin`	Вычисляет значение синуса угла, заданного в радианах
`Sqr`	Вычисляет значение квадратного корня своего параметра
`Tan`	Вычисляет значение тангенса угла, заданного в радианах

Получить текущее время, текущую дату, а также одновременно и дату, и время, установленные на компьютере пользователя, можно с помощью функций `Time`, `Date` и `Now`. Эти три процедуры-функции не требуют параметров, поэтому при их использовании в выражениях следует задавать открывающую и закрывающую скобки, указывающие на отсутствие параметров, как показано в следующем фрагменте кода:

```
iTime = Time()     ' переменная iTime содержит текущее время
iDate = Date()     ' переменная iDate содержит текущую дату
iDateTime = Now()  ' переменная iDateTime содержит текущую дату и время
```

Что хранится в этих переменных, зависит от системных установок времени и даты, определенных на компьютере пользователя. Например, на компьютере с установленным форматом времени Ч:мм:сс и даты дд.ММ.гг печать в документе значений переменных предыдущего фрагмента кода будет выглядеть следующим образом:

```
14:49:31              ' Значение переменной iTime
23.06.99              ' Значение переменной iDate
23.06.99 14:49:31     ' Значение переменной iDateTime
```

Если какая-либо переменная содержит значение подтипа `Date`, или ее содержимое может быть преобразовано в этот подтип, то функции

`Day(переменная)`, `Month(переменная)` и `Year(переменная)` возвращают соответственно число, месяц и год даты, не зависимо от формата хранения дат на компьютере пользователя. Функции `Hour(переменная)`, `Minute(переменная)` и `Second(переменная)` возвращают целые числа, соответствующие часам, минутам и секундам времени, хранящимся в переменной, являющейся параметром этих функций.

Для работы со строками предназначен целый ряд функций, которые полезны при проверке правильности ввода информации в поля формы.

Функция `Len(строка)` возвращает число символов в строке с учетом лидирующих и замыкающих пробелов. Если пользователь ввел в поле формы строку `" строка "` с двумя пробелами в начале и одним пробелом в конце, то эта функция возвратит значение 9. Для удаления незначащих пробелов в начале или конце строки можно обратиться к функциям `Ltrim`, `Rtrim` и `Trim`, которые, соответственно, удаляют лидирующие, замыкающие или оба типа пробелов из строки, переданной этим функциям в качестве параметров.

Функции `LCase` и `UCase` переводят все содержимое строки соответственно в нижний или верхний регистр.

Для сравнения двух строк используется функция `StrComp(строка1, строка2, метод_сравнения)`. Она возвращает значение `0` (если строки совпадают в соответствии с методом сравнения, заданным третьим параметром функции), значение `-1` (если `строка1` меньше `строки2`) и значение `1` в любом не совпадающем с двумя предыдущими случае. Параметр `метод_сравнения` может принимать два значения: `0` — строки сравниваются с учетом регистра, `1` — строки сравниваются без учета регистра.

Функция `InStr(нач_позиция, строка1, строка2, метод_сравнения)` позволяет определить, содержится ли `строка2` в `строке1`. Параметр `нач_позиция` задает позицию в `строке1`, с которой начинается поиск подстроки, заданной параметром `строка2`. Это достаточно полезно, если известно, что `строка1` содержит несколько вхождений `строки2`. Параметр `метод_сравнения` определяет, учитывается ли регистр при поиске соответствующей подстроки, и совпадает с аналогичным параметром функции `StrComp`. Функция поиска подстроки `IntStr` возвращает номер позиции в `строке1` первого вхождения `строки2`, если такое найдено, и значение `0` — в противном случае.

Две стандартные функции `InputBox` и `MsgBox` позволяют отобразить диалоговое окно с полем ввода и диалоговое окно с сообщением пользователю.

Функция `InputBox` отображает диалоговое окно с полем ввода и двумя кнопками: **OK** и **Cancel** (Отмена). Она имеет следующий синтаксис:

```
InputBox(подсказка, заголовок, умалч_знач, x, y)
```

Строковые параметры `подсказка` и `заголовок` задают соответственно подсказку для пользователя и заголовок диалогового окна. Параметр `умалч_знач`

определяет значение, выводимое в поле ввода при отображении окна. Необязательные два последних параметра задают в твипах (1 твип = 1/1440 дюйма = = 1/20 пойнта) координаты левого верхнего угла диалогового окна относительно левого верхнего угла окна браузера.

Эта функция возвращает значение строки, введенной пользователем, или заданное умалчиваемое значение при нажатии кнопки **OK**. При нажатии кнопки **Cancel** возвращаемое функцией значение равно `Empty`. На рис. 9.14 показано диалоговое окно, отображаемое оператором:

```
IC = InputBox("Введите Ваш регистрационный номер.", "Регистрация", "")
```

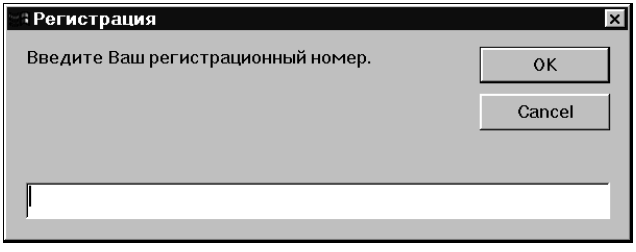


Рис. 9.14. Диалоговое окно, отображаемое функцией `InputBox`

С помощью функции `MsgBox` можно не только просто информировать пользователя о возникших проблемах, но и предоставить ему возможность разрешить ее некоторыми стандартными способами. Например, если не удается из сценария связаться с каким-нибудь сервером, пользователь может отказаться от этого действия, повторить его или отменить. Подобное взаимодействие осуществляется отображением диалоговых окон разных типов, задаваемых значением соответствующего параметра функции `MsgBox`. Синтаксис вызова этой функции следующий:

```
IC=MsgBox(строка_сообщения, тип_окна, заголовок_окна)
```

Параметр `строка_сообщения` задает строку сообщения, отображаемую в диалоговом окне, параметр `заголовок_окна` определяет надпись в заголовке окна, а параметр `тип_окна` задает тип отображаемого окна. Он является целым числом и его значение определяется как сумма трех целочисленных подпараметров, задающих элементы диалогового окна. Эти подпараметры определяют количество и типы кнопок, тип значка и умалчиваемую кнопку, отображаемые в диалоговом окне. Значения этих параметров представлены в табл. 9.12.

Таблица 9.12. *Значения подпараметров вида диалогового окна*

Типы кнопок	
Значение	**Результат**
0	Отображается одна кнопка **OK**
1	Отображаются кнопки **OK** и **Cancel** (Отмена)

Таблица 9.12 *(окончание)*

Типы кнопок	
Значение	**Результат**
2	Отображаются кнопки **Abort** (Стоп), **Retry** (Повтор) и **Ignore** (Пропустить)
3	Отображаются кнопки **Yes** (Да), **No** (Нет) и **Cancel**
4	Отображаются кнопки **Yes** и **No**
5	Отображаются кнопки **Retry** (Повтор) и **Cancel**

Типы значков	
Значение	**Результат**
0	Никакой значок не отображается
16	Отображается значок **Стоп**
32	Отображается значок **Вопрос**
48	Отображается значок **Восклицание**
64	Отображается значок **Информация**

Умалчиваемая кнопка	
Значение	**Результат**
0	Первая кнопка
256	Вторая кнопка
512	Третья кнопка

Примечание

Надписи на кнопках отображаются в зависимости от используемой операционной системы. В таблице в скобках после английского названия кнопок указано их название при работе в русской версии Windows 95.

Если необходимо отобразить диалоговое окно с тремя кнопками **Yes**, **No** и **Cancel** (значение подпараметра равно 3), со значком **Восклицание** (значение подпараметра равно 48) и третьей кнопкой (**Cancel**) по умолчанию (значение подпараметра равно 512), то необходимо при вызове функции `MsgBox` в качестве значения параметра `тип_окна` указать 563 (3+48+512), как показано в следующем примере:

```
MsgBox "Вы действительно хотите завершить процесс?", 563,"Сообщение"
```

Результат выполнения этого оператора можно увидеть на рис. 9.15.

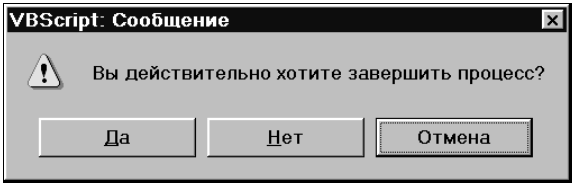


Рис. 9.15. Диалоговое окно, отображаемое функцией `MsgBox`

Функция `MsgBox` возвращает целочисленное значение, по которому можно определить, какая кнопка была нажата, и предпринять необходимые действия. Коды возврата функции показаны в табл. 9.13.

Таблица 9.13. *Коды возврата функции* `MsgBox`

Возвращаемое значение	Нажата кнопка
1	**OK**
2	**Cancel** (Отмена)
3	**Abort** (Стоп)
4	**Retry** (Повтор)
5	**Ignore** (Пропустить)
6	**Yes** (Да)
7	**No** (Нет)

VBScript предоставляет ряд объектов, не связанных с объектной моделью HTML-страницы и предназначенных для сценариев, выполняющихся на сервере. Это объекты для работы с файловой системой: дисками, папками и файлами. Свойства и методы этих объектов позволяют получать информацию о файлах, расположенных на компьютере, на котором выполняется сценарий. Потенциально их можно включать в сценарии HTML-страниц, но вероятность их выполнения мала, так как при попытке использования подобных объектов браузер отображает предупреждающее сообщение (рис. 9.16), и пользователь может отказаться от дальнейшего выполнения сценария.

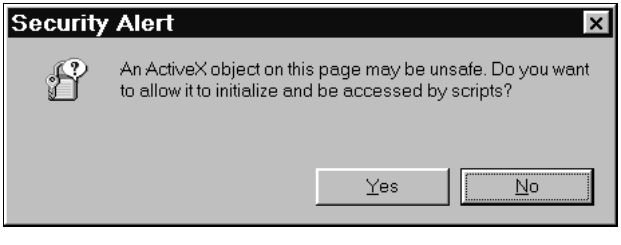


Рис. 9.16. Сообщение браузера при использовании в сценарии объектов доступа к файлам

Здесь дадим краткое описание использования подобных объектов на примере объектов `FileSystemObjects` (Объекты файловой системы), так как с их помощью можно быстро решить какие-то свои домашние задачи, не обращаясь к "серьезным" системам программирования.

Прежде чем использовать методы и свойства любого объекта VBScript, его необходимо создать. Это достигается использованием оператора `Set` совместно с функцией `CreateObject`:

```
Dim fso, MyFile
Set fso = CreateObject("Scripting.FileSystemObject")
```

Сначала описывается переменная `fso`, в которой будет храниться ссылка на объект файловой системы, а затем в операторе `Set` создается сам объект, и в переменную заносится ссылка на него. Теперь можно использовать все методы и свойства созданного объекта, применяя обычную точечную нотацию объектно-ориентированных технологий. Например, в следующем фрагменте кода создается объект `TextStream`, метод `WriteLine` которого записывает строку текста в файл:

```
Set MyFile = fso.CreateTextFile("a:\testfile.txt", True)
MyFile.WriteLine("Проверка записи в файл из VBScript.")
MyFile.Close
```

Если выполнить приведенные два фрагмента кода в одном сценарии, то на дискете будет создан новый файл, содержащий одну текстовую строку.

Примечание

Создание нового объекта всегда осуществляется с помощью оператора `Set`. После знака равенства указывается *конструктор* объекта — функция, возвращающим значением которой является объект. Конструктор может иметь параметры.

Совет

Описание всех объектов VBScript можно найти на сервере фирмы Microsoft по адресу **http://msdn.microsoft.com**. С него же можно установить на свой компьютер документацию по языку сценариев VBScript.

Объекты MS Internet Explorer

Браузер MS Internet Explorer разработан с использованием технологии Automation. Приложение, созданное на основе этой технологии, раскрывает свои объекты другим приложениям, поддерживающим данную технологию. Эти приложения имеют доступ к методам и свойствам объектов другого приложения Automation. Иерархическая структура объектов Internet Explorer достаточна сложна, но для написания сценариев используется часть объектной модели, связанной с элементами HTML-страницы.

Во главе иерархии, как и в случае с Netscape Navigator, стоит объект `window`, представляющий окно браузера и порождающий все остальные объекты модели. При ссылке в программе на любой объект из иерархии можно не указывать "родительский" объект `window`.

Модель охватывает практически все элементы HTML-страницы. На рис. 9.17 показана иерархическая структура объектной модели, которая отражает подчиненность элементов страницы.

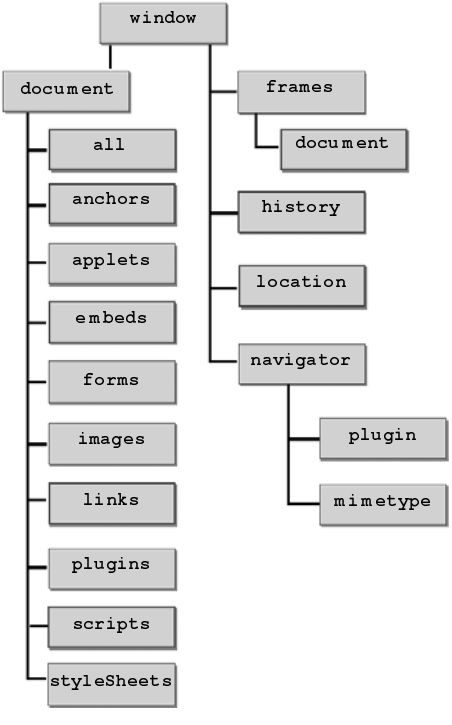


Рис. 9.17. Объектная модель MS Internet Explorer

Для каждого типа элементов в модели предусмотрены соответствующие наборы. Например, для объектов `image`, определяемых тэгами `<IMG>`, существует набор `images`. Ссылку на соответствующий объект можно определить с использованием имени объекта, задаваемого значением параметра `NAME`, или с помощью набора объектов, в данном случае `images`. В наборе объекты расположены в последовательности, в которой они задаются на HTML-странице.

Примечание

Наборы объектов, собственно говоря, являются свойствами порождающих их объектов. Имена наборов объектов задаются в виде множественного числа существительных (в английском языке добавляется буква "s" в конце слова), определяющих типы объектов.

Формы HTML, отображаемые в объектах `Form`, содержат разнообразные элементы управления, которые порождают соответствующие объекты, подчиненные объекту `Form`. Все, что сказано об этих объектах в разделе *"Язык создания сценариев JavaScript"*, остается в силе и при работе с этими объектами с помощью языка VBScript.

Процедуры обработки событий

Как и в сценарии JavaScript, весь код VBScript должен располагаться в тэге-контейнере `<SCRIPT>...</SCRIPT>`. Для обратной совместимости с браузерами, не поддерживающими язык сценариев VBScript, следует размещать код внутри тэга `<SCRIPT>` в контейнере-комментарии, как показано ниже:

```
<SCRIPT TYPE="text/vbscript" LANGUAGE="VBScript">
<!--
       код сценария VBScript
'-->
</SCRIPT>
```

Завершающий тэг комментария предваряется знаком (`'`), чтобы интерпретатор браузера не сгенерировал ошибку синтаксиса.

Предупреждение

Следует избегать появления в коде сценария последовательности символов (`</`), так как интерпретатор воспринимает ее как завершающий тэг и может не обработать до конца сценарий. Если необходимо напечатать в документе последовательность указанных символов, в коде VBScript следует заменить знак `/` на его представление через десятичный код `Chr(47)` и использовать операцию конкатенации строк. Например, если необходимо при динамическом создании документа напечатать `<p>Абзац</p>`, то следует использовать метод `Write` объекта `Document` следующим образом:

```
Document.Write "<p>Абзац<" & Chr(47) & "p>"
```

Параметр `LANGUAGE` задает язык сценария. Параметр `TYPE`, определяющий MIME-тип заключенного в тэг `<SCRIPT>` кода, задает также язык сценария. Для VBScript значение этого параметра должно быть строкой `"text/vbscript"`. Этот параметр включен в HTML 4.0, но может не поддерживаться некоторыми браузерами, поэтому рекомендуется задавать оба параметра одновременно.

Примечание

Все сказанное выше о параметрах `TYPE` и `LANGUAGE` относится и к языку JavaScript, для которого MIME-тип следует задавать в виде строки `"text/javascript"`.

Указанным выше способом код сценария непосредственно встраивается в страницу. Параметр `SRC` тэга `<SCRIPT>` позволяет указать полный или отно-

сительный адрес внешнего файла, содержащего код VBScript, который будет загружен и обработан интерпретатором браузера.

Internet Explorer поддерживает еще два параметра: `FOR` и `EVENT` тэга `<SCRIPT>`, непосредственно связанных с процедурами обработки событий, и речь о которых пойдет немного ниже.

В каком месте документа лучше размещать код сценария? В любом месте документа и в неограниченном количестве. Но, как и в случае с JavaScript, определения процедур рекомендуется размещать в разделе `<HEAD>`, хотя это и не обязательно. Подобная практика улучшает чтение кода сценария, позволяя избегать поиска процедуры по всему документу. Более того, при интерпретации браузером документа раздел `<HEAD>` интерпретируется до начала обработки тела документа, следовательно, все процедуры будут размещены в памяти до того, как в документе встретится их вызов.

Если читатель прочитал раздел *"Язык создания сценариев JavaScript"*, то он помнит, что каждый объект, соответствующий какому-либо элементу HTML-документа, имеет возможность перехватывать и обрабатывать события, возникающие в результате разнообразных действий пользователя с данным объектом (установка курсора мыши над связью, щелчок на кнопке формы, нажатие клавиши клавиатуры при установленном в поле ввода курсоре и т. д.). Для этого в тэги HTML, описывающие элементы документа, введены специальные параметры обработки событий: `onMouseOver`, `onClick`, `onKeyUp` и т. д. Значениями этих параметров является код вызова процедур, которые выполняются в ответ на произошедшее событие.

Задавать выполняемый код можно несколькими способами. Самый простой — задание кода в качестве значения параметра с указанием в начале строки используемого языка:

```
<BODY>
<P>Пример обработки событий в VBScript<BR>
<FORM>
<INPUT TYPE="BUTTON" VALUE="Нажми меня" NAME="Button1"
       OnClick="vbscript: Alert 'Сообщение в ответ на щелчок!'">
</FORM>
</BODY>
```

Представленный фрагмент кода задает в документе кнопку с надписью "`Нажми меня`", щелчок на которой приводит к вызову диалогового окна с надписью "`Сообщение в ответ на щелчок!`".

Префикс `vbscript:` сообщает интерпретатору, что необходимо использовать язык сценариев VBScript. Далее задается код VBScript, вызывающий метод `Alert` объекта `Window`. Подобный метод применим, когда необходимо выполнить один-два оператора языка.

Примечание

Если в качестве префикса указать javascript:, то будет вызван интерпретатор языка JavaScript, а последующий код должен быть, соответственно, написан на языке JavaScript.

Второй способ использует параметры FOR и EVENT в тэге <SCRIPT>. Предыдущий фрагмент может быть переписан следующим образом:

```
<BODY>
<P>Пример обработки событий в VBScript<BR>
<FORM>
<INPUT TYPE="BUTTON" VALUE="Нажми меня" NAME="Button1">
<SCRIPT FOR="Button1" EVENT="OnClick" LANGUAGE="VBScript">
<!--
     Alert 'Сообщение в ответ на щелчок!'
'-->
</SCRIPT>
</FORM>
```

При использовании данного способа задания процедуры обработки события в тэге <INPUT>, определяющем кнопку формы, параметр обработки события Click не задается, но в тэге <SCRIPT>, определяющем код сценария, параметр EVENT определяет, в ответ на какое событие будет выполняться код, а параметр FOR — для какого объекта. Значением первого параметра является имя параметра обработки события (onClick) тэга элемента, а значением второго параметра — имя элемента (Button1), для которого вызывается код, определенное в параметре NAME тэга элемента.

Примечание

Указанный способ обработки события распознается только браузером Internet Explorer, и поэтому его следует избегать. Здесь он упомянут исключительно ради полноты описания.

Третий способ задания процедуры обработки события унаследован VBScript от своего "родителя" Visual Basic и заключается в специальном именовании процедуры обработки события при ее объявлении. Любая процедура, имя которой составлено из имени объекта и имени события этого объекта с префиксом (on), соединенных знаком подчеркивания, рассматривается VBScript как процедура обработки указанного события для указанного объекта. Процедуру обработки события для рассмотренного выше примера можно определить следующим образом:

```
<SCRIPT TYPE="text/vbscript" LANGUAGE="VBScript">
<!--
Sub Button1_onClick()
    Alert "Сообщение в ответ на щелчок!"
End Sub
```

```
'-->
</SCRIPT>
```

Этот способ удобен тем, что можно в одном тэге `<SCRIPT>` собрать *все* процедуры обработки событий *всех* элементов документа.

Для тех, кто программирует на Visual Basic или Visual Basic for Application последний способ инициирования процедуры обработки события, вероятно, окажется самым подходящим. Более того, в специальных приложениях Windows, предоставляющих среду разработки HTML-страниц на языке VBScript, этот способ является основным.

Практические примеры

В этом разделе приводится несколько примеров программирования сценариев VBScript.

Плавающий фрейм

Тэг `<IFRAME>` задает плавающий фрейм на HTML-странице. Его можно использовать для отображения динамически создаваемой страницы. Такая техника полезна, например, при написании учебной страницы по языкам сценариев. В тексте можно привести код изучаемой конструкции, а при щелчке кнопкой мыши внутри плавающего фрейма она отобразится так, как ее увидит конечный пользователь.

Прежде всего, необходимо создать страницу с плавающим фреймом, и организовать перехват обработки какого-либо события плавающего фрейма. Ниже представлен текст страницы, на которой при получении плавающим фреймом фокуса (обработчик событий `onFocus`) вызывается процедура VBScript, динамически создающая содержимое, отображаемое в этом фрейме:

```
<BODY BGCOLOR="FFFF00">
<P>Щелчок кнопкой мыши на плавающем фрейме приводит
к отображению в нем содержимого</P>
<IFRAME WIDTH="400" HEIGHT="100" FRAMEBORDER="1"
        NAME="fra1" OnFocus="FillIFrame">
</IFRAME>
</BODY>
```

При щелчке на плавающем фрейме он получает фокус, и происходит вызов процедуры `FillIFrame`. На рис. 9.18 показана эта страница, отображенная в окне браузера.

Теперь остается только создать процедуру VBScript, создающую содержимое плавающего фрейма. Ее текст представлен ниже и должен быть размещен в разделе `<HEAD>` предыдущего документа:

```
<SCRIPT LANGUAGE="VBScript">
<!--
Sub FillIFrame()
 With Parent.Frames("fra1").Document
  .Clear
  .WriteLn "<HTML>"
  .WriteLn "<BODY BGCOLOR='00CCFF'>"
  .WriteLn "<P>Тест работы с плавающим фреймом!<BR>"
  .WriteLn "<FORM>"
  .WriteLn "<SCRIPT LANGUAGE=" & Chr(34) & "VBScript" & _
             Chr(34) & "><" & Chr(47) & "SCRIPT>"
  .WriteLn "<INPUT TYPE=BUTTON VALUE=" & Chr(34) & _
             "Нажми" & Chr(34) & " NAME=Button1 "
  .WriteLn "OnClick='Alert " & Chr(34) &        _
             "Простое сообщение." & Chr(34) & _
            ", vbExclamation, " & Chr(34) &  _
             "VBScript тест" & Chr(34) & "'>"
  .WriteLn "<" & Chr(47) & "FORM>"
  .WriteLn "<" & Chr(47) & "BODY>"
  .WriteLn "<" & Chr(47) & "HTML>"
  .Close
 End With
End Sub
'-->
</SCRIPT>
```

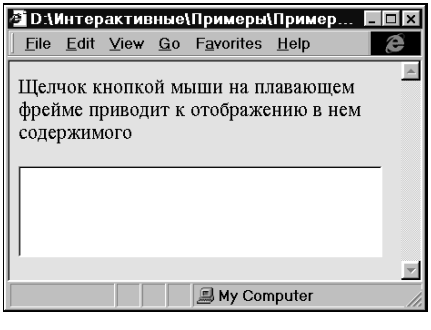


Рис. 9.18. Страница с плавающим фреймом

Оператор `With` задает объект, методы которого вызываются в теле оператора. В нашем примере — это документ, отображаемый в плавающем фрейме. Методы `WriteLn` записывают в документ строки, содержащие тэги HTML, и тем самым динамически формируют документ, отображаемый во фрейме.

Щелчок кнопкой мыши при расположении ее курсора в области фрейма приводит к возникновению события `Focus`, которое интерпретатор перехватывает и запускает на выполнение процедуру `FillIFrame()`. Результат отображения динамически созданной страницы показан на рис. 9.19.

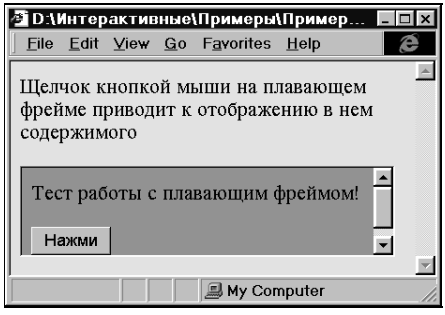


Рис. 9.19. Содержимое фрейма после получения им фокуса

Это полноправная страница. При щелчке на кнопке, расположенной во фрейме, отобразится диалоговое окно подсказки — действие, определенное в процедуре обработки события `Click` кнопки страницы.

Баннер

Можно организовать динамическое отображение разных страниц во фрейме, имитируя, таким образом, работу баннера. Реализация подобного процесса осуществляется вызовом разных процедур, готовящих и отображающих соответствующие страницы во фрейме с использованием функции `SetTimeOut`. В этом примере мы создадим плавающий фрейм, в котором будет постепенно по частям отображаться строка "Издательство BHV", затем также по частям она будет исчезать, и процесс начнется сначала. Каждая динамически появляющаяся во фрейме строка будет отображаться разным цветом, создавая эффект "неоновой" рекламы.

Как и в предыдущем примере, сначала создадим HTML-страницу с плавающим фреймом, а затем напишем сценарий, реализующий необходимый нам процесс отображения. Текст HTML-страницы выглядит следующим образом:

```
<BODY BGCOLOR="FFFF00">
<P ALIGN="center">
<IFRAME NAME="frameBHV" HEIGHT="70" WIDTH="200"></IFRAME></P>
Вас приветствует "Издательство BHV", Санкт-Петербург!
</BODY>
```

В верхней части страницы отображается плавающий фрейм, ссылаться на который из сценария можно при помощи имени `frameBHV`. Ниже фрейма располагается текст приветствия.

Теперь приступим к созданию сценария. Прежде всего определим, что отображаемая во фрейме строка полностью будет появляться в нем за пять отображений: сначала отобразится "Из", затем "Издат", далее "Издатель", потом "Издательство" и наконец "Издательство BHV". Исчезать она также будет за пять отображений, но в обратном порядке. Для каждого отображения необходимо динамически создать соответствующую страницу, что и реализуется

пятью процедурами AnimFrame с соответствующим номером в конце имени процедуры.

В них используются переменные Page1, Page2, Page3, Page4 и Page5, содержащие определения цветов фона и текста, и переменная PageEnd, которая хранит завершающие тэги страницы.

```
Dim PageEnd
Page1 = "<BODY BGCOLOR='#00CCFF' TEXT='#0000FF'><PRE>"
Page2 = "<BODY BGCOLOR='#00CCFF' TEXT='#00FFFF'><PRE>"
Page3 = "<BODY BGCOLOR='#00CCFF' TEXT='#00FF00'><PRE>"
Page4 = "<BODY BGCOLOR='#00CCFF' TEXT='#FFFF00'><PRE>"
Page5 = "<BODY BGCOLOR='#00CCFF' TEXT='#FF0000'><PRE>"
PageEnd = "</PRE></BODY>"
```

Первая вызываемая процедура — процедура без параметров AnimFrame1, которая отображает во фрейме строку "Из" синим цветом, и по прошествии 1/4 секунды вызывает процедуру AnimFrame2 с параметром True. Это действие реализуется функцией SetTimeOut, которая выполняет код, заданный первым параметром, через количество миллисекунд, определяемых вторым параметром этой функции.

```
Sub AnimFrame1
  Window.frameBHV.Document.Write Page1 & "Из" & EndPage
  Window.frameBHV.Document.Close
  SetTimeOut "AnimFrame2 True", 250, "VBScript"
End Sub
```

Процедура AnimFrame2 отображает во фрейме строку "Издат" голубым цветом и вызывает процедуру AnimFrame3 или AnimFrame1, в зависимости от того, должна ли появиться вся надпись (параметр forw равен True) или вся надпись должна исчезнуть (параметр forw равен False).

```
Sub AnimFrame2(forw)
  Window.frameBHV.Document.Write Page2 & "Издат" & EndPage
  Window.frameBHV.Document.Close
  If forw Then
       SetTimeOut "AnimFrame3 True", 250, "VBScript"
  Else
       SetTimeOut "AnimFrame1", 250, "VBScript"
  End If
End Sub
```

Процедуры AnimFrame3 и AnimFrame4 осуществляют аналогичные действия, отображая соответственно строки "Издатель" и "Издательство" и вызывая другие процедуры для выполнения необходимых действий.

```
Sub AnimFrame3(forw)
  Window.frameBHV.Document.Write Page3 & "Издатель" & EndPage
```

```
  Window.frameBHV.Document.Close
  If forw Then
       SetTimeOut "AnimFrame4 True", 250, "VBScript"
  Else
       SetTimeOut "AnimFrame2 False", 250, "VBScript"
  End If
End Sub
Sub AnimFrame4(forw)
  Window.frameBHV.Document.Write Page4 & "Издательство" & EndPage
  Window.frameBHV.Document.Close
  If forw Then
       SetTimeOut "AnimFrame5", 250, "VBScript"
  Else
       SetTimeOut "AnimFrame3 False", 250, "VBScript"
  End If
End Sub
```

Последняя, пятая процедура `AnimFrame5` отображает строку "Издательство BHV" и запускает процесс отображения в обратном порядке (процедура `AnimFrame4` вызывается с параметром `False`).

```
Sub AnimFrame5
  Window.frameBHV.Document.Write Page5 & "Издательство BHV" & EndPage
  Window.fraHello.Document.Close
  SetTimeOut "AnimFrame4 False", 250, "VBScript"
End Sub
```

Теперь, для запуска всего процесса необходимо инициализировать выполнение первой процедуры `AnimFrame1` при загрузке документа в окно браузера. Это действие реализуется в процедуре обработки события `Load` объекта `window`:

```
Sub Window_OnLoad()
       AnimFrame1
End Sub
```

Результаты отображения информации во фрейме в разные моменты времени показаны на рис. 9.20 и 9.21.

Примечание

На рис. 9.20 строка во фрейме отображается синим цветом, а на рис. 9.21 — красным.

Предупреждение

Описанный подход к циклическому вызову функций требует постоянной работы интерпретатора, поэтому следует помнить, что подобная техника программирования значительно загружает процессор компьютера клиента.

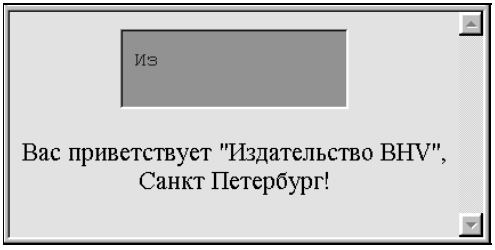


Рис. 9.20. Отображение строки "Из" во фрейме страницы

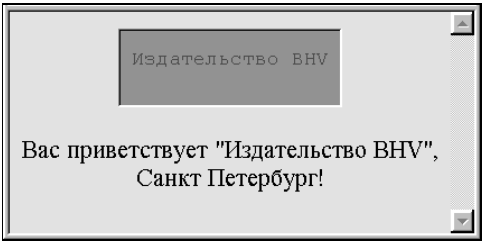


Рис. 9.21. Отображение строки "Издательство BHV" во фрейме страницы

CGI-сценарии и язык Perl

Основные понятия

Основу "Всемирной паутины" WWW составляют *Web-узлы* — компьютеры, на которых выполняется специальная программа — *Web-сервер*, ожидающая запроса со стороны клиента на выдачу документа. Документы сохраняются на Web-узле, как правило, в формате HTML. Клиентом Web-сервера является программа-браузер, выполняющаяся на удаленном компьютере, которая осуществляет запрос к Web-серверу, принимает запрошенный документ и отображает его на экране.

Аббревиатура CGI (Common Gateway Interface) обозначает часть Web-сервера, которая может взаимодействовать с другими программами, выполняющимися на этом же Web-узле, и в этом смысле является шлюзом (*gateway* — шлюз) для передачи данных, полученных от клиента, программам обработки, таким как СУБД, электронные таблицы, и др. CGI включает общую среду (набор переменных) и протоколы для взаимодействия с этими программами.

Общая схема работы CGI состоит из следующих элементов.

- *Получение Web-сервером информации от клиента-браузера.* Для передачи данных Web-серверу в языке HTML имеется средство, называемое *форма*. Форма задается в HTML-документе при помощи тэгов `<FORM>...</FORM>` и состоит из набора *полей ввода*, отображаемых браузером в виде графических элементов управления: селекторных кнопок, опций, строк ввода текста, управляющих кнопок и т. д. (рис. 9.22).
- *Анализ и обработка полученной информации.* Данные, извлеченные из HTML-формы, передаются для обработки CGI-программе. Они не всегда могут быть обработаны CGI-программой самостоятельно. Например, они могут содержать запрос к некоторой базе данных, которую CGI-программа читать "не умеет". В этом случае CGI-программа на основании полученной информации формирует запрос к компетентной программе, выполняющейся на том же компьютере. CGI-программа может быть на-

писана на любом языке программирования, имеющем средства обмена данными между программами. В среде UNIX для этой цели наиболее часто используется язык Perl, а так как UNIX является наиболее популярной операционной системой для Web-серверов, то можно считать Perl наиболее популярным языком CGI-программирования. Программа на языке Perl представляет собой последовательность операторов, которые *интерпретатор* языка выполняет при каждом запуске без преобразования исходного текста программы в выполняемый двоичный код. По этой причине CGI-программы называют также *CGI-сценариями* или *CGI-скриптами.*

- *Создание нового HTML-документа и пересылка его браузеру.* После обработки полученной информации CGI-программа создает динамический или, как говорят, виртуальный HTML-документ, или формирует ссылку на уже существующий документ и передает результат браузеру.

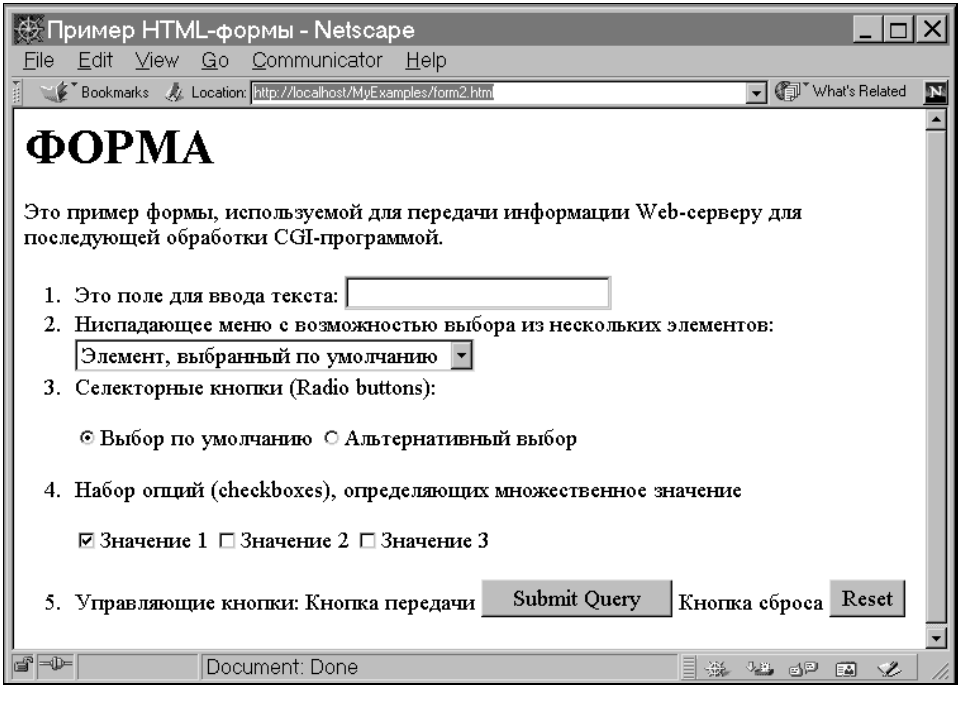


Рис. 9.22. Пример отображения HTML-формы браузером

HTML-формы

HTML-формы предназначены для пересылки данных от удаленного пользователя к Web-серверу. С их помощью можно организовать простейший диалог между пользователем и сервером, например, регистрацию пользователя

на сервере или выбор нужного документа из представленного списка. Формы поддерживаются всеми популярными браузерами. Различные аспекты передачи данных Web-серверу будут рассмотрены ниже в разделе *"Передача информации CGI-программе"*. В этом разделе рассмотрены средства языка HTML, используемые для создания форм. В описании тэгов приведены только наиболее употребительные параметры и опущены параметры, специфические для отдельных браузеров.

Тэг *<FORM>*

В HTML-документе для задания формы используются тэги `<FORM>...</FORM>`, отмечающие, соответственно, начало и конец формы. Документ может содержать несколько форм, но они не могут быть вложены одна в другую. Тэг `<FORM>` имеет параметры `ACTION`, `METHOD` и `ENCTYPE`. Отдельные браузеры (Netscape, Internet Explorer) поддерживают дополнительные параметры помимо стандартных, например, `CLASS`, `NAME`, `STYLE` и др. В общем виде форма задается следующим образом:

```
<FORM ACTION="URL" METHOD=метод_передачи ENCTYPE=MIME-тип>
содержание_формы
</FORM>
```

Параметр `ACTION` является единственным обязательным. Его значением является URL-адрес CGI-программы, которая будет обрабатывать информацию, извлеченную из данной формы.

Параметр `METHOD` определяет метод пересылки данных, содержащихся в форме, от браузера к Web-серверу. Он может принимать два значения: `GET` (по умолчанию) и `POST`.

Примечание

Взаимодействие между клиентом-браузером и Web-сервером осуществляется по правилам, заданным протоколом HTTP, и состоит из *запросов* клиента и ответов сервера. Запрос разбивается на три части. В первой строке запроса содержится HTTP-команда, называемая *методом*, URL-адрес запрашиваемого файла и номер версии протокола HTTP. Вторая часть — заголовок запроса. Третья часть — тело запроса, собственно данные, посылаемые серверу. *Метод* сообщает серверу о цели запроса. В протоколе HTTP определены несколько методов. Для передачи данных формы в CGI-программу используются два метода: `GET` и `POST`.

При использовании метода `GET` данные формы пересылаются в составе URL-запроса, к которому присоединяются после символа "?" в виде совокупности пар *переменная=значение*, разделенных символом "&". В этом случае первая строка запроса может иметь следующий вид:

```
GET /cgi-bin/cgi-program.pl?name=Mike&surname=Ivanoff   HTTP/1.1
```

URL запроса — Данные формы

После выделения данных из URL сервер присваивает их переменной среды QUERY_STRING, которая может быть использована CGI-программой.

При использовании метода POST данные формы пересылаются Web-серверу в теле запроса, после чего передаются сервером в CGI-программу через стандартный ввод.

Значением параметра ENCTYPE является *медиа-тип*, определяющий формат кодирования данных при передаче их от браузера к серверу. Браузер кодирует данные, чтобы исключить их искажение в процессе передачи. Возможны два значения этого параметра: *application/x-www-form-urlencoded* (по умолчанию) и *multipart/form-data*.

Примечание

Одним из первых применений Интернета была электронная почта, ориентированная на пересылку текстовых сообщений. Часто возникает необходимость вместе с текстом переслать данные в нетекстовом формате, например, упакованный zip-файл, рисунок в формате GIF, JPEG и т. д. Для того, чтобы пересылать средствами электронной почты такие файлы без искажения, они кодируются в соответствии с некоторым стандартом. Стандарт MIME (Multipurpose Internet Mail Extensions, многоцелевые расширения электронной почты для Интернета) определяет набор *MIME-типов*, соответствующих различным типам данных, и правила их пересылки по электронной почте. Для обозначения MIME-типа используется запись вида *тип/подтип*, где *тип* определяет общий тип данных, например, *text*, *image*, *application* (тип *application* обозначает специфический внутренний формат данных, используемый некоторой программой), а *подтип* — конкретный формат внутри типа данных, например, *application/zip*, *image/gif*, *text/html*. MIME-типы нашли применение в Web, где они называются также *медиа-типами*, для идентификации формата документов, передаваемых по протоколу HTTP. В HTML-форме параметр ENCTYPE определяет медиа-тип, который используется для кодирования и пересылки специального типа данных — содержимого формы.

Как видно из примера на рис. 9.22, форма отображается в окне браузера в виде набора стандартных элементов управления, используемых для заполнения полей формы значениями, которые затем передаются Web-серверу. Значение вводится в поле ввода пользователем или назначается по умолчанию. Для создания полей средствами языка HTML существуют специальные тэги: <INPUT>, <SELECT>, <TEXTAREA>, которые употребляются только внутри тэга <FORM>.

Тэг *<INPUT>*

- <INPUT TYPE=тип_поля_ввода NAME=имя_поля_ввода другие_параметры>

  Это наиболее употребительный тэг, с помощью которого можно генерировать внутри формы поля для ввода строки текста, пароля, имени файла, различные кнопки. Он имеет два обязательных параметра: TYPE и NAME. Параметр TYPE определяет тип поля: селекторная кнопка, кнопка передачи и др. Параметр NAME определяет имя, присваиваемое полю. Оно

не отображается браузером, а используется в качестве идентификатора значения, передаваемого Web-серверу. Остальные параметры меняются в зависимости от типа поля. Ниже приведено описание типов полей, создаваемых при помощи тэга `<INPUT>`, и порождаемых ими элементов ввода.

- `TYPE=TEXT`

  Создает элемент для ввода строки текста. Дополнительные параметры:

  - `MAXLENGTH=n`

    Задает максимальное количество символов, разрешенных в текстовом поле. По умолчанию — не ограничено.

  - `SIZE=n`

    Максимальное количество отображаемых символов.

  - `VALUE=начальное_значение`

    Первоначальное значение текстового поля.

- `TYPE=PASSWORD`

  Создает элемент ввода строки текста, отличающийся от предыдущего только тем, что все вводимые символы представляются в виде символа *.

Примечание

Поле `PASSWORD` не обеспечивает безопасности введенного текста, так как на сервер он передается в незашифрованном виде.

- `TYPE=FILE`

  Создает поле для ввода имени локального файла, сопровождаемое кнопкой **Browse**. Выбранный файл присоединяется к содержимому формы при пересылке на сервер. Имя файла можно ввести непосредственно или, воспользовавшись кнопкой **Browse**, выбрать его из диалогового окна, отображающего список локальных файлов. Для корректной передачи присоединенного файла следует установить значения параметров формы равными `ENCTYPE="multipart/form-data"` и `METHOD=POST`. В противном случае будет передана введенная строка, то есть маршрутное имя файла, а не его содержимое. Дополнительные параметры `MAXLENGTH` и `SIZE` имеют тот же смысл, что и для элементов типа `TEXT` и `PASSWORD`.

- `TYPE=CHECKBOX`

  Создает поле для установки флажка, который можно установить или сбросить (on/off (вкл./выкл.), истина/ложь). Элементы `CHECKBOX` можно объединить в группу, установив одинаковое значение параметра `NAME`. Дополнительные параметры:

  - `VALUE=строка`

    Значение, которое будет передано серверу, если данная кнопка выбрана. Если кнопка не выбрана, значение не передается. Обязательный параметр.

- CHECKED

  Если указан параметр CHECKED, элемент является выбранным по умолчанию.

Если флажки образуют группу, то передаваемым значением является строка разделенных запятыми значений параметра VALUE всех установленных флажков.

- TYPE=RADIO

  Создает элемент-переключатель, существующий только в составе группы подобных элементов, из которых может быть выбран только один. Все элементы группы должны иметь одинаковое значение параметра NAME. Отображается в виде круглой кнопки. Дополнительные параметры:

  - VALUE=строка

    Обязательный параметр, значение которого передается серверу при выборе данной кнопки. Должен иметь уникальное значение для каждого члена группы.

  - CHECKED

    Устанавливает элемент выбранным по умолчанию. Один и только один элемент в группе должен иметь этот параметр.

- TYPE=SUBMIT

  Создает кнопку передачи, нажатие которой вызывает пересылку на сервер всего содержимого формы. По умолчанию отображается в виде прямоугольной кнопки с надписью **Submit** (или **Submit Query** — для браузера Netscape). Дополнительный параметр

  VALUE=*название_кнопки*

  позволяет изменить надпись на кнопке. Параметр NAME для данного элемента может быть опущен. В этом случае значение кнопки не включается в список параметров формы и не передается на сервер. Если параметры NAME и VALUE присутствуют, например,

  ```
  <INPUT TYPE=SUBMIT NAME="submit_button" VALUE="OK">,
  ```

  то в список параметров формы, передаваемых на сервер, включается параметр submit_button="OK". Внутри формы могут существовать несколько кнопок передачи.

- TYPE=RESET

  Создает кнопку сброса, нажатие которой отменяет все сделанные изменения, восстанавливая значения полей формы на тот момент, когда она была загружена. По умолчанию отображается в виде прямоугольной кнопки с надписью **Reset**. Надпись можно изменить при помощи дополнительного параметра

  VALUE=*название_кнопки*

Значение кнопки **Reset** никогда не пересылается на сервер, поэтому у нее отсутствует параметр NAME.

- TYPE=IMAGE

  Создает элемент в виде графического изображения, действующий аналогично кнопке **Submit**. Дополнительные параметры:

  - SRC=*url_изображения*

    Задает URL-адрес файла с графическим изображением элемента.

  - ALIGN=*тип_выравнивания*

    Задает тип выравнивания изображения относительно текущей строки текста точно так же, как одноименный параметр тэга <IMG>.

  Если на изображении элемента щелкнуть мышью, то координаты указателя мыши в виде NAME.x=n&NAME.y=m включаются браузером в список параметров формы, посылаемых на сервер.

- TYPE=HIDDEN

  Создает скрытый элемент, не отображаемый пользователю. Информация, хранящаяся в скрытом поле, всегда пересылается на сервер и не может быть изменена ни пользователем, ни браузером. Скрытое поле можно использовать, например, в следующем случае. Пользователь заполняет форму и отправляет ее серверу. Сервер посылает пользователю для заполнения вторую форму, которая частично использует информацию, содержащуюся в первой форме. Сервер не хранит историю диалога с пользователем, он обрабатывает каждый запрос независимо и при получении второй формы не будет знать, как она связана с первой. Чтобы повторно не вводить уже введенную информацию, можно заставить CGI-программу, обрабатывающую первую форму, переносить необходимые данные в скрытые поля второй формы. Они не будут видимы пользователем и, в то же время, доступны серверу. Значение скрытого поля определяется параметром VALUE.

Тэг *<SELECT>*

```
<SELECT  NAME=имя_поля  SIZE=n  MULTIPLE>
элементы OPTION
</SELECT>
```

Тэг <SELECT> предназначен для того, чтобы организовать внутри формы выбор из нескольких вариантов без применения элементов типа CHECKBOX и RADIO. Дело в том, что если элементов выбора много, то представление их в виде флажков или кнопок-переключателей увеличивает размеры формы, делая ее труднообозримой. С помощью тэга <SELECT> варианты выбора более компактно представляются в окне браузера в виде элементов ниспадающего меню или списка прокрутки. Тэг имеет следующие параметры.

- NAME=строка

  Обязательный параметр. При выборе одного или нескольких элементов формируется список выбранных значений, который передается на сервер под именем NAME.

- SIZE=n

  Устанавливает число одновременно видимых элементов выбора. Если n=1, то отображается ниспадающее меню, если n>1, то — список прокрутки с n одновременно видимыми элементами.

- MULTIPLE

  Означает, что из меню или списка можно выбрать одновременно несколько элементов.

Элементы меню задаются внутри тэга <SELECT> при помощи тэга <OPTION>:

<OPTION SELECTED VALUE=*строка*>*содержимое_тэга*

Закрывающий тэг </OPTION> не требуется. Параметр VALUE содержит значение, которое пересылается серверу, если данный элемент выбран из меню или списка. Если значение этого параметра не задано, то по умолчанию оно устанавливается равным содержимому тэга <OPTION>. Например, элементы

```
<OPTION VALUE=Red>Red
<OPTION>Red
```

имеют одно значение Red. В первом случае оно установлено явно при помощи параметра VALUE, во втором — по умолчанию. Параметр SELECTED изначально отображает элемент как выбранный.

Тэг *<TEXTAREA>*

<TEXTAREA NAME=*имя* ROWS=*m* COLS=*n*>
текст
</TEXTAREA>

Тэг <TEXTAREA> создает внутри формы поле для ввода многострочного текста, отображаемое в окне браузера в виде прямоугольной области с горизонтальной и вертикальной полосами прокрутки. Для пересылки на сервер каждая введенная строка дополняется символами %0D%0A (ASCII-символы "Возврат каретки" и "Перевод строки" с предшествующим символом %), полученные строки объединяются в одну строку, которая и отправляется на сервер под именем, задаваемым параметром NAME. Параметры:

- NAME

  Необходимый параметр, используемый для идентификации данных при пересылке на сервер.

- COLS=n

  Задает число столбцов видимого текста.

- ROWS=n

  Задает число строк видимого текста.

Между тэгами <TEXTAREA> и </TEXTAREA> можно поместить текст, который будет отображаться по умолчанию.

Пример формы

Ниже представлен пример формы, включающей набор характерных полей (рис. 9.23), и HTML-код, использованный для ее создания.

Пример 9.5. HTML-код формы, представленной на рис. 9.23

```
<html><head><title>Пример формы</title></head>
<body>
<h2>Регистрационная страница Клуба любителей фантастики</h2>
Заполнив анкету, вы сможете пользоваться нашей электронной библиотекой.
<br>
<form method="get" action="/cgi-bin/registrar.cgi">
<pre>
Введите регистрационное имя: <input type="text" name="regname">
Введите пароль:       <input type="password" name="password1" max-
length=8>
Подтвердите пароль:   <input type="password" name="password2" max-
length=8>
</pre>
Ваш возраст:
<input type="radio" name="age" value="lt20" checked>До 20
<input type="radio" name="age" value="20_30">20-30
<input type="radio" name="age" value="30_50">30-50
<input type="radio" name="age" value="gt50">старше 50
<br><br>
На каких языках читаете:
<input type="checkbox" name="language" value="russian" checked>русский
<input type="checkbox" name="language" value="english">английский
<input type="checkbox" name="language" value="french">французский
<input type="checkbox" name="language" value="german">немецкий
<br><br>
Какой формат данных является для Вас предпочтительным
<br>
<select name="format" size=2>
       <option selected value="HTML">HTML
       <option value="Plain text">Plain text
       <option value="PostScript">PostScript
       <option value="PDF">PDF
</select>
```

```
<br><br>
Ваши любимые авторы:
<br>
<textarea name="wish" cols=40 rows=3>
</textarea>
<br><br>
<input type="submit" value="OK"> <input type="reset" value="Отменить">
</form>
</body>
</html>
```

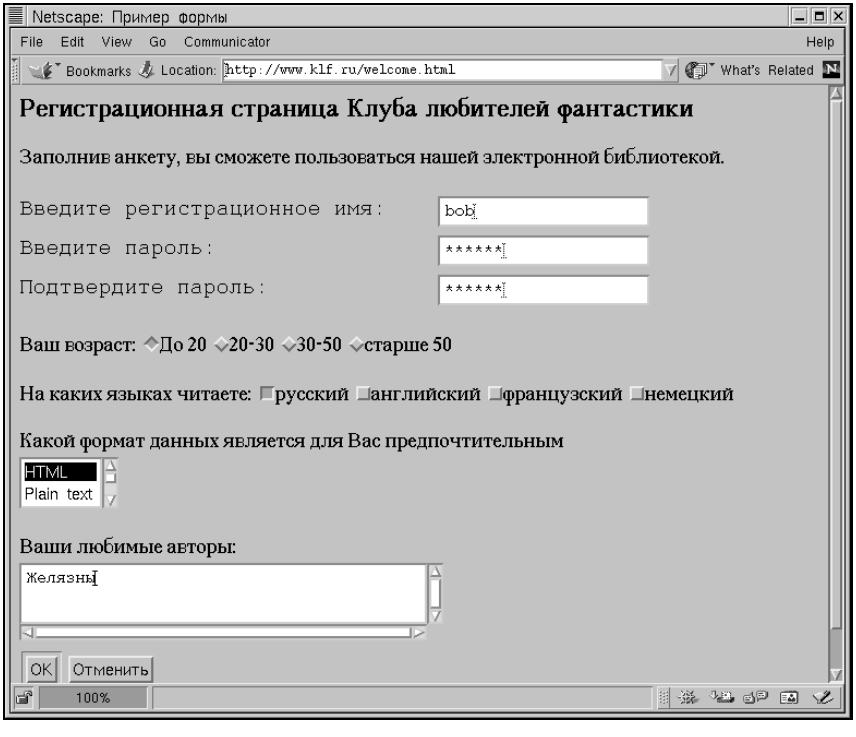


Рис. 9.23. Пример заполнения формы

Данная форма содержит:

- текстовое поле для ввода регистрационного имени пользователя;
- текстовое поле для ввода пароля, отображаемого в окне символами *;
- текстовое поле для подтверждения пароля, также отображаемого символами *;
- группу кнопок-переключателей для указания возраста пользователя (единственный выбор);

- группу полей для установки флажков указания языков, которыми владеет пользователь (множественный выбор);
- список прокрутки для указания предпочтительного формата данных (выбор из ограниченного списка);
- блок ввода многострочного текста для перечисления любимых авторов (неизвестное заранее количество строк);
- кнопку передачи с меткой **OK** (у этого элемента отсутствует параметр `NAME`, он не нужен, так как в данном примере всего одна кнопка передачи, а, значит, CGI-программе нет необходимости определять, от какой именно кнопки поступила команда передачи данных);
- кнопку сброса с меткой **Отменить**.

Итак, пользователь заполнил форму и щелкнул кнопку передачи **Submit**. Дальнейшее прохождение данных выглядит следующим образом.

1. Информация кодируется и пересылается на Web-сервер, который передает ее для обработки CGI-программе.
2. CGI-программа обрабатывает полученные данные, возможно обращаясь за помощью к другим программам, выполняющимся на том же компьютере, и генерирует новый "виртуальный" HTML-документ, либо определяет ссылку на уже имеющийся.
3. Новый HTML-документ или ссылка передаются CGI-программой Web-серверу для возврата клиенту.

Рассмотрим эти шаги более подробно.

Передача информации CGI-программе

Кодирование и пересылка данных формы

Как мы уже знаем, существуют два метода кодирования информации, содержащейся в форме:

- стандартный метод *application/x-www-form-urlencoded*, используемый по умолчанию;
- дополнительный *multipart/form-data*.

Второй метод нужен только в том случае, если к содержимому формы присоединяется локальный файл, выбранный при помощи элемента формы `<INPUT TYPE=FILE>`. В остальных случаях следует использовать метод кодирования по умолчанию.

Схема кодирования *application/x-www-form-urlencoded* одинакова для обоих методов пересылки (`GET` и `POST`) и заключается в следующем.

Для каждого элемента формы, имеющего имя, заданное параметром `NAME`, формируется пара "*name=value*", где *value* — значение элемента, введенное

пользователем или назначенное по умолчанию. Если значение отсутствует, соответствующая пара имеет вид "*name*=". Для радиокнопок и переключателей используются значения только выбранных элементов. Если элемент выбран, а значение параметра `VALUE` не определено, по умолчанию используется значение "`ON`".

Все пары объединяются в строку, в качестве разделителя служит символ "&". Так как имена и значения представляют собой обычный текст, то они могут содержать символы, недопустимые в составе URL (метод `GET` пересылает данные как часть URL). Такие символы заменяются последовательностью, состоящей из символа % и их шестнадцатеричного ASCII-кода. Символ пробела может заменяться не только кодом `%20`, но и знаком + (плюс). Признак конца строки, встречающийся в поле `TEXTAREA`, заменяется кодом `%0D%0A`. Такое кодирование называется URL-кодированием.

Закодированная информация пересылается серверу одним из методов `GET` или `POST`. Основное отличие заключается в том, как метод передает информацию CGI-программе.

При использовании метода `GET` данные формы пересылаются серверу в составе URL-запроса, к которому добавляются после символа "?" (вспомним, что запрос это формализованный способ обращения браузера к Web-серверу). Тело запроса в этом случае является пустым. Для формы из примера 9.5 запрос выглядит следующим образом:

```
GET /cgi-bin/registrar.cgi? reg-
name=bob&password1=rumata&password2=rumata&age=lt20&language=
russian&format=HTML&wish=%F6%C5%CC%D1%DA%CE%D9 HTTP/1.0
.
.       (заголовки запроса, сообщающие серверу информацию о клиенте)
.
<пусто>        (тело запроса)
```

Часть URL после символа "?" называется *строкой запроса*. Web-сервер, получив запрос, присвоит переменной среды `QUERY_STRING` значение строки запроса и вызовет CGI-программу, обозначенную в первой части URL до символа "?": `/cgi-bin/registrar.cgi`. CGI-программа `registrar.cgi` сможет затем обратиться к переменной `QUERY_STRING` для обработки закодированных в ней данных.

Предупреждение

Обратите внимание на то, что данные, введенные в поле типа `PASSWORD`, передаются открытым текстом без шифрования. При передаче данных методом `GET` они в составе URL помещаются в файл регистрации доступа access.log, обычно открытый для чтения всем пользователям. Таким образом "секретные" данные, введенные в поле типа `PASSWORD`, оказываются доступными посторонним.

Примечание

Метод GET позволяет передавать данные CGI-программе вообще без использования форм. Информацию, содержащуюся в приведенном выше URL, можно передать при помощи следующей гиперссылки, помещенной в HTML-документ:

```
<A HREF="http://www.domain/cgi-bin/registrar.cgi?
regname=bob&password1=rumata&password2=rumata&age=lt20&language=
russian&format=HTML&wish=%F6%C5%CC%D1%DA%CE%D9 ">CGI-программа</A>,
```

заменив в этом фрагменте символ "&" его символьным примитивом &#38 или &amp для правильной интерпретации браузером.

К сожалению, эта информация является статической. Форма же позволяет менять данные.

Строка запроса — не единственный способ передачи данных через URL. Другим способом является *дополнительная информация о пути* (*extra path information*), представляющая собой часть URL, расположенную после имени CGI-программы. Сервер выделяет эту часть и сохраняет ее в переменной среды PATH_INFO. CGI-программа может затем использовать эту переменную для извлечения данных. Например, URL

```
http://www.domain/cgi-bin/registrar.cgi/ regname=bob&password1=
rumata&password2=rumata&age=lt20&language=russian&format=HTML&wish=
%F6%C5%CC%D1%DA%CE%D9
```

содержит уже знакомые нам данные, но не в виде строки запроса, а в виде дополнительной информации о пути. При получении запроса с таким URL сервер сохранит данные в переменной среды.

```
PATH_INFO= "/regname=bob&password1=rumata&password2=rumata&age=
lt20&language=russian&format=HTML&wish=%F6%C5%CC%D1%DA%CE%D9"
```

Название объясняется тем, что обычно этим способом передается информация о местоположении какого-либо файла (*extra path information*). Например, URL

```
http://www.domain/cgi-bin/registrar.cgi/texts/jdk_doc.txt
```

содержит дополнительную информацию PATH_INFO="/texts/jdk_doc.txt" о местонахождении файла jdk_doc.txt относительно корневого каталога дерева документов. Другая переменная среды PATH_TRANSLATED содержит информацию об абсолютном местоположении файла в файловой системе, например,

```
PATH_TRANSLATED="/home/httpd/docs/texts/jdk_doc.txt"
```

а переменная DOCUMENT_ROOT содержит путь к корневому каталогу дерева документов, в нашем случае DOCUMENT_ROOT="/home/httpd/docs/".

При использовании метода POST данные формы пересылаются серверу в теле запроса. Если в примере 9.5 вместо метода GET использовать метод POST

```
<form method="post" action="/cgi-bin/registrar.cgi">,
```

то запрос клиента будет иметь следующий вид:

```
POST /cgi-bin/registrar.cgi HTTP/1.1
.
.       (заголовки запроса, сообщающие серверу информацию о клиенте)
.
```

```
Content-length: 126
regname=bob&password1=rumata&password2=rumata&age=lt20&language=
russian&format=HTML&wish=%F6%C5%CC%D1%DA%CE%D9
```

В этом фрагменте среди прочих заголовков выделен заголовок `Content-length`, сообщающий серверу количество байт, переданных в теле запроса. Это значение сервер присваивает переменной среды `CONTENT_LENGTH`, а данные посылает в стандартный ввод CGI-программы.

Примечание

В операционной системе UNIX для каждого процесса, выполняющего программу, по умолчанию система открывает три специальных файла: стандартный файл ввода (STDIN), стандартный файл вывода (STDOUT) и стандартный файл диагностики (STDERR) соответственно для передачи данных в программу, вывода результатов и сообщений программы. По умолчанию, если для программы не предусмотрено иное, эти специальные файлы ассоциированы с терминалом, т. е. стандартным устройством для ввода данных, которым является клавиатура, а для вывода результатов и сообщений — дисплей. Стандартные ввод, вывод и вывод сообщений могут быть перенаправлены средствами UNIX в произвольный файл или на вход другой программы.

Методы `GET` и `POST` имеют свои достоинства и недостатки. Метод `GET` обеспечивает лучшую производительность при пересылке форм, состоящих из небольшого набора коротких полей. При пересылке большого объема данных следует использовать метод `POST`, так как браузер или сервер могут накладывать ограничения на размер данных, передаваемых в составе URL, и отбрасывать часть данных, выходящую за границу. Метод `POST`, к тому же, является более надежным при пересылке конфиденциальной информации.

CGI-сценарии

Общие сведения

Назначение CGI-программы — создать новый HTML-документ, используя данные, содержащиеся в запросе, и передать его обратно клиенту. Если такой документ уже существует, то передать ссылку на него. Какой язык можно использовать для написания CGI-программ? Сам интерфейс CGI не накладывает ограничений на выбор языка программирования. Зная, какую задачу решает CGI-программа и каким образом она получает входную информацию, мы можем назвать свойства, которыми должен обладать язык CGI-программирования.

- *Средства обработки текста.* Необходимы для декодирования входной информации, поступающей в виде строки, состоящей из отдельных полей, разделенных символами-ограничителями.
- *Средства доступа к переменным среды.* Необходимы, так как с помощью переменных среды данные передаются на вход CGI-программы.

- *Возможность взаимодействовать с другими программами.* Необходима для обращения к СУБД, программам обработки графики и другим специальным программам.

Выбор языка зависит и от операционной системы Web-сервера. Большая часть имеющихся серверов предназначены для работы под управлением операционной системы UNIX. В этой операционной системе очень распространено использование скриптов — текстовых командных файлов, представляющих собой программы, состоящие из обращений к командам операционной системы и управляющих конструкций языка shell — командной оболочки UNIX. Программа shell является интерпретатором команд операционной системы и, в то же время, имеет встроенные средства, характерные для языков программирования: строковые переменные и управляющие конструкции — операторы цикла, перехода, условные операторы и т. д. Shell выполняет командные файлы как интерпретатор, т. е. считывает из файла и выполняет команды одну за другой с учетом управляющих конструкций, не преобразуя исходный текст в исполняемый двоичный код. При помощи скриптов осуществляется значительная часть работы по конфигурированию ОС. Видимо, с распространенностью командных процедур в UNIX и связано появление в этой операционной системе ряда интерпретирующих языков, предназначенных для создания сценариев: Perl, Python, Tcl. Все они могут использоваться для написания CGI-сценариев так же, как и традиционные языки программирования C/C++. Наибольшее распространение в качестве языка CGI-программирования получил язык Perl (Practical Extraction and Report Language). Он удовлетворяет перечисленным выше требованиям и, кроме того, обладает такими полезными свойствами, как:

- переносимость (существуют версии интерпретатора Perl для MS-DOS, Windows, Macintosh и множества других платформ);
- наличие готовых модулей, облегчающих CGI-программирование и свободно доступных в сети Интернет на многих ftp-узлах в архивах CPAN (Comprehensive Perl Archive Network).

Perl — достаточно простой язык (по утверждению его создателя). Во-первых, это интерпретируемый язык, а интерпретируемые языки в целом проще языков компилируемых. Во-вторых, его конструкции, в основном заимствованные из других языков, достаточно понятны тем, кто имеет отношение к программированию. Вместе с тем, это мощный язык, обладающий многими возможностями таких языков, как C, UNIX shell, awk (язык обработки текстовых данных в UNIX), sed (утилита UNIX, предназначенная для обработки текста) и, следовательно, требующий времени для усвоения. К счастью, для понимания и создания простых программ не требуется знания всех возможностей Perl, вполне достаточно начального уровня владения языком. Данный раздел не предназначен для того, чтобы служить введением или кратким справочником по языку Perl и технике программирования на нем. Задача другая — дать представление о том, как используется Perl в CGI-про-

граммировании. Поэтому мы не стараемся давать формальное определение конструкций языка, предпочитая пояснять их использование на конкретных примерах.

Примечание

При написании программ следует помнить, что язык Perl, как и ОС UNIX, является чувствительным к регистру символов, поэтому "Perl" и "PERL" обозначают не одно и то же.

Первая программа "Hello, world!", с которой начинается изучение любого языка программирования, на языке Perl выглядит так:

```
#!/usr/bin/perl
print "Hello, world!\n";
```

Первая строка сообщает, где находится интерпретатор языка Perl. В UNIX-системах она служит для того, чтобы сопоставить файлу с текстом Perl-программы обрабатывающее его приложение. Другие операционные системы ее игнорируют.

Функция `print` выводит список своих аргументов, преобразованных в строку, в стандартный вывод, по умолчанию ассоциированный с терминалом.

Управляющая последовательность `\n` вызывает переход на новую строку после вывода информации.

Отметим также, что большая часть строк Perl-программы завершается символом точка с запятой (;). Исключением являются строки, содержащие операторы цикла и условные операторы. Сохраним текст программы в файле `hello.pl`. Запустить программу можно по-разному, например, передав имя файла в качестве аргумента командной строки интерпретатору языка:

```
perl hello.pl
```

В ОС UNIX можно сделать файл выполняемым, применив команду

```
chmod 755 hello.pl,
```

и запустить его из командной строки:

```
./hello.pl
```

CGI-сценарий на языке Perl это программа, имеющая свою специфику, заключающуюся в том, что она, как правило, генерирует HTML-документ, посылаемый клиенту в виде *ответа сервера*. Ответ сервера, так же как и запрос клиента, имеет определенную структуру. Он состоит из следующих трех частей:

1. Строка состояния, содержащая три поля: номер версии протокола HTTP, код состояния и краткое описание состояния, например:

```
HTTP/1.0 200 OK          # Запрос клиента обработан успешно
HTTP/1.0 404 Not Found   # Документ по указанному адресу не существует
```

2. Заголовки ответа, содержащие информацию о сервере и о возвращаемом HTML-документе, например:

```
Date: Mon, 26 Jul 1999 18:37:07 GMT  # Текущая дата и время
Server: Apache/1.3.6                 # Имя и номер версии сервера
Content-type: text/html              # Описывает медиа-тип содержимого
```

3. Содержимое ответа — HTML-документ, являющийся результатом выполнения CGI-программы.

CGI-программа передает результат своей работы — HTML-документ — серверу, который возвращает его клиенту. При этом сервер не анализирует и не изменяет полученные данные, он может только дополнять их некоторыми заголовками, содержащими общую информацию (например, текущая дата и время) и информацию о самом себе (например, имя и версия сервера). Информация о содержимом ответа формируется CGI-программой и должна содержать как минимум один заголовок, сообщающий браузеру формат возвращаемых данных:

```
Content-type: text/html
```

Примечание

Информацию о заголовках можно найти в спецификации протокола HTTP. Мы же ограничимся еще одним примером. Если в качестве ответа клиенту посылается статический документ, например, подтверждение о получении заполненной формы, то неэффективно каждый раз создавать его заново. Лучше создать один раз и сохранить в файле. В этом случае CGI-сценарий вместо заголовка

```
Content-type: media-type
```

описывающего формат данных, формирует заголовок `Location: URL`, указывающий серверу местонахождение документа, который следует передать клиенту.

Заголовки отделяются от содержимого документа пустой строкой.

Преобразуем, учитывая сделанные замечания, программу hello.pl в CGI-сценарий, который сохраним в файле hello.cgi.

```
#!/usr/bin/perl
print "Content-type:text/html\n\n";
print "<html><head><title>HELLO</title></head>\n";
print "<body>\n";
print "<h2>Hello, world!</h2>\n";
print "</body></html>\n";
```

Если поместить файл hello.cgi в каталог CGI-программ Web-сервера, а затем обратиться к нему из браузера, то браузер отобразит HTML-документ, созданный программой hello.cgi (рис. 9.24).

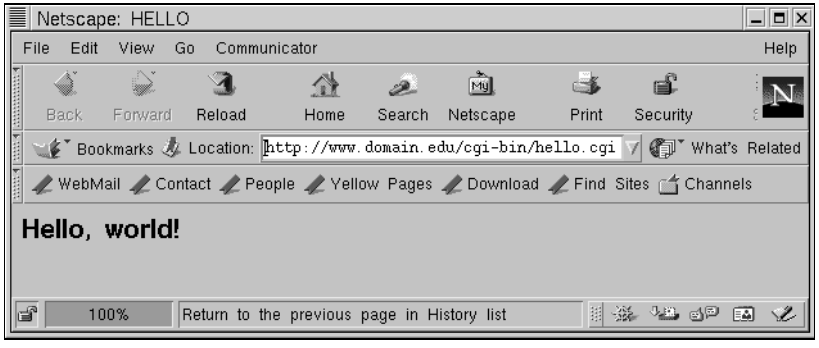


Рис. 9.24. Web-страница, сформированная программой hello.cgi

Примечание

Большинство Web-серверов по умолчанию предполагают, что файлы CGI-сценариев находятся в специальном каталоге, обычно называемом `cgi-bin`. Можно настроить сервер таким образом, чтобы все файлы, находящиеся в определенном каталоге, он воспринимал не как обычные документы, а как выполняемые сценарии. Можно также указать серверу, что все файлы с определенным расширением (например, CGI) должны рассматриваться как CGI-сценарии. Когда пользователь открывает URL, ассоциированный с CGI-программой, клиент посылает запрос серверу, запрашивая файл. Сервер распознает, что запрошенный адрес является адресом CGI-программы, и пытается выполнить эту программу. Подробности конфигурирования Web-серверов можно найти в соответствующей литературе и документации на конкретный сервер.

Переменные в языке Perl

В языке Perl существует три типа данных и соответственно им три типа переменных: скалярные переменные, массивы скалярных переменных и ассоциативные массивы скалярных переменных (или хеш-массивы). В Perl-программе переменные не нужно явно описывать перед их использованием. Тип переменной определяется начальным символом, предшествующим ее имени.

Скалярные переменные

Скалярный тип является базовым типом, на основании которого строятся более сложные структуры данных. Скалярная переменная содержит одно значение, которое может быть числом, строкой или ссылкой на другую переменную. Признаком скалярной переменной является начальный символ "$":

```
$name = "mike";
$x = 7;
$nameref = \$name;
```

В первой строке примера скалярной переменной `$name` присваивается строковое значение `"mike"`, во второй строке — скалярной переменной `$x` присваивается числовое значение `7`, в третьей строке — скалярной переменной `$nameref` присваивается значение ссылки на скалярную переменную `$name`. Операция `\x` создает ссылку на объект `x`, который может быть переменной, подпрограммой или константой.

И число, и строка относятся к скалярному типу данных, поэтому в программе скалярная переменная может последовательно принимать числовые и строковые значения, не вызывая сообщения о неправильном использовании типов:

```
#!/usr/bin/perl
$pi=3.1416;
$pi="Число пи равно $pi";
print "$pi\n";
```

В результате выполнение этой программы будет выведена строка:

```
Число пи равно 3.1416
```

Над данными скалярного типа в языке Perl определены арифметические и логические операции, показанные в табл. 9.14.

Таблица 9.14. *Основные арифметические и логические операции языка Perl*

Операция	Описание	Пример	Результат
`+`	Сложение	`5+3`	`8`
`-`	Вычитание	`3-5`	`-2`
`*`	Умножение	`2*3`	`6`
`/`	Деление	`10/2`	`5`
`%`	Остаток от деления	`20/3`	`2`
`**`	Возведение в степень	`2**3`	`8`
`.`	Конкатенация строк	`"Hello, "." world!"`	`"Hello, world!"`
`x`	Повторение первого операнда; количество повторений задается вторым операндом	`"!"x3`	`!!!`
`&`	Побитовое И	`123&456`	`72` $123=1111011_2$ $456=11001000_2$ $72=1001000_2$
`|`	Побитовое ИЛИ	`123|456`	`507` $123=1111011_2$ $456=11001000_2$ $507=11111011_2$

Таблица 9.14 *(окончание)*

Операция	Описание	Пример	Результат
^	Побитовое исключающее ИЛИ	123^456	435 123=1111011_2 456=11001000_2 435=10110011_2
>>	Побитовый сдвиг вправо	32>>2	8 32=100000_2 8=1000_2
<<	Побитовый сдвиг влево	32<<2	128 32=100000_2 128=10000000_2
&&	Логическое И: $a && $b = $a, если $a — ложь, иначе $b	1 && 2 32 && 0	2 0
||	Логическое ИЛИ: $a || $b = $a, если $a — истина, иначе $b	1 || 2 32 || 0	1 32

Примечание

В языке Perl нет специального типа для представления логических данных, но, в то же время, есть логические операции, выполняемые над скалярными величинами. При выполнении логических операций над данными скалярного типа следует руководствоваться следующими правилами:

- Любая строка, кроме пустой строки "" и строки "0", соответствует логическому значению "истина".
- Любое число, кроме 0, соответствует значению "истина".
- Любая ссылка соответствует значению "истина".
- Любое неопределенное значение соответствует значению "ложь".

Так как логические значения "истина" и "ложь" не имеют однозначной записи, то логические операции && и || в качестве результата возвращают последнее вычисленное в результате применения операции скалярное значение.

Операции сравнения скалярных величин определены отдельно для числовых и строковых значений, как показано в табл. 9.15.

Таблица 9.15. *Операции сравнения*

Операция	Описание	Пример	Результат
>	Числовое "больше чем"	2>3	ложь ("")
<	Числовое "меньше чем"	2<3	истина (1)

Таблица 9.15 *(окончание)*

Операция	Описание	Пример	Результат
`>=`	Числовое "больше или равно"	`3>=2`	`истина (1)`
`<=`	Числовое "меньше или равно"	`3<=3`	`истина (1)`
`==`	Числовое равенство	`2==3`	`ложь ("")`
`!=`	Числовое неравенство	`2!=3`	`истина (1)`
`gt`	Строковое "больше чем"	`"hat" gt "cat"`	`истина (1)`
`lt`	Строковое "меньше чем"	`"hat" lt "cat"`	`ложь ("")`
`ge`	Строковое "больше или равно"	`"hat" ge "hat"`	`истина (1)`
`le`	Строковое "меньше или равно"	`"pat" le "hat"`	`ложь ("")`
`eq`	Строковое равенство	`"cat" eq "cats"`	`ложь ("")`
`Ne`	Строковое неравенство	`"cat" ne "cats"`	`истина (1)`

Примечание

При сравнении строковых значений используется лексикографический порядок, принятый в словарях: `"a" lt "aa" lt "ab" lt ... "ay" lt "az"`.

В порядке убывания приоритета операции, перечисленные в табл. 9.14 и 9.15, располагаются следующим образом:

```
**
*       /       %       x
+       -       .
<<      >>
<       >       <=      >=      lt      gt      le      ge
==      !=      eq      ne
&
|       ^
&&
||
```

Внесем изменения в нашу первую программу hello.pl, добавив в нее скалярные переменные и некоторые новые конструкции:

```
#!/usr/bin/perl
$my_name="Издательство BHV, Санкт-Петербург";
print "Привет, привет. Как Ваше имя?\n";
$your_name=<STDIN>;
chomp($your_name);
print "Здравствуйте, $your_name. Вас приветствует $my_name.\n";
```

Запуск программы вызовет отображение на экране следующего диалога:

```
perl hello.pl
Привет, привет. Как Ваше имя?
mike
Здравствуйте, mike. Вас приветствует Издательство BHV, Санкт-Петербург.
```

Программа выводит приглашение ввести имя и считывает его в переменную `$your_name`, используя оператор присваивания `$your_name=<STDIN>`. Идентификатор `STDIN` обозначает дескриптор файла стандартного ввода программы. Дескриптор файла — это внутреннее имя файла для интерпретатора Perl. Он создается и связывается с конкретным файлом на диске при помощи специальной функции, о чем будет сказано дальше. Интерпретатор предоставляет Perl-программе несколько предопределенных, автоматически открываемых дескрипторов, которые он сам получает от операционной системы. Это дескриптор стандартного ввода `STDIN`, дескриптор стандартного вывода `STDOUT` и дескриптор стандартного вывода диагностики `STDERR`, обычно связанные с терминалом. Оператор `$your_name=<STDIN>` читает одну строку из файла, соответствующего дескриптору `STDIN`, то есть из стандартного ввода программы — терминала.

Считанное значение обрабатывается функцией `chomp`. Функция

```
chomp variable
```

удаляет из строкового аргумента `variable` признак конца строки. Если его не удалить, то в нашем примере последняя выводимая строка будет разорвана:

```
Здравствуйте, mike
. Вас приветствует Издательство BHV, Санкт-Петербург.
```

Массивы

Массив это упорядоченный список скалярных значений. Признаком переменной, обозначающей массив, является начальный символ имени `"@"`, например,

```
@names = ("Kate","Bob","Mike");
```

Здесь определен упорядоченный массив из трех элементов `"Kate"`, `"Bob"` и `"Mike"`. Значение массива в правой части оператора присваивания представляется конструкцией, состоящей из списка скалярных значений, разделенных запятыми и заключенных в круглые скобки. Префикс `@` в имени переменной используется для обозначения массива как совокупности элементов. Для обращения к единичному элементу используется имя массива с префиксом `$`, сопровождаемое индексом, заключенным в квадратные скобки. Первому элементу массива всегда соответствует индекс 0.

```
$names[0] = "Ann";
@selected_names = @names[1..2];
```

Обратите внимание на использование префиксов в именах переменных. В первом случае происходит обращение к *единичному элементу* массива `$names[0]` для изменения его значения. Во втором случае определяется новый массив `@selected_names`, которому присваивается *массив значений* `@names[1..2]`, состоящий из элементов `"Bob"` и `"Mike"`.

Использование имени массива в качестве скалярной переменной позволяет легко определить число элементов массива:

```
$number_of_names = @names;
```

Значение переменной `$number_of_names` равно числу элементов в массиве `@names`.

В языке Perl можно добавлять новые элементы к массиву при помощи присваивания значений:

```
$names[3] = "John";
```

Теперь массив `@names` состоит из четырех элементов: `"Ann","Bob","Mike"` и `"John"`.

Функции для работы с массивами

Perl располагает набором функций для работы с массивами. Ниже определены наиболее употребительные из них.

- `splice @ARRAY, OFFSET [, LENGTH [, LIST]]`

  Удаляет `LENGTH` элементов из массива `@ARRAY`, начиная с элемента с индексом `OFFSET`, и заменяет их элементами списка `LIST`. Возвращает удаленные элементы. Если количество удаляемых элементов `LENGTH` не указано, удаляются все элементы, начиная с элемента `OFFSET`.

  Пример:

  ```
  @colors=("red","green","black","white","blue");
  splice @colors,2,2, "orange","brown";
  ```

  Элементы `"black"` и `"white"` массива `@colors` заменяются на элементы `"orange"` и `"brown"`, соответственно.

- `shift @ARRAY`

  Осуществляет сдвиг элементов массива `@ARRAY` влево с отбрасыванием первого элемента. Возвращает значение удаленного первого элемента массива.

- `unshift @ARRAY, LIST`

  Добавляет к началу массива `@ARRAY` список элементов `LIST`. Возвращает число элементов в новом массиве.

- `pop @ARRAY`

  Удаляет последний элемент массива `@ARRAY` и возвращает его значение.

- `push @ARRAY, LIST`

  Добавляет в конец массива `ARRAY` элементы из списка `LIST`.

- `sort @ARRAY`

  Сортирует элементы массива `@ARRAY` и возвращает отсортированный массив.

- `reverse @ARRAY`

  Возвращает массив `@ARRAY` с элементами, расположенными в обратном порядке.

- `join EXPR, @ARRAY`

  Объединяет в одну строку элементы массива `@ARRAY`, используя в качестве разделителя полей значение `EXPR`, и возвращает эту строку.

  Пример:

```
@numbers=("1","2","3");
$string=join "***", @numbers;
```

  Значение переменной `$string` равно `"1***2***3"`.

Ассоциативные массивы

Ассоциативный массив или хеш-массив, это неупорядоченное множество пар скалярных величин "ключ-значение". Поскольку хеш-массив представляет собой неупорядоченное, а, значит, ненумерованное множество элементов, то обращение к отдельному элементу осуществляется не через его индекс, которого нет, а через *ключ*, ассоциированный с этим элементом. Имя переменной, обозначающей ассоциативный массив, содержит в качестве префикса символ `"%"`, например,

```
%phones = ("Ann", "123-45-67",
           "Bob", "765-43-21",
           "Mike", "543-21-76");
```

или более наглядно

```
%phones =("Ann" => "123-45-67",
          "Bob" => "765-43-21",
          "Mike" => "543-21-76");
```

В этом примере определен хеш-массив `%phones`, содержащий номера телефонов, ассоциированные с именами их владельцев. Имя служит ключом для выбора отдельного элемента:

```
$ann_phone=$phones{'Ann'};
```

Переменной `$ann_phone` присвоено значение `"123-45-67"`. Снова обратите внимание на использование префиксов в именах переменных. Префикс `$`

в имени хеш-массива, за которым следует значение ключа в фигурных скобках, используется для обращения к единичному элементу, в то время, как префикс `%` — для ссылки на весь ассоциативный массив. Символы "," и "=>", используемые для создания пары "ключ-значение", являются синонимами.

Для получения списка всех ключей или всех значений ассоциативного массива можно использовать функции `keys` и `values`, соответственно:

```
@persons= keys %phones;
@numbers= values %phones;
```

Массив `@persons` содержит список ключей хеш-массива `%phones`, расположенных в случайном порядке. Массив `@numbers` содержит список значений хеш-массива `%phones`, расположенных в порядке, соответствующем порядку расположения ключей в массиве `@persons`:

`("Bob", "Ann", "Mike")` — массив `@persons`,

`("234-56-78", "123-45-67"," 345-67-89")` — массив `@numbers`

Можно добавлять элементы к ассоциативному массиву при помощи присваивания:

```
$phones{'john'}="456-78-90"
```

или удалять при помощи функции `delete`:

```
delete $phones{'john'}
```

Для выполнения операций над множеством элементов массива в языках программирования существуют операторы цикла. Perl имеет несколько таких операторов, из которых мы отметим один, позволяющий работать как с обычными, так и с ассоциированными массивами.

```
foreach $VAR (LIST) {
    ...
}
```

В операторе цикла `foreach` для каждого значения, принимаемого переменной `$VAR` из списка `LIST`, выполняется последовательность операторов, заключенная в фигурные скобки {}. Аргумент `LIST` может быть списком скалярных значений, разделенных запятыми, или переменной, представляющей массив или хеш-массив.

Пример:

```
foreach $name (@names)
{
       print "$name\n";
}
foreach $key ( keys %phones)
```

```
{
        print "$key ' phone is $phones{$key}\n";
}
```

Результатом выполнения первого оператора `foreach` будет вывод списка имен, являющихся элементами массива `@names`:

```
Ann
Bob
Mike
```

В результате выполнения второго оператора `foreach` будут выведены следующие строки:

```
Bob 's phone is 234-56-78
Ann 's phone is 123-45-67
Mike 's phone is 345-67-89
```

Обратите внимание на то, что список телефонов выведен в случайном порядке. Применение функции `sort` позволяет получить список, отсортированный по значениям ключей:

```
foreach $key ( sort( keys %phones))
{
        print "$key ' phone is $phones{$key}\n";
}
```

Читатель, должно быть, уже догадался, что ассоциативные массивы естественным образом находят применение в CGI-сценариях для обработки данных формы, передаваемых в виде совокупности пар "имя=значение".

Все операции в языке Perl выполняются в определенном *контексте*. Существуют два основных контекста: скалярный и контекст массивов. Результат операции зависит от того, в каком контексте она выполняется. Например, допустимы такие операции присваивания:

```
@a=("one", "two","three");
$a=("one", "two","three");
```

В первом случае операция выполняется в контексте массива, она присваивает целый список значений переменной-массиву `@a`. Во втором случае операция выполняется в скалярном контексте, она присваивает единственное значение, равное числу элементов списка (3), скалярной переменной `$a`. В контексте массива допустима, например, операция присваивания `($one, $two, $three)= @a`, в результате которой переменным `$one, $two, $three` будут присвоены последовательные значения из массива `@a`.

Переменные среды CGI

В зависимости от метода данные формы передаются в CGI-программу или через стандартный ввод (`POST`), или через переменную среды `QUERY_STRING`

(GET). Помимо этих данных CGI-программе доступна и другая информация, поступившая от клиента в заголовках запроса или предоставленная Web-сервером. Эта информация сохраняется в переменных среды UNIX. С некоторыми из них мы уже познакомились ранее. В табл. 9.16 перечислены переменные, обычно используемые в CGI.

Таблица 9.16. *Переменные среды CGI*

Переменная среды	Описание
GATEWAY_INTERFACE	Версия CGI, которую использует сервер
SERVER_NAME	Доменное имя или IP-адрес сервера
SERVER_SOFTWARE	Имя и версия программы-сервера, отвечающей на запрос клиента (например, Apache 1.3)
SERVER_PROTOCOL	Имя и версия информационного протокола, который был использован для запроса (например, HTTP 1.0)
SERVER_PORT	Номер порта компьютера, на котором работает сервер (по умолчанию 80)
REQUEST_METHOD	Метод, использованный для выдачи запроса (GET, POST)
PATH_INFO	Дополнительная информация о пути
PATH_TRANSLATED	Та же информация, что и в переменной PATH_INFO с префиксом, задающим путь к корневому каталогу дерева Web-документов
SCRIPT_NAME	Относительное маршрутное имя CGI-сценария (например, /cgi-bin/program.pl)
DOCUMENT_ROOT	Корневой каталог дерева Web-документов
QUERY_STRING	Строка запроса — информация, переданная в составе URL запроса после символа "?"
REMOTE_HOST	Имя удаленной машины, с которой сделан запрос
REMOTE_ADDR	IP-адрес удаленной машины, с которой сделан запрос
REMOTE_USER	Идентификационное имя пользователя, посылающего запрос
CONTENT_TYPE	Медиа-тип данных запроса, например, "text/html"
CONTENT_LENGTH	Количество байт в теле запроса, переданных в CGI-программу через стандартный ввод
HTTP_HOST	Хост-имя компьютера, на котором работает сервер
HTTP_FROM	Адрес электронной почты пользователя, направившего запрос

Таблица 9.16 *(окончание)*

Переменная среды	Описание
HTTP_ACCEPT	Список медиа-типов, которые может принимать клиент
HTTP_USER_AGENT	Браузер, которым клиент пользуется для выдачи запроса
HTTP_REFERER	URL-адрес документа, на который клиент указывал перед обращением к CGI-программе

Предупреждение

Имена переменных среды CGI на разных Web-серверах могут различаться. Следует обратиться к документации на соответствующий сервер.

CGI-программа на языке Perl имеет доступ к переменным среды через автоматически создаваемый интерпретатором ассоциативный массив `%ENV`, к элементам которого можно обратиться по ключу, совпадающему с именем переменной среды. Ниже приведен пример CGI-сценария, формирующего HTML-документ с информацией о всех установленных переменных среды, и отображение этого документа в окне браузера.

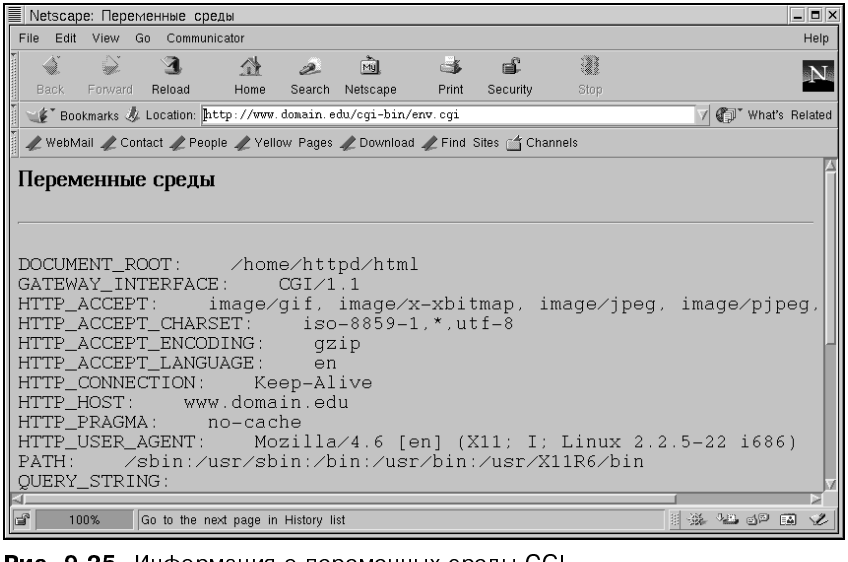


Рис. 9.25. Информация о переменных среды CGI

Пример 9.6. Получение информации о переменных среды CGI

```
#!/usr/bin/perl
print "Content-type:text/html\n\n";
```

```
print "<html>\n";
print "<head><title>Переменные среды</title></head>\n";
print "<body><h2>Переменные среды</h2>\n";
print "<hr><pre>\n";
foreach $name (sort(keys %ENV))
{
       print "$name:    $ENV{$name}\n";
}
print "<hr></pre>\n";
print "</body></html>\n";
```

Поиск и замена текста. Регулярные выражения

При обработке текста часто приходится осуществлять поиск фрагментов, удовлетворяющих некоторым условиям, например,

- найти все слова, содержащие последовательность `qwerty`;
- найти все слова, начинающиеся с последовательности `qwerty`;
- найти все слова, имеющие окончание `qwerty`;
- найти все слова, не содержащие последовательности `qwerty`;
- найти все слова, начинающиеся с последовательности `qwerty` и оканчивающиеся цифрой, и т. д.

Как видим, условия поиска могут быть достаточно сложными. В языке Perl существуют специальные конструкции, предписывающие интерпретатору организовать поиск, замену или преобразование фрагментов текста, удовлетворяющих некоторому образцу. Рассмотрим следующее выражение:

```
$variable =~ s/pattern/substitution/[e] [g] [i] [m] [o] [s] [x]
```

Левая часть `$variable` является скалярной переменной. Правая часть представляет собой конструкцию

s/*pattern*/*substitution*/[e] [g] [i] [m] [o] [s] [x],

которая служит для интерпретатора Perl командой, организующей поиск образца *pattern* и замену его выражением *substitution*. Опции `egimosx` модифицируют процесс поиска и замены, в частности, опция "`g`" означает выполнение глобальной (а не единичной) замены, а опция "`e`" означает, что заменяющая строка должна вычисляться как Perl-выражение. Оператор "`=~`" связывает скалярную переменную с командой подстановки, указывая, что замену следует осуществлять в строке `$variable`. Значением данного выражения является количество произведенных замен, так что допустима следующая операция присваивания:

```
$count=($variable=~s/pattern/substitution/),
```

в результате которой переменная `$count` будет равна количеству замен, сделанных в строке `$variable`.

Выражение:

```
$variable=~tr/searchlist/replacementlist/cds
```

предписывает интерпретатору заменить в строке `$variable` все вхождения символов из списка `searchlist` на соответствующие символы из списка `replacementlist`. Значением выражения является число замененных символов. Опции модифицируют процесс замены следующим образом:

`c` — будет использован не список поиска `searchlist`, а его дополнение;

`d` — удаление всех символов, для которых нет соответствующего символа в списке замены `replacementlist`;

`s` — все последовательности символов, преобразуемые в один и тот же символ, заменяются одним экземпляром этого символа.

Образец для поиска `pattern` представляется в виде *регулярного выражения*. Регулярное выражение можно рассматривать как шаблон для выделения цепочек символов. Для задания этого шаблона используются специальные символы \ | () [{ ^ $ * + ? . и двухсимвольные конструкции, имеющие в составе регулярного выражения особый смысл. Все остальные символы являются обычными символами, представляющими самих себя. Подробно регулярные выражения рассматриваются в литературе по операционной системе UNIX, где различные их формы используются в командном интерпретаторе shell, утилитах grep, sed, awk, текстовых редакторах ed, vi, emacs. Ограничимся перечислением некоторых конструкций, имеющих в регулярном выражении специальную интерпретацию (табл. 9.17).

Таблица 9.17. *Конструкции, используемые в регулярных выражениях*

Конструкция	Интерпретация
.	Любой одиночный символ, кроме символа новой строки
[...]	Любой символ из [...]; возможно задание диапазона значений, например, [A-Z] означает любой символ от A до Z
[^...]	Любой символ, не входящий в [...]
(...)	Ряд элементов, сгруппированных как один элемент
(...|...|...)	Выбор одного из вариантов
^	Обозначает положение в начале строки
$	Обозначает положение в конце строки
\	Отмена специального значения следующего за ним символа

Таблица 9.17 *(окончание)*

Конструкция	Интерпретация
+	Одно или более повторений предыдущего элемента
*	Нуль или более повторений предыдущего элемента
?	Нуль или одно повторение предыдущего элемента
{n,m}	Минимум n, максимум m повторений предыдущего элемента; {n} означает ровно n повторений, {n,} — минимум n повторений
\w	Множество алфавитно-цифровых символов плюс символ подчеркивания, эквивалентно [a-zA-Z_0-9]
\W	Дополнение предыдущего множества, т. е. [^a-zA-Z_0-9]
\s	Разделитель: пробел, символ табуляции (\t), символ новой строки (\n), символ возврата каретки (\r), символ подачи бланка (\f)
\S	Не разделитель
\b	Обозначает положение на границе слова
\B	Обозначает положение не на границе слова

Часть регулярного выражения может быть заключена в круглые скобки. В этом случае все заключенные в скобки подвыражения будут пронумерованы слева направо числами 1, 2, и т. д. При нахождении фрагмента, удовлетворяющего образцу, в нем будут выделены части, соответствующие подвыражениям, заключенным в круглые скобки, и сохранены в специальных переменных. К этим переменным в дальнейшем можно обратиться внутри регулярного выражения (при помощи ссылок \1, \2 и т. д.) или за пределами регулярного выражения (при помощи ссылок $1, $2 и т. д.).

Примеры:

- `$str =~ s/\bred\b/white/g;`

  Глобальная замена целого слова `red` на слово `white`. Части слов замене не подлежат. Слово *redactor*, например, останется неизменным.

- `$count=($x =~ s/\bmore\b/less/g);`

  Подсчет числа замен слова `more` словом `less`.

- `$str =~ s/([^ ]*) *([^ ]*)/$2 $1/;`

  Поменять местами первые два поля в строке `$str`.

- `$str =~ s/^\s*(.*?)\s*$/$1/;`

  Убрать символы-ограничители в начале и конце строки `$str`.

- `($b=$a)=~s/Perl/Shell/;`

  Присвоить переменной `$b` значение переменной `$a`, заменив слово `Perl` словом `Shell`.

Обработка данных формы

Данные формы поступают в CGI-программу в закодированном виде, поэтому в качестве первого шага обработки CGI-сценарий должен выполнить декодирование полученной информации. При пересылке данных методом `GET` данные формы присваиваются переменной среды `QUERY_STRING`, при передаче методом `POST` — передаются в программу через стандартный ввод и тоже могут быть присвоены некоторой внутренней переменной. Таким образом, декодирование данных сводится к следующей последовательности манипуляций со строкой:

- замена каждой группы `%hh`, состоящей из шестнадцатеричного ASCII-кода `hh` с префиксом `%`, на соответствующий ASCII-символ;
- замена символов `+` пробелами;
- выделение отдельных пар *имя=значение*, разделенных ограничителем `&`;
- выделение из каждой пары *имя=значение* имени и значения соответствующего поля формы.

Рассмотренные выше средства языка Perl позволяют написать следующую короткую программу декодирования

```
#!/usr/bin/perl
# Декодирование данных формы, переданных методом GET
$form_data = $ENV{'QUERY_STRING'};
# преобразование цепочек %hh в соответствующие символы
$form_data =~ s/%(..)/pack ("C", hex ($1))/eg;
# преобразование плюсов в пробелы
$form_data =~ tr/+/ /;
# разбиение на пары имя=значение
@pairs = split (/&/, $form_data);
# выделение из каждой пары имени и значения поля формы и сохранение
# их в ассоциативном массиве $form_fields
foreach $pair (@pairs)
{
        ($name, $value)=split(/=/,$pair);
        $form_fields{$name}=$value;
}
```

Если данные формы переданы методом `POST`, то в приведенном тексте следует заменить оператор присваивания

```
$form_data = $ENV{'QUERY_STRING'};
```

оператором

```
read(STDIN,$form_data,$ENV{'CONTENT_LENGTH'});
```

считывающим из стандартного ввода программы `CONTENT_LENGTH` байтов, составляющих содержимое запроса клиента, в переменную `$form_data`.

В приведенном примере используются три новые функции: `split`, `pack` и `hex`. Поясним их назначение прежде, чем перейти к обсуждению текста программы.

- Функция `split /pattern/, string`

  разбивает строку *string* на подстроки, используя в качестве разделителя регулярное выражение *pattern*.

- Функция `pack template, list`

  упаковывает список значений *list* в двоичную структуру по заданному шаблону *template*.

  Аргумент *template* представляет собой последовательность символов, определяющих формат представления пакуемых данных:

a/A	Текстовая строка, заполненная нулями/пробелами
b/B	Двоичная строка, значения расположены в порядке возрастания/убывания
c/C	Обычное символьное значение/ Символьное значение без знака
f/d	Значение в формате с плавающей точкой одинарной/двойной точности
h/H	Шестнадцатеричная строка, младший/старший полубайт первый
i/I	Целое со знаком/ без знака
l/L	Значение типа long со знаком/без знака
n/N	Значение типа short/long с "сетевым" порядком байтов ("старший в старшем")
p/u	Указатель на строку/Uu-кодированная строка
s/S	Значение типа short со знаком/без знака
v/V	Значение типа short/long с VAX-порядком байтов ("старший в младшем")
x/X	Нулевой байт/резервная копия байта
@	Заполнение нулевыми байтами (до абсолютной позиции)

  За каждым символом может следовать число, обозначающее счетчик применений данного символа в качестве формата. Символ * в качестве счетчика означает применение данного формата для оставшейся части списка.

  Примеры:

```
$x = pack "cccc", 80, 101, 114, 108;
$x = pack "c4", 80, 101, 114, 108;
$x = pack "B32", "01010000011001010111001001101100";
$x = pack "H8", "5065726C";
$x = pack "H*", "5065726C";
$x = pack "cB8H2c",80,"01100101",72,108;
```

  Значение переменной `$x` во всех случаях равно "`Perl`".

- Функция `hex expr`

  Интерпретирует аргумент *expr* как шестнадцатеричную строку и возвращает ее десятичное значение.

В приведенном выше тексте программы все представляется очевидным. Прокомментируем только наиболее насыщенную строку:

```
$form_data =~ s/%(..)/pack ("C", hex ($1))/eg;
```

Образец для поиска задан в виде регулярного выражения `%(..)`. Этому образцу удовлетворяет произвольная последовательность вида `%xy`, где `x`, `y` — любые символы. В результате кодирования данных в качестве `x`, `y` могут появиться только шестнадцатеричные цифры, поэтому можно не задавать более точный, но менее компактный шаблон `%([0-9A-Fa-f][0-9A-Fa-f])`. Часть выражения заключена в скобки `(..)`. При нахождении подходящего фрагмента `%hh` его часть, содержащая шестнадцатеричное число `hh`, сохраняется в переменной, которая затем будет использована в качестве аргумента функции `hex($1)` для преобразования в десятичное значение. Функция `pack` упакует это десятичное значение в двоичную структуру, которая в соответствии с шаблоном `"C"` будет интерпретироваться как символ. Этот символ заменяет в тексте найденную цепочку `%hh`.

После выделения и декодирования данных можно приступить к их обработке. Попробуем написать CGI-сценарий, обрабатывающий данные формы из примера 9.5. Программа должна декодировать полученные данные, проверять заполнение обязательных полей формы и правильность подтверждения пароля, в зависимости от результатов проверки формировать документ для отсылки клиенту. Сохраним сценарий в файле `/cgi-bin/registrar.cgi`. Полный маршрут к данному файлу определяется параметрами конфигурации Web-сервера. Местоположение каталога `cgi-bin` обычно указывается не относительно корневого каталога файловой системы, а относительно корня дерева документов Web-сервера. Например, если корнем является каталог `/home/httpd/html/`, то файл сценария будет иметь маршрутное имя `/home/httpd/html/cgi-bin/registrar.cgi`, которое в запросе клиента будет указано как `/cgi-bin/registrar.cgi`. В первом приближении текст сценария может выглядеть следующим образом.

Пример 9.7. Первый вариант сценария

```
#!/usr/bin/perl
print "Content-type:text/html\n\n";
$method = $ENV{'REQUEST_METHOD'};
if ($method eq "GET") {
    $form_data = $ENV{'QUERY_STRING'};
}
```

```
else  {
    read (STDIN, $form_data, $ENV{'CONTENT_LENGTH'});
}
$form_data =~ s/%(..)/pack ("C", hex ($1))/eg;
$form_data =~ tr/+/ /;
@pairs = split (/&/, $form_data);
foreach $pair (@pairs) {
    ($name, $value)=split(/=/,$pair);
    $FORM{$name}=$value;
}
#Проверка заполнения обязательных полей
if (!$FORM{'regname'} || !$FORM{'password1'}) {
print<<goback
     <html>
     <head><title>Неполные данные</title></head>
     <body><h2>Извините, Вы пропустили обязательные данные</h2>
     <br>
     <a href="http://www.klf.ru/welcome.html">Попробуйте еще раз,
     пожалуйста</a>
     </body>
     </html>
goback
;}
#Проверка правильности ввода пароля
elsif ($FORM{'password1'} eq $FORM{'password2'}){
print<<confirmation
     <html>
     <head><title>Поздравляем!</title></head>
     <body><h2>Поздравляем!</h2><br>
     Ваша регистрация прошла успешно. Вы можете пользоваться нашей
     библиотекой. Спасибо за внимание.
     </body>
     </html>
confirmation
;}
else {
print<<new_form
    <html><head><title>Ошибка при вводе пароля</title></head>
    <body><h3>Введенные Вами значения пароля не совпадают
    <br><form method="get" action="/cgi-bin/registrar.cgi">
    <pre>
    Введите пароль:    <input type="password" name="password1">
    Подтвердите пароль:    <input type="password" name="password2">
    </pre>
new_form
;
```

```
foreach $key ( keys %FORM) {
        if ($key ne "password1" && $key ne "password2") {
        print "<input type=\"hidden\" name=$key value=$FORM{$key}>\n";
        }
}
print<<EndOfHTML
        <br><br>
        <input type="submit" value="OK"> <input type="reset"
value="Отменить">
        </form></body></html>
EndOfHTML
;}
```

После вывода строки заголовка осуществляется считывание переданной серверу информации в переменную `$form_data`. В зависимости от метода передачи эта информация считывается из переменной среды `QUERY_STRING` (метод `GET`) или из стандартного ввода программы (метод `POST`). Для выбора одного из вариантов здесь использован условный оператор, общая форма которого имеет следующий вид:

```
if (выражение) {...} [[ elsif (выражение) {...} ...] else {...}]
```

Если значение выражения истинно, выполняется соответствующий блок операторов, заключенных в фигурные скобки. Ветви `elsif` и `else` являются необязательными.

Считанная информация декодируется и помещается в ассоциативный массив `%FORM`.

Отсутствие обязательных данных — регистрационного имени и пароля — проверяется с помощью условия `if (!$FORM{'regname'} || !$FORM{'password1'})`, смысл которого становится понятен, если вспомнить замечание о том, как интерпретируются значения скалярных величин в условных выражениях. В случае отсутствия необходимых данных формируется виртуальный HTML-документ, предлагающий повторить попытку, который и посылается клиенту (рис. 9.26)

При выводе этого документа в операции `print` использована конструкция, заимствованная из командного интерпретатора UNIX shell и называемая "here document" ("документ здесь"): текст, заключенный между `<<word` и следующим вхождением идентификатора *word*, расположенного в отдельной строке, трактуется как заключенный в двойные кавычки. Такая конструкция позволяет внутри себя использовать символы, которые при заключении в обычные двойные кавычки необходимо маскировать символом "\", например, сами двойные кавычки ", символы `@`, `$`, `%`. Важно помнить, что между символом `<<` и идентификатором *word* не должно быть пробела, а закрывающий конструкцию идентификатор *word* должен располагаться в отдельной строке.

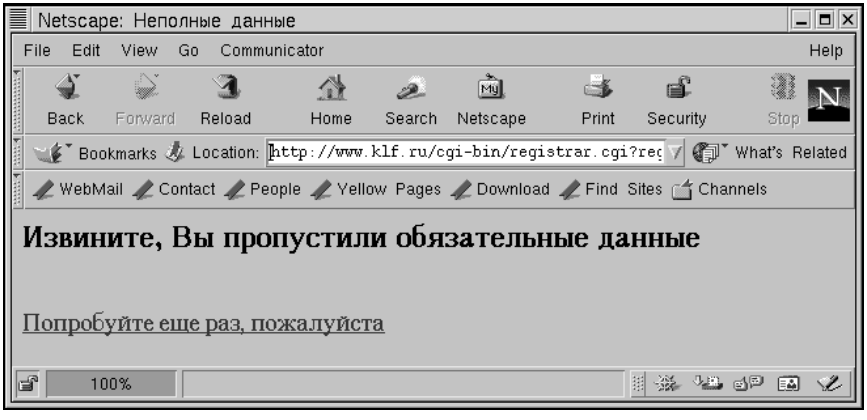


Рис. 9.26. Ответ сервера в случае отсутствия обязательной информации

Условие `elsif ($FORM{'password1'} eq $FORM{'password2'})` предназначено для проверки совпадения двух копий введенного пользователем пароля. Если значения совпадают, то пользователю посылается сообщение, подтверждающее успешную регистрацию (рис. 9.27).

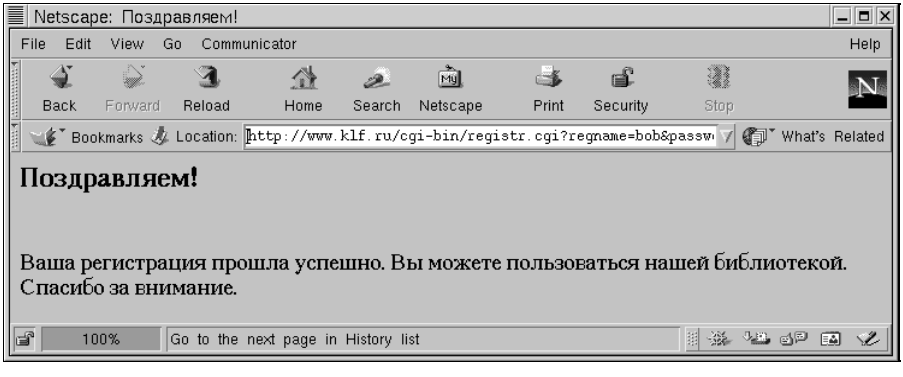


Рис. 9.27. Подтверждение регистрации

В противном случае формируется HTML-документ, предлагающий ввести пароль повторно (рис. 9.28). Этот новый документ содержит форму, в состав которой входят два видимых поля типа `"password"` — для ввода и подтверждения пароля, и скрытые поля типа `"hidden"` — для сохранения остальных данных, введенных при заполнении исходной формы. Каждое скрытое поле новой формы наследует у соответствующего поля исходной формы параметры `name` и `value`. Если эти данные не сохранить, то их придется вводить заново, принуждая пользователя повторно выполнять уже сделанную работу. Информация, сохраненная в скрытых полях, невидима пользователю и недоступна для изменения.

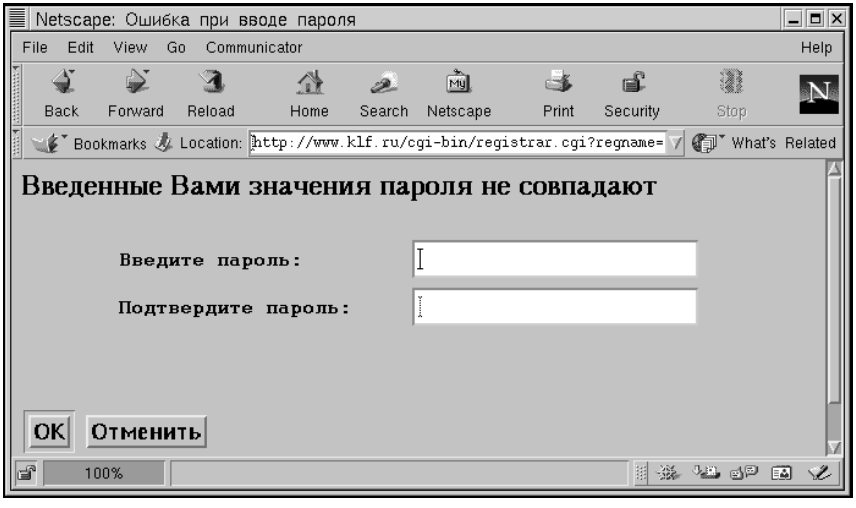


Рис. 9.28. Повторное приглашение для ввода пароля

Культура Perl допускает различные уровни владения языком. В рассмотренном варианте использован минимальный набор средств. Очевидно, что часть кода, например декодирование, требуется при обработке не только данной, но и любой другой формы. Естественным шагом в развитии исходного варианта сценария является выделение этой части в отдельную подпрограмму и предоставление доступа к ней другим сценариям.

Подпрограммы, библиотеки, модули

Подпрограммы в языке Perl играют ту же роль, что и функции в языке C, или процедуры и функции в языке Pascal. Они объединяют операторы в одну группу для дальнейшего использования. Подпрограмма может быть объявлена в любом месте основной программы при помощи описания

```
sub subroutine_name {…}
```

Последовательность операторов Perl, заключенная в фигурные скобки, называется *блоком*. Блоки применяются также в условных операторах и операторах цикла.

В языке Perl, как мы знаем, нет обязательного явного описания переменных. Точкой определения переменной является место, где она впервые встречается в программе, а признаком типа — первый символ ее имени.

Область действия большинства переменных (исключение составляют некоторые предопределенные глобальные переменные интерпретатора Perl) ограничена *пакетом*. Пакет это механизм, позволяющий создать свое пространство имен для некоторого отрезка программы (этот отрезок может включать всю программу). Каждый фрагмент кода Perl-программы относит-

ся к некоторому пакету. Пакет может быть задан по умолчанию или явно при помощи объявления

package *package_name*

Обычно описание пакета встречается в первой строке файлов специального назначения — библиотек и модулей, о чем пойдет речь ниже. Но может встречаться и в любом месте программы. Область пакета распространяется от оператора объявления до конца самого внутреннего вложенного блока или до объявления следующего пакета того же уровня. С каждым пакетом связана своя таблица символов, представляющая собой ассоциированный массив, имя которого совпадает с именем пакета. В ней перечислены все переменные, образующие пространство имен пакета. Каждый пакет имеет имя. Неявно предполагается, что основная программа, не имеющая явного определения пакета, связана с пакетом main. К переменным другого пакета можно обращаться, используя нотацию *имя_пакета*::*имя_переменной* с префиксом, соответствующим типу переменной, например:

$main::myname, @main::list.

Для того чтобы ограничить область действия переменных, внутри блока допускается их явное описание при помощи функций объявления local (Perl 4.0 и Perl 5.0) и my (Perl 5.0). Переменные, объявленные при помощи конструкции local, являются доступными только для операций внутри блока и для подпрограмм, вызываемых внутри блока. Переменные, объявленные при помощи конструкции my, являются доступными только для операций внутри блока и для подпрограмм, определенных в том же блоке, и недоступны для подпрограмм, определенных за пределами блока. Они не входят в таблицу символов пакета. В версии языка Perl 5.0 в большинстве случаев рекомендуется использовать функцию my. Если объявляются несколько переменных, то они должны быть заключены в скобки. Переменным при их объявлении могут быть присвоены начальные значения. Примеры объявления локальных переменных:

```
local $a;          # объявляется переменная $a
my $x="red";       # объявляется и инициализируется переменная $x
my ($x,$y)=(2,7);  # объявляются и инициализируются переменные $x,$y
```

В языке Perl не различаются понятия "подпрограмма" и "функция". Подпрограмма может быть использована в выражении как функция, возвращающая значение. По умолчанию таким значением подпрограммы является последнее вычисленное в ней выражение. Его можно задать и явно, указав в качестве аргумента функции return() в любой точке подпрограммы. Возвращаемое значение может быть скалярной величиной или массивом.

Вызов подпрограммы осуществляется по ее имени, которое может сопровождаться списком параметров в скобках или без скобок. Имя подпрограммы может иметь необязательный префикс "&", являющийся признаком типа:

```
имя_подпрограммы (список_параметров);
имя_подпрограммы список_параметров);
&имя_подпрограммы;
```

Для передачи в подпрограмму аргументы помещаются в специальный массив `@_`. Внутри подпрограммы они доступны как элементы `@_[0]`, `@_[1]`, и т. д. Такой механизм позволяет передавать в подпрограмму произвольное количество параметров.

В языках программирования различают передачу параметров *по ссылке* и *по значению*. При передаче параметров по значению подпрограмма получает копию переменной. Изменение копии внутри подпрограммы не влияет на ее оригинал. При передаче параметров по ссылке подпрограмма получает доступ к самой переменной и может ее изменять.

В языке Perl можно реализовать передачу параметров по значению, если внутри подпрограммы объявить локальные переменные и присвоить им значения аргументов из массива `@_`, как это сделано в следующем примере

```
#!/usr/bin/perl
# Передача в подпрограмму параметров по значению
$val=&sub1(9,11);
print "Значение (9+1) * (11-1) равно $val.\n";
$x = 9;
$y = 11;
$val = &sub1($x,$y);
print "Значение ($x+1) * ($y-1) равно $val.\n";
print "Значение \$x остается равным $x, а \$y равным $y.\n";
sub  sub1{
my($x, $y) = @_;
return (++$x * --$y);
}
```

Результат выполнения:

```
Значение (9+1) * (11-1) равно 100.
Значение (9+1) * (11-1) равно 100.
Значение $x остается равным 9, а $y равным 11.
```

Примечание

В примере использованы заимствованные из языка C префиксные операции увеличения на единицу ++ и уменьшения на единицу --.

Передача скалярных параметров по ссылке реализуется обращением внутри подпрограммы непосредственно к элементам массива `@_`. В этом случае фактический параметр должен быть переменной и не может быть константой, не способной изменять свое значение.

```
#!/usr/bin/perl
# Передача в подпрограмму параметров по значению
$x = 9;
$y = 11;
$val = &sub2($x,$y);
print "Значение ($x+1) * ($y-1) равно $val.\n";
print "Значение \$x стало равным $x, а \$y равным $y.\n";
sub  sub2{
return (++$x * --$y);
}
```

Результат выполнения:

```
Значение (10+1) * (10-1) равно 100.
Значение $x стало равным 10, а $y равным 10.
```

Передача по ссылке этим способом параметров-массивов тоже возможна, но с некоторыми ограничениями: во-первых, несколько параметров-массивов передаются как один массив в составе `@_`; во-вторых, внутри процедуры нельзя изменять размер массива, следовательно, использовать функции `push()` и `pop()`.

Для передачи по ссылке массивов можно воспользоваться другим методом, который заключается в том, чтобы внутри подпрограммы объявить локальные переменные, присвоив им значения аргументов из массива `@_`, а подпрограмме в качестве аргументов передать не массивы, а ссылки на них:

```
#!/usr/bin/perl
# Пример подпрограммы, использующей массивы, переданные по ссылке
&sub3(\@a1,\@a2);
print "Измененный массив \@a1: @a1\nИзмененный массив \@a2: @a2\n";
sub sub3{
       my($a1ref,$a2ref)=@_;
       print "Эта подпрограмма изменяет массивы \@a1 and \@a2\n";
       @$a1ref=("q","w","e");
       @$a2ref=("r","t","y");
}
```

Результат работы программы:

```
Эта подпрограмма изменяет массивы @a1 and @a2
Измененный массив @a1: q w e
Измененный массив @a2: r t y
```

Примечание

Обратите внимание на операцию разыменования, позволяющую обратиться к переменной через ссылку на нее:

```
@$a1ref=("q","w","e");
```

Обращение к индивидуальному элементу массива в этом случае может иметь одну из следующих эквивалентных форм:

```
$$a1ref  [0] = "q";
${$a1ref}[0] = "q";
$a1ref-> [0] = "q";
```

В последнем случае в начале выражения опущен символ $, который неявно подразумевается. Эту форму удобно применять в тех в случаях, когда ссылочное выражение является достаточно сложным.

Подпрограмма, объявленная внутри основной программы, может быть вызвана в любой ее точке. Чтобы сделать подпрограмму доступной другим программам, в языке Perl существуют два средства: библиотеки (Perl 4.0 и Perl 5.0) и модули (Perl 5.0).

Библиотека — это просто коллекция подпрограмм, которая обычно размещается в отдельном файле и образует отдельный пакет со своим пространством имен. Собственное пространство имен позволяет не смешивать одноименные переменные из библиотеки и основной программы.

Для создания библиотеки подпрограммы помещаются в один файл с расширением PL, начинающийся строкой объявления пакета `package` *`package_name`*. Имя файла должно совпадать с именем пакета. В конец файла надо добавить строку

```
1;
```

чтобы она возвращала значение ИСТИНА при включении пакета в основную программу при помощи функции `require`. Имя файла должно совпадать с именем пакета *`package_name`*. Пример библиотеки, состоящей из одной подпрограммы:

```
package cgi_utils;
sub print_header {
        print "Content-type: text/html\n\n";
}
1;
```

Библиотека `cgi_utils` состоит из одной подпрограммы `print_header` и сохраняется в файле с именем `cgi_utils.pl`.

Для использования библиотеки в вызывающей программе нужно указать имя библиотеки при помощи функции `require`:

```
#!/usr/bin/perl
require cgi_utils;
&cgi_utils::print_header;
```

Более развитым средством, чем библиотеки, являются модули в версии языка Perl 5.0. Модуль представляет собой библиотеку подпрограмм, обладаю-

щую дополнительными свойствами по сравнению с библиотеками Perl 4.0. Модуль позволяет управлять экспортом своих переменных и подпрограмм в другие программы, объявляя, какие из них экспортируются по умолчанию, а какие должны быть явно указаны в соответствующем операторе вызывающей программы. Для создания простого модуля следует поместить в начало файла строки следующего вида, за которыми должны следовать собственно подпрограммы:

```
package      package_name;
require      Exporter;
@ISA       = qw(Exporter);
@EXPORT    = qw(func1 func2);
@EXPORT_OK = qw($var @list %hash func3);
```

Имя файла должно совпадать с именем пакета *package_name* и иметь расширение PM. Первая из указанных строк является объявлением пакета. Предложение `require Exporter` предоставляет возможность наследовать подпрограмму `import`, реализованную в модуле `Exporter`. Механизм экспорта имен устроен таким образом, что каждый экспортирующий модуль должен иметь свою подпрограмму `import`, которая используется программой, импортирующей имена. Подпрограмма `import` должна быть определена в самом экспортирующем модуле или наследована у модуля `Exporter`. В третьей строке элементам специального массива `@ISA` присваиваются имена некоторых пакетов. Применение массива `@ISA` связано с новыми объектно-ориентированными возможностями Perl 5.0. Мы не будем их обсуждать, так как это выходит за рамки нашей темы.

Массив `@EXPORT` содержит имена, экспортируемые по умолчанию. В четвертой строке указывается, что из данного модуля по умолчанию будут экспортированы имена `func1` и `func2`. Конструкция

```
qw(string)
```

возвращает список слов в строке *string*. Ограничителями, служащими для выделения слов, являются символы пробела, табуляции, новой строки, возврата каретки, подачи бланка.

Массив `@EXPORT_OK` в пятой строке содержит имена, которые будут экспортироваться только в том случае, если они явно указаны в списке импорта вызывающей программы.

В вызывающей программе для обращения к модулю используется функция

```
use Module [list]
```

Здесь *Module* — имя модуля, а *list* — необязательный список аргументов, позволяющий управлять импортом имен. Если аргументы отсутствуют, то будут импортироваться все имена, определенные в модуле в массиве `@EXPORT`. Для того чтобы импортировались имена, содержащиеся в массиве

@EXPORT_OK, они должны быть явно указаны в списке аргументов *list*. Если представлен пустой список аргументов (), символы вообще не импортируются.

Преобразуем рассмотренную выше в качестве примера библиотеку в модуль. Для этого создадим файл cgi_utils.pm, в который запишем следующие строки:

```
package cgi_utils;
require      Exporter;
@ISA       = qw(Exporter);
@EXPORT    = qw(print_header);
sub print_header {
        print "Content-type: text/html\n\n";
}
1;
```

Соответствующим образом преобразуем вызывающую программу:

```
#!/usr/bin/perl
use cgi_utils;
&print_header;
```

Используем методы, изложенные в этом разделе, для исправления замечаний, сделанных относительно CGI-сценария из примера 9.7.

Пример 9.8. Структурированный вариант сценария

а) Часть исходного кода может быть использована другими CGI-программами. Преобразуем ее в отдельный модуль, сохраняемый в файле cgi_utils.pm.

```
package cgi_utils;
require Exporter;
@ISA = qw(Exporter);
@EXPORT = qw(print_header process_input);

# Подпрограмма вывода заголовка ответа
sub print_header {
       print "Content-type: text/html\n\n";
}

# Подпрограмма декодирования данных формы
sub process_input {
       my ($form_ref)=@_;
       my ($form_data,@pairs);
       my ($temp)="";
```

```
        if ($ENV{'REQUEST_METHOD'} eq 'POST') {
                read(STDIN,$form_data,$ENV{'CONTENT_LENGTH'});
        }
        else {
                $form_data=$ENV{'QUERY_STRING'};
        }
        $form_data=~s/%(..)/pack("c",hex($1))/ge;
        $form_data=~tr/+/ /;
        $form_data=~s/\n/\0/g;
        @pairs=split(/&/,$form_data);
        foreach $item(@pairs) {
                ($name,$value)=split (/=/,$item);
                if (!defined($form_ref->{$name})) {
                        $form_ref->{$name}=$value;
                }
                else {
                        $form_ref->{$name} .= "\0$value";
                }
        }
        foreach $item (sort keys %$form_ref) {
        $temp.=$item."=".$form_ref->{$item}."&";}
        return($temp);
}
1;
```

б) Текст основного сценария обработки формы registrar.cgi преобразуем следующим образом:

```
#!/usr/bin/perl
use cgi_utils;
my %FORM, $file_rec;
$file_rec=&process_input(\%FORM);
#Проверка заполнения обязательных полей
#if ($FORM{'regname'} eq "" || $FORM{'password1'} eq "") {
if (!$FORM{'regname'} || !$FORM{'password1'}) {
        print "Location: /goback.html\n\n";
}
#Проверка правильности ввода пароля
elsif ($FORM{'password1'} eq $FORM{'password2'}){
        print "Location: /confirmation.html\n\n";
        open (OUTF, ">>users");
        print OUTF $file_rec, "\n";
        close OUTF
}
else {
&print_header;
```

```
print<<new_form
       <html>
       <head><title>Ошибка при вводе пароля</title></head>
       <body><h3>Введенные Вами значения пароля не совпадают
       <br><form method="get" action="/cgi-bin/registrar.cgi">
       <pre>
       Введите пароль:       <input type="password" name="password1">
       Подтвердите пароль:   <input type="password" name="password2">
       </pre>
new_form
;
foreach $key ( keys %FORM) {
       if ($key ne "password1" && $key ne "password2") {
       print "<input type=\"hidden\" name=$key value=$FORM{$key}>\n";
       }
}
print<<EndOfHTML
       <br><br>
       <input type="submit" value="OK">
       <input type="reset" value="Отменить">
       </form>
       </body>
       </html>
EndOfHTML
;
}
exit
```

в) В исходном варианте сценария в качестве ответов сервера при получении неполных данных и для подтверждения регистрации пользователя формируются виртуальные HTML-документы. В этом нет необходимости, так как они содержат только статическую информацию. Соответствующие фрагменты сценария преобразуем в HTML-код готовых документов, которые сохраним в отдельных файлах. В основном сценарии в качестве ответа сервера возвращаются ссылки на эти документы.

Файл confirmation.html содержит документ, посылаемый клиенту в качестве сообщения об успешной регистрации:

```
<html>
<head><title>Поздравляем!</title></head>
<body><h2>Поздравляем!</h2><br>
Ваша регистрация прошла успешно. Вы можете пользоваться нашей библиоте-
кой.
<br>
Спасибо за внимание.
</body>
</html>
```

Файл goback.html содержит документ, посылаемый клиенту при получении неполных данных:

```
<html>
<head><title>Неполные данные</title></head>
<body><h2>Извините, Вы пропустили обязательные данные</h2>
<br>
<a href="http://www.klf.ru/welcome.html">Попробуйте еще раз, пожалуй-
ста</a>
</body>
</html>
```

В приведенном тексте появились некоторые новые элементы, которые необходимо пояснить.

Подпрограмма `process_input` модуля `cgi_utils.pm` передает декодированные данные через вызываемый по ссылке параметр — ассоциативный массив. Кроме того, она возвращает при помощи функции `return()` те же данные, но в виде строки, состоящей из пар `имя=значение`, разделенных символом `&`. Обратите внимание на то, как подпрограмма вызывается в основной программе:

```
$file_rec=&process_input(\%FORM);
```

В качестве аргумента ей передается ссылка на ассоциативный массив. В тексте подпрограммы появилась проверка наличия полей формы с совпадающими именами и разными значениями

```
if (!defined($form_ref->{$name})) {
       $form_ref->{$name}=$value;
}
else {
}}
```

Этот фрагмент необходим для того, чтобы правильно обработать следующую ситуацию из нашего примера. Выбраны несколько переключателей, определяющих языки, которыми владеет пользователь: русский, английский, французский. Так как соответствующие элементы формы имеют одинаковые имена `name=language`, то без проверки в ассоциативный массив `%form_ref`, куда помещаются обработанные данные, попадет только информация от последнего обработанного элемента `name=language value=french`. В подобном случае обычное присваивание заменяется операцией присваивания с конкатенацией

```
$form_ref->{$name} .= "\0$value",
```

которая к переменной `$form_ref->{$name}` добавляет нулевой символ и значение `$value`.

В основной программе `registrar.cgi` обратим внимание на то, как передается ссылка на готовый HTML-документ. Для этого вместо заголовка `Content-type:text/html` выводится заголовок `Location: URL`, сообщающий серверу адрес документа.

Еще один новый элемент в основной программе — работа с файлами. В примере данные сохраняются в файле с именем `users`. Для записи информации в файл он открывается функцией `open` с дескриптором OUTF. Напомним, что дескриптор — это внутреннее, используемое системой имя, которое может быть ассоциировано с файлом на диске, устройством (в UNIX устройства также представлены специальными файлами) и другим объектом, к которому применимы файловые операции записи/считывания. Вызов функции

```
open (OUTF, ">>users");
```

создает дескриптор OUTF, открывает его для добавления данных и связывает с конкретным файлом `users`. Для указания операций допустимых над открытым файлом используются следующие обозначения:

```
open(filehandle, "filename");      # чтение из существующего файла
open(filehandle, "> filename ");   # открытие нового файла для записи
open(filehandle, ">> filename "); # открытие для записи в конец файла
```

Запись в файл осуществляется функцией `print` с указанием дескриптора файла в качестве первого аргумента:

```
print OUTF $file_rec, "\n";
```

После завершения работы с файлом его необходимо закрыть при помощи функции `close`.

ГЛАВА 10

Динамический HTML

Динамический HTML (Dynamic HTML или DHTML) не является каким-то особым языком разметки страниц. Это всего лишь термин, применяемый для обозначения HTML-страниц с динамически изменяемым содержимым.

Реализация DHTML покоится на трех "китах": непосредственно HTML, каскадных таблицах стилей (Cascade Style Sheets — CSS) и языке сценариев (JavaScript или VBScript). Эти три компонента DHTML связаны между собой объектной моделью документа (Document Object Model — DOM), являющейся, по сути, интерфейсом прикладного программирования (API). DOM связывает воедино три перечисленных компонента, придавая простому документу HTML новое качество, — возможность динамического изменения своего содержимого без перезагрузки страницы. Символически подобное единство показано на рис. 10.1.

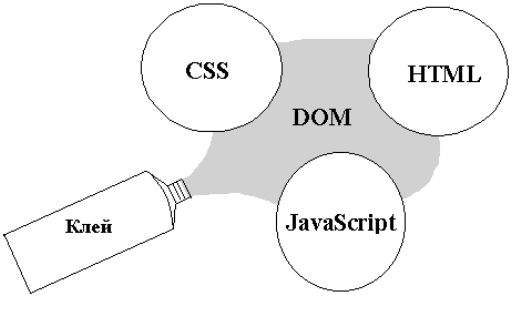


Рис. 10.1. Компоненты динамического HTML

Каскадные таблицы стилей можно сравнить со стилевыми файлами любого текстового редактора. С их помощью определяется внешний вид отображаемого HTML-документа: цвет шрифта и фона документа, сам шрифт, разбивка текста и многое другое. Для каждого элемента, задаваемого определенным тэгом HTML, можно определить свой стиль отображения в окне браузера. Например, заголовки первого уровня будут отображаться шрифтом Arial 16pt синего цвета, заголовки второго уровня — Arial 14pt красного цве-

та, основной текст — Times New Roman 10pt черного цвета с одинарным интервалом между строками. Можно создать таблицу стилей и использовать ее для всех документов, расположенных на сервере, что придаст стройность и строгость всему Web-сайту.

Объектная модель документа делает все элементы страницы программируемыми объектами. С ее помощью через языки сценариев можно получить доступ и управлять всем, что есть в документе. Каждый элемент HTML доступен как индивидуальный объект, а это означает, что можно изменять значение любого параметра любого тэга HTML-страницы, и, как следствие, документ действительно становится динамическим. Любое действие пользователя (щелчок кнопкой мыши, перемещение мыши в окне браузера или нажатие клавиши клавиатуры) объектной моделью документа трактуется как событие, которое может быть перехвачено и обработано процедурой сценария.

DHTML достаточно новая технология, и не все браузеры поддерживают объектную модель документа и каскадные таблицы стилей. Однако DHTML использует стандартные тэги HTML, и поэтому пользователи браузеров, не поддерживающих DOM, практически увидят все, что задумано разработчиком динамической страницы, но только в статическом виде.

Есть еще одна "неприятность", связанная с тем, что разные фирмы-разработчики браузеров могут реализовывать собственную объектную модель документов, как это произошло с двумя популярными браузерами Internet Explorer и Netscape Navigator. Поэтому разработчикам динамических страниц приходится, в конечном счете, писать два варианта своих приложений, чтобы пользователи указанных браузеров могли правильно просматривать их страницы.

В данной главе описываются каскадные таблицы стилей, объектные модели документов браузеров Internet Explorer и Netscape Navigator и приемы создания динамических HTML-страниц.

Каскадные таблицы стилей

Каскадные таблицы стилей впервые были реализованы в Internet Explorer 3.0, но информация о них в то время была большей частью противоречивой. При реализации Internet Explorer 4.0 были приняты во внимание рекомендации REC-CSS1 Консорциума W3 относительно каскадных таблиц стилей, датированные 17 декабря 1996 года. К настоящему времени они пересмотрены и известны как рекомендации по каскадным таблицам стилей, уровень 1, документ REC-CSS1-19990111 от 11 января 1999 года. В мае 1998 года Консорциум издал рекомендации по каскадным таблицам стилей, уровень 2, документ REC-CSS2-19980512, часть из которых реализована в Internet Explorer 4.01. В соответствии с этими рекомендациями и будет вестись описание каскадных таблиц стилей.

Общие положения

Каскадные таблицы стилей, уровень 1, представляют собой простую технологию определения и присоединения стилей к документам HTML. *Стиль*, говоря житейским языком, — это все то, что определяет внешний вид документа HTML при его отображении в окне браузера: шрифты и цвета заголовков разных уровней, шрифт и разрядка основного текста, задаваемого в тэге абзаца `<P>`, и т. д. Стиль задается по определенным правилам, о которых, собственно говоря, и пойдет речь в данном разделе, а таблица стилей — набор правил отображения, применяемых в документе, к которому присоединена соответствующая таблица стилей.

Таблица стилей — это шаблон, который управляет форматированием тэгов HTML в Web-документе. Если читатель работал с текстовым редактором Microsoft Word, то концепция таблицы стилей напомнит ему концепцию стилевых файлов этого редактора: изменить внешний вид документа Word можно простым изменением присоединенных к нему стилей. Точно также изменить внешний вид документа HTML можно простым изменением присоединенной к нему таблицы стилей.

Почему в название таблиц стилей включено определение "каскадные"? Дело в том, что рекомендации Консорциума W3 позволяют использовать несколько таблиц стилей для управления форматированием одного документа HTML, а браузер по определенным правилам выстраивает приоритетность применения этих таблиц. Они выстраиваются неким "каскадом", по которому и "прокатывается" документ. Правила приоритетности и разрешения возникающих конфликтов описаны ниже в данном разделе.

Для разработки таблицы стилей достаточно немного ориентироваться в языке HTML и быть знакомым с базовой терминологией настольных издательских систем. Как отмечалось выше, таблица стилей представляет собой набор правил форматирования элементов HTML. Эти правила достаточно просты и легко запоминаемы. Например, если необходимо, чтобы в документе все заголовки первого уровня отображались синим цветом и шрифтом с кеглем (размером) 16 пунктов, то в таблице следует задать правило:

```
H1 {color: blue;
    font-size: 16pt}
```

Любое *правило каскадных таблиц* стилей состоит из двух частей: *селектора* и *определения*. Селектором может быть любой тэг HTML, для которого определение задает, каким образом необходимо его форматировать. Само определение, в свою очередь, также состоит из двух частей: *свойства* и его *значения*, разделенных знаком двоеточия (`:`). Назначение свойства очевидно из его названия. В приведенном правиле селектором является элемент `H1`, а определение, записанное в фигурных скобках, задает значения двух свойств заголовка первого уровня: цвет шрифта (свойство `color`) определен как синий

(значение `blue`) и размер шрифта (свойство `font-size`) определен в 16 пунктов (значение `16pt`). В одном правиле можно задать несколько определений, разделенных символом точка с запятой (`;`), как это демонстрируется в приведенном примере.

Созданная только что таблица стилей влияет на форматирование элемента определенного типа: заголовок первого уровня. Ее комбинация с другими таблицами стилей определяет окончательное представление документа при его просмотре в окне браузера.

Предупреждение

Синтаксис правил каскадных таблиц стилей не чувствителен к регистру. Селекторы, свойства и их значения можно задавать как строчными, так и прописными буквами, или в смешанном порядке. Однако каскадные таблицы стилей чувствительны к синтаксису задания правил и правильности названий свойств, значений и селекторов. Любая грамматическая ошибка приводит к тому, что правило пропускается анализатором браузера, и никакого предупреждающего сообщения не появляется.

Авторы страниц HTML должны писать свои таблицы стилей, только если они хотят придать документу вид, отличный от вида, предоставляемого умалчиваемой таблицей стилей браузера.

Встраивание таблиц стилей в документ

Чтобы таблица стилей могла воздействовать на внешнее представление документа, браузер должен знать о ее существовании. Для этого ее необходимо связать с HTML-документом.

Существует четыре способа связывания документа и таблицы стилей:

1. Связывание — позволяет использовать одну таблицу стилей для форматирования многих страниц HTML.
2. Внедрение — позволяет задавать все правила таблицы стилей непосредственно в самом документе.
3. Импортирование — позволяет встраивать в документ таблицу стилей, расположенную на сервере.
4. Встраивание в тэги документа — позволяет изменять форматирование конкретных элементов страницы.

Связывание позволяет хранить таблицу стилей в отдельном файле и присоединять ее к документам с помощью тэга `<LINK>`, задаваемого в разделе `<HEAD>`:

```
<LINK REL="stylesheet" TYPE="text/css" HREF="mystyles.css">
```

Связываемый файл содержит набор правил каскадных таблицей стилей, определяющих форматирование документа, и должен иметь расширение CSS.

Связывание позволяет разработчику применить одинаковый набор правил форматирования к группе HTML-документов, что приводит к единообразному отображению различных документов и придает некоторую системность серверу разработчика.

При *внедрении* таблицы стилей в документ правила, ее составляющие, задаются в стилевом блоке, ограниченном тэгами `<STYLE TYPE="text/css">` и `</STYLE>`, который должен размещаться в разделе `<HEAD>` документа:

```
<HEAD>
<STYLE TYPE="text/css">
<!--
       B { text-transform: uppercase; }
       P { background-color: lightgrey;
           text-align:center; }
-->
</STYLE>
</HEAD>
```

Обычно браузеры, не поддерживающие какие-либо тэги, игнорируют их, интерпретируя, однако, их содержимое в том виде, в каком оно задано, что может приводить к ошибкам. Поэтому, как обычно, следует задавать содержимое тэгов, которые потенциально не обрабатываются старыми версиями браузеров, заключенным в тэг комментария `<!-- ... -->`.

В приведенном выше примере встроенная таблица стилей определяет отображение всех абзацев в документе (элемент `P`) на сером фоне с центрированными строками. Полужирный текст, определяемый любым элементом `B` (тэг `<B>`) документа, будет отображаться прописными буквами, даже если в документе он задан строчными.

В тэге `<STYLE>` можно *импортировать* внешнюю таблицу стилей с помощью свойства `@import` таблицы стилей:

```
@import: url(mystyles.css);
```

Его следует задавать в начале стилевого блока или связываемой таблицы стилей перед заданием остальных правил. Значением свойства `@import` является URL-адрес файла таблицы стилей.

Последний способ задания значений свойств таблицы стилей предназначен для оперативного форматирования определенного элемента документа и называется *внедрение*. Каждый тэг HTML имеет параметр `STYLE`, в котором можно задать значения его свойств в соответствии с синтаксисом каскадных таблиц стилей. Например, в следующем примере задается форматирование заголовка первого уровня, определяющее его отображение шрифтом красного цвета:

```
<H1 STYLE="color: red">Заголовок отображается шрифтом красного цвета</H1>
```

Если связанные, внедренные и импортируемые таблицы стилей влияют на форматирование всех элементов документа, для которых определены в таблицах правила, то встраивание определений стилей в конкретный тэг влияет на отображение только элемента, определяемого данным тэгом.

Совет

Рекомендуется избегать встраивания в тэги документа определений форматирования, так как подобная техника лишает разработчика преимуществ задания таблиц стилей в отдельном файле или в головной части документа, где их можно легко и быстро откорректировать, в случае необходимости. При форматировании отдельных тэгов придется просмотреть весь документ целиком, что требует достаточно большой и кропотливой работы.

Все способы встраивания таблиц стилей свободно сочетаются в одном документе. Например, можно разработать главную таблицу стилей для всех документов и связывать ее с каждым HTML-документом. Импортируемая или внедряемая таблица стилей будет уточнять форматирование элементов конкретного документа, а встраиваемые в тэг определения форматирования будут уточнять их отображение.

Группирование и наследование

Правила каскадных таблиц стилей состоят из селектора и определения. Для уменьшения размеров таблиц стилей можно *группировать* разные селекторы в виде списка элементов страницы HTML, разделенных запятыми, если для них задается одно правило. Например, следующие правила

```
H1 {font-family: Arial}
H2 {font-family: Arial}
H3 {font-family: Arial}
```

можно сгруппировать и задать в виде одного правила со списком селекторов

```
H1, H2, H3 {font-family: Arial}
```

Аналогично группируются определения, только в списке они разделяются точками с запятой (;). Следующие правила форматирования заголовка первого уровня

```
H1 {font-weight: bold}
H1 {font-size: 14pt}
H1 {font-family: Arial}
```

можно задать в виде одного правила, сгруппировав определения:

```
H1 {font-weight: bold;
    font-size: 14pt;
    font-family: Arial;
   }
```

Некоторые свойства имеют собственный синтаксис группирования, связанный с заданием значений нескольких свойств в одном. Например, предыдущий пример при использовании свойства `font` запишется так:

```
H1 {font: bold 14pt Arial}
```

При задании таблицы стилей можно свободно комбинировать все три правила группирования для уменьшения ее размеров.

В HTML некоторые элементы могут содержать другие. Как будет отображаться элемент, расположенный внутри другого элемента страницы, если для последнего задано правило форматирования, а для вложенного элемента нет? Например, пусть цвет шрифта абзаца определен как синий (`P {color: blue}`). Как будет отображаться выделенный элемент текста, задаваемый тэгом `<EM>`, если для него не определено правило форматирования? В подобных случаях вложенный элемент *наследует* правила форматирования элемента-родителя. В нашем примере выделенный элемент будет также отображаться синим цветом. Другие свойства ведут себя аналогично свойству `color`, например `font-family`, `font-size`.

Некоторые свойства не наследуются вложенными элементами от своих родителей, например свойство `background`, но по умолчанию вложенные элементы будут отображаться с фоном родительского элемента.

Наследование полезно при задании значений свойств, применяемых к документу по умолчанию. Для этого достаточно задать все свойства для элемента, порождающего все остальные элементы страницы HTML. Таким элементом является тело документа, определяемое тэгом `<BODY>`:

```
BODY {
       color: black;
       font-family: "Times New Roman";
       background: url(texture.gif) white;
  }
```

Приведенные правила задают форматирование документа по умолчанию: черным шрифтом гарнитуры `Times New Roman` с фоном, задаваемым графическим файлом `texture.gif`, или на белом фоне, если файл не доступен.

Примечание

Приведенное задание правил форматирования по умолчанию будет работать всегда, даже если разработчик пропустит в документе тэг `<BODY>`, что допускается стандартом языка HTML, так как синтаксический анализатор HTML всегда вставляет пропущенный тэг `<BODY>`.

Селекторы

Правила каскадных таблиц стилей, в которых в качестве селектора используются тэги HTML, влияют на отображение всех элементов заданного типа

в документе. Следующее правило отображает без подчеркивания *все* ссылки (тэг <A>) в документе:

```
<STYLE TYPE="text/css">
<!--
  A { text-decoration:none; }
-->
</STYLE>
```

А что делать, если необходимо некоторые ссылки отобразить по-другому? Можно задать для них правило форматирования непосредственно в тэге, а можно применить параметр CLASS, добавленный в HTML 4.0 в качестве стандарта для всех тэгов. Значением параметра CLASS является ссылка на класс, задаваемый в таблице стилей.

Селектор *CLASS*

Класс позволяет задать разные правила форматирования для одного элемента определенного типа или всех элементов документа. Имя класса указывается в селекторе правила после имени тэга и отделяется от него точкой. Можно определить несколько правил форматирования для одного элемента и с помощью параметра CLASS соответствующего тэга применять разные правила форматирования в документе. Например, можно определить два класса отображения заголовка первого уровня:

```
<STYLE TYPE="text/css">
<!--
  H1.red { color: red; }
  H1.blueBgrd { color: red; background-color: blue}
-->
</STYLE>
```

В тексте документа ссылка на соответствующий класс задается в параметре CLASS:

```
<H1 CLASS="red">Красный шрифт</H1>
<H1 CLASS="blueBgrd">Красный шрифт на синем фоне</H1>
```

Примечание

Имя класса в параметре CLASS задается без лидирующей точки. Оно может быть заключено в двойные или одинарные кавычки, или задаваться вообще без кавычек, например CLASS=red.

В приведенном примере классы задавались для разного отображения элементов одного типа. Если класс должен применяться ко всем элементам документа, то в селекторе задается имя класса с лидирующей точкой без указания конкретного элемента:

```
<STYLE TYPE="text/css">><!--
  .red { color: red; }
  .blueBgrd { color: red; background-color: blue}
-->></STYLE>
```

Теперь два класса red и blueBgrd можно применять к любым элементам документа:

```
<P ClASS=red>Первый абзац</P>
<P ClASS=blueBgrd>Второй абзац</P>
```

Первый абзац отобразится красным шрифтом, а второй — красным шрифтом на синем фоне.

Селектор *ID*

Параметр ID, как и параметр CLASS, не влияет на отображение браузером элемента HTML, но он задает уникальное имя элемента, которое используется для ссылок на него в сценариях и таблицах стилей. Параметр ID можно применять к любому элементу документа.

Правила таблиц стилей регламентируют использование уникального идентификационного имени элемента в качестве селектора, предваряя его символом #:

```
<STYLE TYPE="text/css">><!--
  #par24 { letter-spacing: 1em; }
  H1#form3 { color: red; background-color: blue}
-->></STYLE>
<BODY>
<P ID=par24>Разреженные слова в абзаце</P>
<H1 ID=form2>Черный шрифт</H1>
```

В этом примере абзац идентифицирован именем par24 в параметре ID, поэтому к нему применимо правило с селектором #par24. Второе правило в таблице стилей должно применяться к заголовку первого уровня с идентификатором form3. Такого элемента в нашем фрагменте нет, и поэтому заголовок form2 отображается с применением правила по умолчанию.

Совет

Для организации связи с правилами форматирования следует избегать использования идентификаторов конкретных элементов страницы, задаваемых в параметре ID. Если действительно необходимо отформатировать конкретный элемент, лучше использовать технику встраивания правил форматирования в тэг.

Контекстные селекторы

При разработке страниц HTML часто приходится одни элементы вкладывать в другие, например, выделять слова тэгом <EM> в каком-нибудь абзаце,

задаваемом тэгом `<P>`. В этом случае говорят, что элемент `P` порождает элемент `EM` и является его *предком*, а сам элемент `EM` является *потомком* элемента `P`. Некоторые свойства предка наследуются потомком, например, цвет шрифта (свойство `color`). Чтобы вложенные элементы отображались со своими значениями свойств, можно определить для них правила форматирования, как показано ниже:

```
<P> {color: blue}
<EM> {color: yellow}
```

Однако это приведет к тому, что *все* выделяемые в документе элементы будут отображаться шрифтом желтого цвета. А если необходимо, чтобы выделяемые только в абзаце элементы отображались желтым цветом, а в других частях документа каким-то другим цветом? Здесь помогут *контекстные селекторы*. Поставленную задачу решит следующее правило:

```
P EM {color: yellow}
```

Контекстный селектор состоит из нескольких простых, разделенных пробелами. Интерпретатор браузера просматривает в стеке все открытые элементы, находит элементы `EM`, порожденные элементом `P`, и применяет к ним указанное правило форматирования.

Таким образом, правила с контекстными селекторами задают исключения из общих правил форматирования элементов документа, определенных с простыми селекторами.

Псевдоклассы

Обычно правила форматирования присоединяются к элементу страницы, имеющему определенное положение в структуре документа. Концепция псевдоклассов и псевдоэлементов расширяет адресацию правил форматирования, позволяя внешней информации влиять на процесс форматирования.

Псевдоклассы и *псевдоэлементы* можно использовать в селекторах правил форматирования, однако реально в исходном тексте документа HTML они не существуют. Вернее, они как бы вставляются браузером при определенных условиях в документ, и на них можно ссылаться в таблицах стилей. Их называют *классами* и *элементами*, так как это удобный способ описания их поведения. Говоря точнее, их поведение определяется *фиктивной последовательностью тэгов*.

Псевдоэлементы применяются для адресации некоторых частей элементов, тогда как псевдоклассы позволяют таблицам стилей для элементов разных типов применять разные процедуры форматирования.

Примечание

В браузерах Internet Explorer и Netscape Navigator реализованы только псевдоклассы для элементов связей `A`. Псевдоэлементы в них не используются вообще.

Псевдоклассы связей

Связь в документе HTML определяется тэгом `<A>` с параметром `HREF`. Обычно браузеры отображают посещенные связи отлично от непосещенных (например, разными цветами). Уровень 1 каскадных таблиц стилей регламентирует правила для отображения связей через псевдоклассы элемента `A`:

```
A:link    { color: red }   /* непосещенная связь */
A:visited { color: blue }  /* посещенная связь */
A:active  { color: green } /* активная связь */
A:hover   { color: lime }  /* связь, на которой расположен курсор мыши */
```

Любую связь в документе можно отнести к одному из перечисленных классов. *Активная связь* — это связь, которая выбрана в данный момент.

Примечание

Комментарии в каскадных таблицах стилей задаются аналогично комментариям в языке C: текст, заключенный между символами `/*` и `*/`, является комментарием.

Примечание

Псевдокласс `hover` включен в каскадные таблицы стилей, уровень 2, который частично реализован в Internet Explorer 4.01.

В приведенном примере задается отображение непосещенных связей красным цветом, посещенных — синим, активных — зеленым. Если курсор мыши будет расположен над связью, то ее цвет изменится на ярко-зеленый.

Предупреждение

Браузер Internet Explorer 4.01 правильно реализует псевдоклассы связей, тогда как Netscape Navigator 4.0 не воспринимает псевдоклассы `active` и `hover`.

Так как псевдоклассы применяются к единственному типу элементов `A`, то при задании их в селекторе правил этот элемент можно опустить. Следующие два правила равносильны:

```
A:link    { color: red }   /* непосещенная связь */
:link     { color: red }   /* непосещенная связь */
```

Применение таблиц стилей

К одному документу можно присоединить несколько таблиц стилей, которые одновременно будут влиять на представление документа в окне браузера. Этот принцип является основополагающим принципом применения кас-

кадных таблиц стилей, и можно указать на две причины, по которым он рекомендован группой разработчиков Консорциума W3:

1. Реализация модульности — разработчики таблиц стилей могут комбинировать таблицы стилей, каждая из которых отвечает за определенный этап форматирования документа. Например, в одной таблице можно определить все правила форматирования шрифтов, во второй — правила позиционирования элементов и т. д.

2. Соблюдение равновесия между авторской разработкой и пристрастиями читателя — и авторы страниц, и читатели могут влиять на представление страницы через таблицы стилей. Пользователь может определить собственные правила отображения элементов документа в своих личных таблицах стилей, а браузер, на основании механизма приоритетности, будет использовать те или иные правила из таблиц пользователя или автора страницы.

Но как взаимодействуют правила, определяющие форматирование одинаковых элементов и заданные в разных таблицах? Может, например, оказаться, что в связанной таблице цвет шрифта определен как черный, в импортируемой как красный, а во встроенной как синий, — какой цвет шрифта будет иметь документ, отображаемый в браузере? Подобные конфликты разрешаются с использованием *принципа приоритетности* разных таблиц стилей. Таблицы выстраиваются в цепочку приоритетности (от низшего к наивысшему), образуют каскад, по которому документ "прокатывается" и постепенно форматируется на основе правил таблиц стилей, образующих каскад.

Говорят, что правила, определенные в таблице с большим приоритетом, имеют больший *вес*. По умолчанию веса правил читателя всегда меньше весов правил, определенных в таблицах разработчика документа. То есть, если возникает конфликт между правилами, определенными в документе, и правилами, определенными в личных таблицах стилей пользователя, применяться будут первые. Правила читателя и автора всегда перекрывают установки по умолчанию значений свойств используемого браузера.

Приоритетность импортируемых таблиц стилей зависит от порядка их импортирования в документ. В таблице стилей можно использовать несколько операторов `@import`. Каждый последующий оператор импортирует таблицу стилей с более высоким приоритетом по отношению к предыдущим, и поэтому ее правила имеют больший вес. Импортируемые таблицы стилей сами могут импортировать и перекрывать другие таблицы.

Любые правила, определенные непосредственно в самом документе, перекрывают правила импортируемых таблиц стилей. Обычно все операторы `@import` задаются в таблице стилей до определений правил, а это как раз и показывает, что правила самой таблицы перекрывают правила импортируемых таблиц.

Можно увеличить вес правила с помощью значения `important`, которое задается после значения свойства, вес которого необходимо увеличить:

```
H1 {color: red ! important; background: white ! important}
```

В приведенном примере оба свойства имеют увеличенный вес.

Правило читателя со значением `important` перекрывает аналогичное правило автора, заданное без увеличения веса. Авторское правило со значением `important` перекрывает аналогичное правило читателя с увеличенным весом.

Разрешение конфликтов осуществляется на основе внутреннего механизма, реализованного в каскадных таблицах стилей. Для определения значения комбинации элемент-свойство используется следующий механизм:

1. Ищутся все определения, применяемые к рассматриваемой комбинации элемент-свойство. Определения применяются, только если селектор соответствует рассматриваемому элементу. Если невозможно применить ни одно определение, используется наследуемое значение свойства. Если не определено наследуемое значение (это справедливо в случае элементов HTML и свойств, которые не могут наследоваться), используется начальное значение.
2. Определения сортируются по явно заданным весам: определения со значением `important` имеют больший вес по сравнению с нормальным (без увеличения веса) определением.
3. Определения сортируются по своему происхождению: авторские таблицы стилей перекрывают таблицы стилей читателя, которые, в свою очередь, перекрывают установки значений свойств по умолчанию браузера. Импортируемая таблица стилей имеет одинаковое происхождение с таблицей, из которой она импортирована.
4. Определения сортируются по *специфичности селектора*: селектор с большим числом специфичности перекрывает селектор с меньшим числом специфичности. Для определения числа специфичности селектора определения подсчитывают количество параметров `ID` (`a`), параметров `CLASS` (`b`) и названий тэгов (`c`) в селекторе. Составляется число `abc`, которое и определяет специфичность данного определения. Несколько примеров, взятых из рекомендаций REC-CSS1-19990111, приведено ниже:

   ```
   LI                {...} /* a=0 b=0 c=1 --> специфичность =   1 */
   UL LI             {...} /* a=0 b=0 c=2 --> специфичность =   2 */
   UL OL LI          {...} /* a=0 b=0 c=3 --> специфичность =   3 */
   LI.red            {...} /* a=0 b=1 c=1 --> специфичность =  11 */
   UL OL LI.red      {...} /* a=0 b=1 c=3 --> специфичность =  13 */
   #x34y             {...} /* a=1 b=0 c=0 --> специфичность = 100 */
   ```

   Псевдоклассы и псевдоэлементы считаются как нормальные элементы и классы, соответственно.

5. Определения сортируются по порядку их задания в таблице стилей: если два правила имеют одинаковый вес, заданное позже правило имеет приоритет. Правила в импортированных таблицах стилей считаются задаваемыми до любого правила в таблице стилей, из которой она импортируется.

При определенном навыке применение алгоритма приоритетности правил не представляет труда. Для новичков можно просто запомнить следующую приоритетность правил (от низшего к высшему): связанная таблица стилей, импортируемая таблица стилей, правило с элементом HTML в качестве селектора, правило с параметром `CLASS` в качестве селектора, правило с параметром `ID` в качестве селектора, встроенное в тэг HTML правило. Для начальных разработок таблиц стилей этих правил вполне достаточно.

Модель форматирования

Для правильного применения правил форматирования следует представлять, как в каскадных таблицах стилей происходит форматирование элементов, т. е. на что можно влиять и что можно изменять в отображении элемента.

Модель форматирования каскадных таблиц стилей ориентирована на представление любого элемента HTML в окружении вложенными прямоугольными блоками, как показано на рис. 10.2.

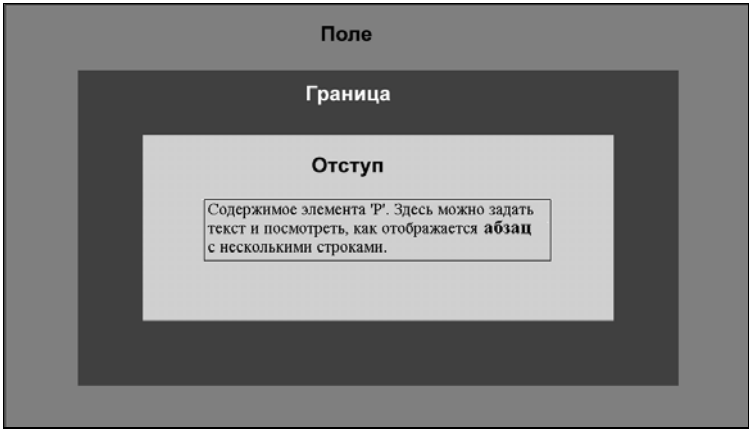


Рис. 10.2. Блоковая модель форматирования абзаца

Блок содержимого элемента (самый внутренний блок) отделен от *границы* отступами. Самым внешним блоком является *поле*. Свойства таблиц стилей позволяют устанавливать размеры и цвета всех блоков, составляющих в сумме отображаемый элемент. Поле всегда является прозрачным прямоугольником, поэтому его цвет наследует цвет родителя элемента (для абзаца это элемент `BODY`). *Отступ* всегда имеет цвет фона самого элемента.

Все перечисленные блоки в совокупности составляют блок форматирования, или отображения элемента, т. е. видимое в окне браузера изображение элемента. Размеры блока форматирования элемента складываются из размеров самого элемента и размеров отступов, границы и полей.

С точки зрения процесса форматирования документа существуют два типа элементов: блоковые и встроенные.

Блоковые элементы

Каждый элемент в модели форматирования имеет свойство `display`, значение которого определяет, отображается или не отображается (`none`) элемент, является ли он блоком (`block`), списком (`list-item`) или встроенным элементом (`inline`).

Элементы со значением свойства `display` равным `block` или `list-item`, а также элементы со значением свойства `float`, отличным от `none` (не "плавающие" элементы), являются *блоковыми элементами*. Их форматирование связано с установкой значений соответствующих параметров вложенных блоков, составляющих элемент в целом. На рис. 10.3 показаны все параметры, доступные в модели форматирования каскадных таблиц стилей для блоковых элементов.

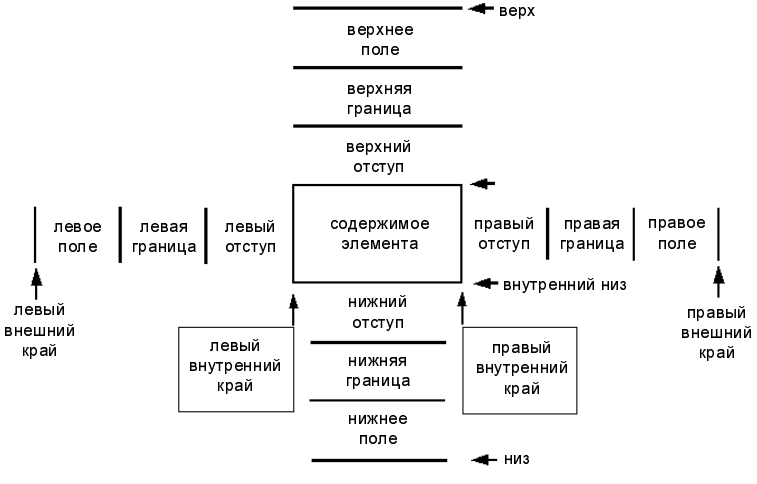


Рис. 10.3. Параметры форматирования блоковых элементов

Примечание

Параметры "верх" (`top`) и "низ" (`bottom`) не применяются к блоковым элементам, они существуют только для встроенных (inline) элементов, речь о которых пойдет ниже.

Ширина (width) элемента — это ширина блока содержимого, и определяется как расстояние между левым и правым внутренними краями. *Высота* (height) элемента — расстояние между внутренним верхом и низом.

При вертикальном форматировании блоковых, не "плавающих" элементов значения параметров "верхнее поле" и "нижнее поле" определяют минимальное расстояние до границ блоков окружающих их элементов. Если у двух примыкающих элементов определены не нулевые значения параметров полей, то при вертикальном форматировании поля двух элементов сливаются в одно со значением, равным максимальной высоте поля одного из двух элементов. Подобное слияние можно наблюдать для элементов `LI` списка `UL` или `OL`.

Горизонтальное форматирование элемента определяется значениями семи свойств: левое поле (`margin-left`), левая граница (`border-left`), левый отступ (`padding-left`), ширина (`width`), правый отступ (`padding-right`), правая граница (`border-right`) и правое поле (`margin-right`). Сумма значений этих семи параметров всегда равняется ширине элемента-родителя или ширине окна браузера, если элемент не вложен в другой элемент.

По умолчание параметр `width` имеет значение `auto`. Если элемент не является замещаемым (в тэге элемента задан параметр `SRC`), то браузер вычисляет ширину элемента из условия равенства суммы значений семи указанных выше параметров ширине элемента-родителя. Для замещаемого элемента значение ширины автоматически заменяется шириной замещаемого элемента, если только значение ширины установлено в `auto`. В противном случае размер замещаемого элемента подгоняется под заданную в параметре ширину (процедура подгонки зависит от браузера).

Чтобы удовлетворить условию равенства суммы значений семи параметров ширине элемента-родителя или окна браузера, в рекомендациях Консорциума W3 определено, в каких случаях значения каких параметров устанавливаются браузером в `auto`.

Свойство `float` может переводить любой элемент в разряд "плавающих". Это приводит к тому, что указанный элемент выводится из нормального потока отображения и форматируется как блоковый элемент. Например, установка свойства `float` элемента `<IMG>` равным `left` позволяет создать *буквицу* при выводе абзаца текста, как показано на рис. 10.4.

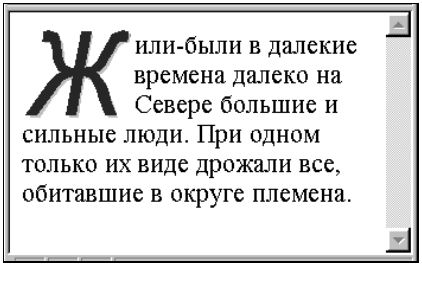


Рис. 10.4. Плавающий элемент `IMG`, представляющий букву Ж

При значении свойства `float` равным `left`, элемент сдвигается влево до поля, отступа или границы другого блокового элемента, а нормальный поток отображения будет обтекать его с правой стороны. Текст файла HTML для приведенного примера представлен ниже.

Пример 10.1. Буквица в тексте

```
<HEAD>
<STYLE TYPE="text/css"><!--
   IMG {float: left}
--></STYLE>
</HEAD>
<P><IMG SRC=Ж.gif>
или-были в далекие времена далеко на Севере
большие и сильные люди. При одном только их виде
дрожали все, обитавшие в округе племена.
</P>
```

Таблица стилей определяет все элементы `IMG` как плавающие элементы со смещением влево. Встроенная в абзац картинка, смещаясь влево, образует буквицу абзаца.

Встроенные элементы

Элементы, которые не форматируются как блоковые, являются *встроенными (inline) элементами*. Они совместно с другими элементами используют область строки. Обычно, выделяемые в строке элементы (`EM`, `STRONG`, `B` и т. д.) классифицируются как встроенные.

Рассмотрим пример задания встроенных элементов в блоковый элемент абзац `P`:

```
<P>В абзац можно помещать
<B STYLE='background-color: lightsteelblue'>встраиваемые элементы</B>.
Их может быть
<EM STYLE='background-color: lighteelblue'>несколько</EM>.</P>
```

Если ширина абзаца достаточна для отображения всех встроенных элементов в строке, то они отобразятся в одной строке. Если ширина недостаточна, то некоторые элементы могут быть разбиты на два и отображены в двух строках (рис. 10.5).

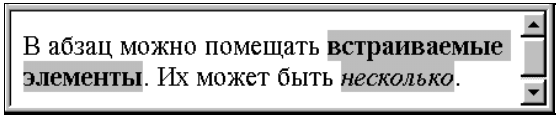


Рис. 10.5. Отображение встраиваемых элементов

Свойства форматирования элементов

В каскадных таблицах стилей, уровень 1 и 2 (CSS1 и CSS2), все доступные свойства форматирования элементов в документе HTML разбиты на 9 категорий, представленных в табл. 10.1. Категории или свойства, появившиеся во втором уровне каскадных таблиц стилей, отмечены верхним индексом [2].

Таблица 10.1. *Категории свойств элементов*

Категория	Устанавливает
Шрифт	Типографские свойства шрифтов
Цвет и фон	Цвет текста и фона, а также картинки в качестве фона
Текст	Выравнивание, форматирование и разрядку текста
Блок	Свойства форматирования блоковых элементов
Визуальное форматирование	Свойства, связанные с блоками отображения элементов, их позиционированием и отображением списков
Печать[2]	Спецификацию разрыва страницы
Фильтры и переходы[2]	Мультимедийные эффекты и преобразования графических изображений
Псевдоклассы и другие свойства	Свойства `@import`, `cursor`[2] и `important`

Прежде чем переходить к описанию свойств форматирования, остановимся на единицах измерения, используемых для задания значений свойств.

Для задания значений свойств, определяющих некоторые размеры, в каскадных таблицах стилей применяются относительные и абсолютные единицы измерения длины. *Относительные единицы* задают длину относительно значения другого свойства, определяющего длину. Документы, в которых таблицы стилей используют относительные единицы измерения, более приспособлены для отображения на разных устройствах (например, дисплей или лазерный принтер). *Абсолютные единицы* измерения полезны только тогда, когда известны физические характеристики устройства отображения. В табл. 10.2 отображены единицы измерения, используемые в таблицах стилей.

Таблица 10.2. *Единицы измерения в таблицах стилей*

Единицы измерения			
Относительные		**Абсолютные**	
`em`	Высота шрифта элемента	`in`	Дюйм (1 in = 2.54 cm)
`ex`	Высота буквы x	`cm`	Сантиметр

Таблица 10.2 *(окончание)*

Единицы измерения			
Относительные		**Абсолютные**	
px	Пиксел	mm	Миллиметр
%	Процент	pt	Пункт (1pt = 1/72 in)
		pc	Пика (1 pc = 12 pt)

Примечание

Относительные единицы измерения em и ex во всех свойствах вычисляются относительно высоты шрифта элемента. Единственное исключение — свойство font-size, в котором эти единицы относятся к высоте шрифта элемента-родителя.

Примечание

Пикселы являются единицами измерения, относящимися к разрешению дисплея компьютера. Если плотность пикселов на устройстве вывода сильно отличается от типового дисплея, браузер должен переопределить эту единицу. Новая единица, *ссылочный пиксел*, — это угол, под которым виден один пиксел монитора с плотностью 90dpi с расстояния вытянутой руки пользователя (28 дюймов).

В качестве значений цветов можно использовать зарезервированные ключевые слова HTML для определения наиболее употребительных цветовых оттенков (например, aqua, black, white и т. п.) или использовать цветовую модель RGB. Примеры способов задания цветовых оттенков приведены ниже:

```
EM { color: #f00 }              /* #rgb соответствует #rrggbb */
EM { color: #ff0000 }           /* #rrggbb */
EM { color: rgb(255,0,0) }      /* целые в интервале 0 — 255 */
EM { color: rgb(100%, 0%, 0%) } /* вещественные от 0.0% до 100.0% */
```

Для задания URL-адреса ресурса используется функциональная запись url(...):

```
BODY { background: url(http://www.bhv.ru/CSS/logo.gif) }
```

Скобки, запятые, пробелы, одинарные и двойные кавычки в URL-адресе задаются с предшествующей обратной косой чертой (\), \(, \,).

Примечание

В каскадных таблицах стилей частичный URL-адрес интерпретируется относительно месторасположения таблицы стилей, а не относительно расположения документа.

Шрифты

Выбор подходящего шрифта для отдельных частей документа является одним из наиболее часто выполняемых действий в процессе разработки HTML-документа. Шрифты различаются по своему внешнему виду (начертанию), по размеру, по стилю (прямой, курсив или наклонный) и по "жирности" отображения (нормальный, полужирный). Каскадные таблицы стилей предоставляют в распоряжение разработчика набор свойств для установки всех перечисленных параметров шрифтов. Кроме того, уровень 2 каскадных таблиц стилей позволяет загружать отсутствующие на компьютере читателя шрифты непосредственно с сервера, на котором расположен документ.

Свойство *font-family*

Свойство `font-family` задает приоритетный список семейств шрифтов и/или типовых семейств шрифтов. Если использовать для отображения страницы один определенный шрифт, то может оказаться, что этот шрифт не поддерживает некоторые символы, содержащиеся на странице, или на компьютере пользователя нет вообще этого шрифта. Для разрешения подобных проблем это свойство позволяет разработчику страницы задать список шрифтов одного стиля и размера, среди которых браузер может искать необходимый символ. В отличие от других свойств каскадных таблиц стилей названия семейств в списке отделяются запятыми, чтобы показать их альтернативность:

```
BODY {font-family: TimesDL, "Times New", serif}
```

При интерпретации HTML-страницы браузер сначала ищет на компьютере пользователя шрифт TimesDL. Если такой шрифт отсутствует, то браузер пытается применить шрифт Times New, а если и он не найден, то используется любой шрифт из семейства шрифтов serif — одного из типовых семейств шрифтов компьютера.

Понятие типовых семейств шрифтов введено в каскадные таблицы стилей с целью реализации наихудшего варианта отображения страницы, если не найдены специально использованные автором шрифты. В любой реализации каскадных таблиц стилей должны существовать пять типовых семейств шрифтов, которые соответствуют реальным шрифтам, обычно устанавливаемым на большинстве компьютеров:

- serif (например, Times)
- sans-serif (например, Helvetica)
- cursive (например, Zapf-Chancery)
- fantasy (например, Western)
- monospace (например, Courier)

Имена шрифтов, состоящих из нескольких слов, должны заключаться в кавычки:

```
BODY {font-family: "Times New Roman", serif}
<BODY STYLE="font-family: 'Times New Roman', serif">
```

Предупреждение

Следует использовать кавычки разных типов при задании последовательности всех определяемых свойств в параметре `STYLE` и при задании имени шрифта в свойстве `font-family`.

Свойство *font-style*

Свойство `font-style` определяет стиль шрифта из выбранного семейства: нормальный (`normal`), курсивный (`italic`) или наклонный (`oblique`).

Нормальный шрифт — это обычный прямой шрифт, используемый для печати документов. *Курсивный стиль* шрифта напоминает каллиграфические этюды в прописях первоклассников и близок к рукописному. *Наклонные шрифты* генерируются из обычных прямых шрифтов небольшим наклоном символов.

Обычно в базе шрифтов браузера все шрифты, в именах которых встречаются слова Oblique, Slanted или Incline отмечены как наклонные (`oblique`) шрифты. Шрифты, в названиях которых присутствуют слова Italic, Cursive или Kursiv, отождествляются браузером с курсивными (`italic`).

Следующие правила определяют курсивный стиль шрифта заголовка первого уровня и нормальный, прямой шрифт выделенных частей заголовка:

```
H1 {font-style: italic}
H1 EM {font-style: normal}
```

Свойство *font-variant*

Каскадные таблицы стилей реализуют еще одну вариацию шрифта выбранного семейства — капитель (small-caps). В шрифте этого стиля все строчные буквы выглядят как прописные, но меньшего размера и с немного измененными пропорциями.

Значение `normal` свойства `font-variant` не изменяет вида шрифта, а значение `small-caps` выбирает вариант капитель шрифта. Рекомендации по каскадным таблицам стилей допускают создание шрифта капитель простой заменой строчных букв масштабированными символами верхнего регистра.

Следующие правила задают отображение заголовка четвертого уровня капителью с наклонной капителью в выделенных частях:

```
H4 {font-variant: small-caps}
EM {font-style: oblique}
```

Примечание

Свойство `font-variant` наследуется элементом `EM` от своего родителя — поэтому в приведенном примере выделенные части заголовка будут отображаться наклонной капителью.

Свойство *font-weight*

Это свойство выбирает из заданного семейства шрифт определенной жирности. В рекомендациях регламентируется 9 градаций жирности шрифта, задаваемых числами 100, 200 и так далее до 900. Значение 100 соответствует самому "бледному" шрифту, тогда как 900 — самому "жирному".

Для задания нормального шрифта используется ключевое слово `normal`, что соответствует цифровому значению 400. Значение `bold` применяется для выбора общепринятого полужирного начертания шрифта и его цифровым эквивалентом является 700.

Выбор определенной градации жирности шрифта не означает, что в семействе существует шрифт с заданной жирностью. Единственное, что гарантируется, — это то, что шрифт с большим значением жирности не светлее предыдущего значения. Некоторые семейства имеют только две градации жирности: нормальную и полужирную.

Свойство *font-size*

Это свойство определяет размер шрифта. Его значение может быть абсолютным или относительным.

Абсолютное значение можно задать одним из следующих ключевых слов: `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`, которые являются индексами в таблице размеров шрифтов, поддерживаемой браузером. Масштабирующий множитель соседних значений в уровне 1 каскадных таблицах стилей определялся как 1.5, а в уровне 2 его значение рекомендовано равным 1.2. Это означает, что шрифт размера `large` в 1.2 раза выше шрифта `medium` и в 1.2 ниже шрифта `x-large`. По умолчанию браузер использует значение `medium`.

Абсолютное значение можно задать и в виде абсолютного значения длины, например 10pt, но в этом случае высота шрифта не зависит от хранимой таблицы размеров шрифтов браузера.

Ключевые слова для задания относительного размера шрифта интерпретируются относительно таблицы размеров шрифтов и размера шрифта элемента-родителя. Возможными значениями могут быть: `larger` и `smaller`. На-

пример, если родитель имеет размер шрифта medium, то значение larger сделает шрифт текущего элемента равным large.

Относительный размер шрифта можно задать также в процентах к размеру шрифта родителя или в относительных единицах длины:

```
P {font-size: 10pt}
EM {font-size: 120%}
EM {font-size: 1.2em}
```

Два последние правила для выделенного в абзаце элемента EM определяют одинаковую высоту шрифта 12pt.

Свойство *font*

Основное назначение свойства font — установить в одном определении значения свойств font-style, font-variant, font-weight, font-size, line-height и font-family. Все значения перечисленных свойств задаются через пробелы в том порядке, как они перечислены выше. Первые три свойства могут не задаваться, что соответствует установке их значений в normal. Размер шрифта и высота строки (свойство line-height) задаются через косую черту. Элементы списка семейств шрифтов свойства font-family задаются через запятую:

```
P {font: oblique 12pt/14pt "Times Cyr", serif}
```

В этом примере для абзаца задается наклонный шрифт Times Cyr высотой 12 пунктов. Высота строк — 14 пунктов. Если не найден шрифт Times Cyr, то применяется любой шрифт типового семейства serif.

Свойство *@font-face*

Это свойство введено в каскадные таблицы стилей в рекомендациях уровня 2. Оно предназначено для задания семейства шрифта. Если указанный шрифт отсутствует на компьютере пользователя, то он загружается из сети по заданному вторым параметром URL-адресу:

```
@font-face {font-family: CoolFont;
src:url(http://myserver.com/CoolFont.eot);}
```

Примечание

Все свойства шрифтов являются наследуемыми и применяются ко всем элементам документа HTML.

Цвет и фон

Свойства этой категории определяют цвет и фон элемента. Фон можно задать в виде цвета или изображения. В случае изображения указывается его

положение, как оно повторяется и фиксировано ли оно или прокручивается вместе с прокруткой содержимого окна браузера.

Для установки цвета текста элемента существует единственное свойство color. Его значением является цвет, задаваемый с помощью ключевых слов или rgb-функции:

```
<P> {color: blue}
<EM> {color: rgb(0,0,255)}
```

Оба правила в примере устанавливают синий цвет текста соответствующих элементов.

Для установки параметров фона элемента существует несколько свойств, задающих значения индивидуальных параметров фона, и свойство background, в котором можно установить одновременно все значения параметров фона.

Цвет фона определяется значением свойства background-color, а изображение, используемое в качестве фона, задается свойством background-image. Начальным значением свойства background-color является transparent, которое определяет фон элемента как прозрачный. Значением свойства background-image является абсолютный или относительный адрес файла изображения, используемого в качестве фона. Если это свойство определено, то рекомендуется задать также и цвет фона, который будет использоваться в случае недоступности файла изображения.

```
BODY {background-color: lightsteelblue;
      background-image: url(/image/image.gif);}
<P> {background-image: none}
```

В приведенном примере задается адрес файла изображения для фона тела документа и явно указывается отсутствие фона для абзацев документа.

Если фон задан в виде изображения, то свойство background-repeat определяет его *повторяемость* и способы повторяемости. Допустимыми значениями являются repeat (повторяемость и по вертикали, и по горизонтали), repeat-x и repeat-y (повторяемость соответственно по горизонтали или вертикали) и no-repeat (изображение не повторяется). В следующем примере

```
BODY {background-color: lightsteelblue;
      background-image: url(/image/image.gif);
      background-repeat: repeat-y;}
```

задается повторяемость изображения фона по вертикали.

Свойство background-attachment определяет, будет ли фон, на котором отображается документ, оставаться неподвижным при прокрутке содержимого окна браузера или он будет прокручиваться вместе с документом. В первом

случае реализуется эффект перемещения содержимого окна над неподвижным рисунком. Значение `fixed` оставляет фон неподвижным, а значение `scroll` заставляет его перемещаться вместе с содержимым документа при прокрутке. Пример закрепленного в окне браузера изображения фона представлен ниже:

```
BODY {background-color: lightsteelblue;
      background-image: url(/image/image.gif);
      background-repeat: repeat-y;
      background-attachment: fixed;}
```

Свойство `background-position` определяет начальное положение изображения, используемого в качестве фона, в блоке содержимого элемента. Значением этого свойства являются координаты привязки определенных точек изображения и блока содержимого. Их можно задавать в процентах, в абсолютных единицах длины, а также с использованием комбинаций ключевых значений.

Пара `0% 0%` означает, что верхний левый угол изображения помещается в верхний левый угол блока содержимого элемента (это значение является значением по умолчанию). Пара `100% 100%` размещает нижний правый угол изображения в нижний правый угол блока содержимого. Пара значений, отличных от указанных, например `10% 80%`, помещает точку изображения, расположенную на расстоянии в 10% ширины от левого края и в 80% высоты от верхнего края, в точно такую же точку блока содержимого элемента.

Пара абсолютных значений, например `10mm 10mm`, размещает верхний левый угол изображения на 10 мм правее и на 10 мм ниже левого верхнего угла блока содержимого.

Ключевые значения и их допустимые комбинации вместе с эквивалентными числовыми значениями представлены в табл. 10.3.

Таблица 10.3. *Допустимые комбинации ключевых значений*

Комбинация	Значение
`top left`, `left top`	`0% 0%`
`top`, `top center`, `center top`	`50% 0%`
`top right`, `right top`	`100% 0%`
`left`, `center left`, `left center`	`0% 50%`
`center`, `center center`	`50% 50%`
`right`, `center right`, `right center`	`100% 50%`
`bottom left`, `left bottom`	`0% 100%`

Таблица 10.3 *(окончание)*

Комбинация	Значение
bottom, bottom right, bottom center	50% 100%
bottom right, right bottom	100% 100%

Свойство background позволяет одновременно устанавливать значения свойств background-color, background-image, background-repeat и background-attachment. Все допустимые значения индивидуальных свойств задаются в виде списка, элементы которого отделены пробелами. Если значение какого-либо свойства не задано, то оно устанавливается в начальное значение, определяемое браузером:

```
BODY {background: lightsteelblue url(/image/image.gif) center}
```

Это правило устанавливает цвет и изображение фона, а также положение изображения в окне браузера. Остальные свойства фона принимают начальные значения.

Примечание

Свойство color наследуется по обычным правилам. Все свойства, определяющие параметры фона, не наследуются, но фон элемента-родителя будет отображаться по умолчанию, так как начальным значением свойства backround-color является transparent (прозрачный).

Форматирование текста

Свойства данной категории влияют на отображение символов, слов и абзацев. Они определяют расстояние между словами и буквами в словах, задают отступы и высоту строк в абзацах.

Свойство letter-spacing влияет на расстояние между символами при отображении текста. Его значение, задаваемое в единицах длины, определяет пробел, добавляемый к установленному по умолчанию пробелу между символами. На рис. 10.6 показано отображение текста с установками по умолчанию и с увеличенным на 0.5em пробелом между символами:

```
<P STYLE="letter-spacing: 0.5em">
Слово слово слово слово слово
</P>
```

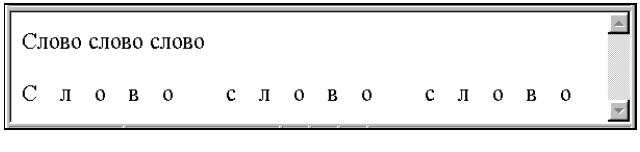


Рис. 10.6. Абзац с увеличенным расстоянием между символами

Примечание

Браузер увеличивает не только расстояние между символами слов, но и расстояние между словами.

Каскадные таблицы стилей позволяют преобразовывать текст. Если значение свойства `text-transform` равно `capitalize`, то все слова отображаются с прописной буквы. Значения `uppercase` и `lowercase` этого свойства приводят, соответственно, к преобразованию всех букв в прописные или строчные, независимо от их задания в тексте документа HTML.

Свойство `text-decoration` задает подчеркивание, надчеркивание или перечеркивание текста. Соответствующие значения этого свойства следующие: `underline`, `overline` и `line-through`.

Выравнивание текста в блоке содержимого элемента определяется значением свойства `text-align`. Текст выравнивается по левому краю при значении `left`, по правому краю — при значении `right` и по центру — при значении `center`.

Предупреждение

Текст выравнивается относительно блока содержимого элемента, а не относительно окна отображения браузера.

Отступ первой строки элемента задается значением свойства `text-indent`, которое определяет величину отступа в абсолютных или относительных единицах длины.

Свойство `vertical-align` определяет положение элемента по вертикали относительно элемента-родителя. Его значением может быть любое ключевое слово из табл. 10.4.

Таблица 10.4. *Ключевые значения выравнивания по вертикали*

Значение	Результат
`baseline`	Выравнивание базовой линии элемента (или низа, если элемент не имеет базовой линии) по базовой линии родителя
`middle`	Выравнивание средней точки элемента (обычно изображения) на уровне базовой линии родителя плюс половина ширины блока содержимого родителя
`sub`	Элемент отображается в виде нижнего индекса
`super`	Элемент отображается в виде верхнего индекса
`text-top`	Выравнивание верха элемента с верхом шрифта элемента-родителя
`text-bottom`	Выравнивание низа элемента с низом шрифта элемента-родителя

Таблица 10.4 *(окончание)*

Значение	Результат
`top`	Выравнивание верха элемента с верхом самого высокого элемента строки
`bottom`	Выравнивание низа элемента с ниже всех расположенным элементом строки

Значения этого свойства, заданные в виде процентов, вычисляются относительно высоты строки (свойство `line-height`) самого элемента. Они поднимают базовую линию (или низ элемента, если он не имеет базовой линии) на заданную высоту относительно базовой линии элемента-родителя, если значение положительно, и опускают, если значение отрицательно.

Расстояние между базовыми линиями двух соседних строк (высота строки) задается установкой значения свойства `line-height`. Числовое значение этого свойства определяет высоту строки, вычисляемую умножением размера шрифта текущего элемента на заданное число.

Примечание

Все текстовые свойства, кроме свойств `text-decoration` и `vertical-align`, наследуются элементами-потомками от родителей.

Блоки

Свойства этой категории устанавливают параметры блоковых элементов *(см. выше раздел "Модель форматирования")*. Среди них можно выделить три больших группы, форматирующих блоки полей, границ и отступов, причем группу, работающую с границей, можно подразделить еще на четыре группы, устанавливающие значения цвета, стиля, ширины и одновременно всех перечисленных свойств границы. Все эти группы отличает то, что в них включены свойства для задания параметров верхних, нижних, правых, левых и одновременно всех четырех частей соответствующих блоков форматирования элемента.

Совет

При знакомстве со свойствами блоковых элементов желательно иметь перед глазами схему параметров форматирования блоковых элементов, схематически представленную на рис. 10.3.

В группу форматирования поля входят свойства, устанавливающие ширину верхнего (`margin-top`), правого (`margin-right`), нижнего (`margin-bottom`) и левого (`margin-left`) поля элемента. В свойстве `margin` можно одновременно установить значения всех четырех параметров поля элемента.

Ширина соответствующих полей задается значением длины или в процентах от ширины ближайшего элемента-родителя. Начальные значения всех полей равны 0.

Если в свойстве `argin` заданы четыре значения, то они, соответственно, относятся к верхнему, правому, нижнему и левому полю. Если определено только одно значение, то оно применяется ко всем сторонам поля элемента. При задании двух или трех значений недостающие значения берутся из установок противоположных сторон. Например:

```
BODY {margin: 1em 2em} /* верх и низ = 1em, право и лево = 2em */
```

Ширина верхнего, правого, нижнего и левого отступа определяется значением, соответственно, свойств `padding-top`, `padding-right`, `padding-bottom` и `padding-left`. Свойство `padding` позволяет одновременно установить значения всех четырех отступов элемента. Все, что было сказано о задании значений для одновременной установки полей, относится и к этому свойству.

Ширину верхней, правой, нижней или левой границы задают соответственно свойствами `border-top-width`, `border-right-width`, `border-bottom-width` и `border-left-width`. Значения свойства `border-width` определяют ширину границы элемента для всех перечисленных ее частей. Все, что было сказано о задании значений для одновременной установки полей, относится и к этому свойству.

Значениями этих свойств могут быть ключевые параметры `thin`, `medium` и `thick` или значение длины. Ширина границы, определяемая ключевыми параметрами, зависит от браузера. Единственное, что можно гарантировать, — это то, что ширина `thin` не больше ширины `medium`, которая, в свою очередь, не больше ширины `thick`.

Цвета частей границы задаются значениями свойств `border-top-color`, `border-right-color`, `border-bottom-color` и `border-left-color`. Свойство `border-color` определяет цвета всех частей границы. Четыре параметра цвета подчиняются все тем же правилам, описанным при задании полей элемента. Если задан тип границы (*см. ниже*), но не задан цвет границы, то по умолчанию используется цвет самого элемента.

Все предыдущие установки свойств границы не будут иметь никакого воздействия на отображение элемента, если не установлен тип границы, так как по умолчанию тип границы не определен, и она не отображается.

Для задания типа любой из четырех частей границы применяются свойства `border-top-style`, `border-right-style`, `border-bottom-style` и `border-left-style`. Свойство `border-style` определяет одновременно типы всех частей границы. Значениями этих свойств могут быть ключевые параметры `none`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`. Типы границ, соответствующих всем перечисленным значениям, представлены в табл. 10.5.

Таблица 10.5. *Типы линий границы*

Ключевой параметр	Тип границы
none	Граница не отображается (несмотря на значение свойства border-width)
solid	Граница отображается сплошной линией
double	Граница отображается двойной линией (сумма толщины двух линий и промежутка между ними равна значению свойства border-width
groove	Граница отображается, как будто она вдавлена в лист ("желобок")
ridge	Граница отображается, как будто она выдавлена из листа ("барельеф")
inset	Весь блок элемента отображается, как будто он вдавлен в лист
outset	Весь блок элемента отображается, как будто он выдавлен из листа ("барельеф")

Последняя большая группа свойств позволяет установить ширину, тип и цвет частей границы или всей границы в целом. Свойства border-top, border-right, border-bottom и border-left определяют ширину, тип и цвет, соответственно, верхней, правой, нижней и левой границы. Свойство border определяет одновременно параметры всех частей границы. В отличие от аналогичных свойств, задающих параметры полей и отступов, данное свойство устанавливает одинаковые значения для всех частей границы.

Визуальное форматирование

Каскадные таблицы стилей предоставляют разработчику Web-документов мощные средства компоновки элементов HTML на странице документа, определяющие внешний вид страницы HTML в окне браузера. В CSS1 большинство этих свойств относилось к категории позиционирования, и, ввиду большой важности, их описание было выделено в отдельный документ рекомендаций, подготовленный Консорциумом W3. В CSS2 эти и некоторые другие свойства сгруппированы в разделах, относящихся к визуальному форматированию.

Элементы HTML отображаются браузером последовательно, в том порядке, как они определены в тексте HTML-файла с учетом их положения в структуре документа и расположения предыдущих отображенных элементов и элементов-контейнеров, в которых они могут содержаться. При компоновке страницы используются установки браузера для определения положения каждого элемента. Например, два последовательных абзаца следуют друг за другом, причем каждый начинается с новой строки.

Свойство `position` элемента позволяет определить способ его позиционирования на странице: *статический*, *относительный* или *абсолютный*. Относительный способ определяет смещение элемента относительно его естественного положения в потоке отображения элементов. Абсолютный способ удаляет элемент из естественного потока позиционирования и позволяет разместить его на странице абсолютно произвольным образом. Статический способ, являющийся умалчиваемым способом позиционирования элементов, предполагает естественный поток отображения элементов страницы в окне браузера в соответствии с иерархией объектов документа.

Значения `static`, `relative` и `absolute` свойства `position` определяют соответствующий способ позиционирования элемента, который, в конечном счете, складывается из значения указанного свойства элемента, его положения в иерархической структуре документа, местом его определения в исходном файле HTML и значениями его свойств `top` и `left`. Эти последние свойства определяют смещение вниз и вправо левого верхнего угла блока отображения элемента *(см. выше раздел "Модель форматирования)*.

Абсолютное позиционирование

Абсолютно позиционированный элемент и все его потомки изымаются из естественного потока отображения элементов и позиционируются независимо, причем сам элемент или его потомки могут перекрывать ранее отображенные элементы.

Чтобы определить точку отсчета местоположения элемента, следует найти его ближайшего родителя, позиционированного абсолютно или относительно. Положение левой верхней вершины блока этого элемента и будет точкой отсчета для абсолютно позиционированного элемента. Если процесс поиска подобного родителя (следует пропускать все позиционированные статически элементы) дойдет до элемента `<BODY>`, то тело документа и будет тем элементом, относительно которого позиционируется исходный элемент.

Следующий фрагмент страницы HTML иллюстрирует абсолютное позиционирование элемента:

```
<SPAN STYLE="position:static;
background-color:#90EE90">
Родитель, позиционированный статически!
  <IMG SRC="Ж.gif" STYLE="position:absolute; top:60px; left:60px;">
</SPAN>
```

На рис. 10.7 показано отображение страниц с абсолютно позиционированным элементом (графическим изображением буквы Ж) в тэге `<SPAN>`, позиционированном статически (*а* и *б*) и относительно (*в* и *г*).

Так как в исходном фрагменте родитель (`<SPAN>`) элемента `<IMG>`, позиционированного абсолютно, является статически позиционированным элемен-

том, то ищется ближайший абсолютно позиционированный родитель. Таковым будет тело документа, относительно начала которого и смещается вправо и вниз на 60 пикселов изображение (рис. 10.7, *а*). Если к этому фрагменту добавить один абзац, то элемент `<SPAN>` сместится вниз, но его потомок — элемент `<IMG>` — все равно будет позиционирован относительно начала документа (рис. 10.7, *б*), что приводит к перекрытию изображением родителя.

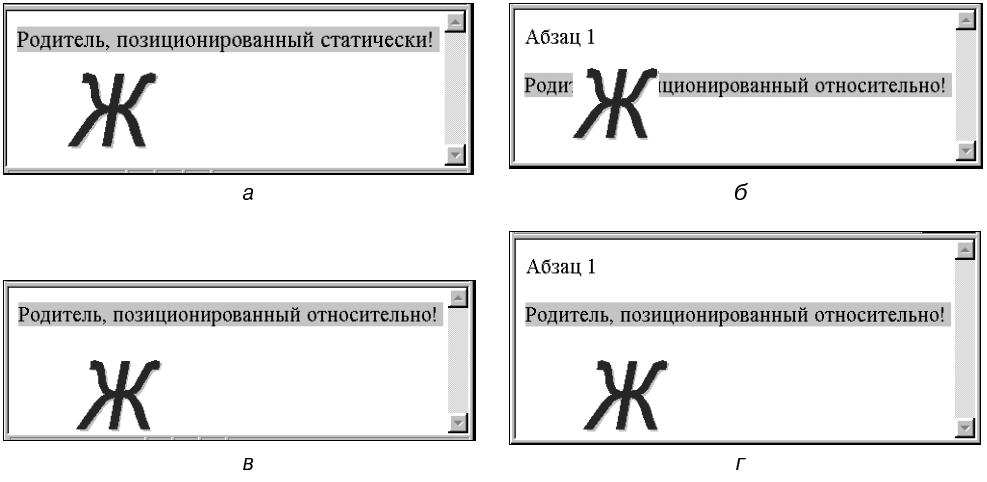


Рис. 10.7. Отображение абсолютно позиционированного изображения

Если теперь изменить позиционирование родителя на относительное, то элемент `<IMG>` будет смещен вниз и вправо относительно начала элемента-родителя (рис. 10.7, *в*). А если добавить перед элементом `<SPAN>` абзац, то изображение сместится вместе со своим родителем, оставаясь расположенным на 60 пикселов вниз и вправо относительно начала элемента-родителя (рис. 10.7, *г*).

Примечание

Изменение размеров окна браузера не изменит положение абсолютно позиционированного элемента. Поэтому при уменьшении размеров окна такой элемент может оказаться за границами окна и, следовательно, не видим.

Значения свойств `top` и `left` влияют на начало отсчета положения абсолютно позиционированного элемента. Если их значения установлены, то именно они и используются для смещения элемента относительно начала позиционированного элемента-родителя. Если значение свойства `top` не задано или установлено равным `auto`, то верхний край элемента совмещается с верхом последней строки текста родителя, если последний содержит текст, или выравнивается по верхнему краю родителя, если последний является изображением. Аналогично, если значение свойства `left` не задано или уста-

новлено равным auto, то левый край элемента совмещается с концом последней строки текста родителя, если последний содержит текст, или выравнивается по правому краю родителя, если последний является изображением.

Еще два параметра, влияющие на отображение абсолютно позиционированного элемента, — это свойства width и height элемента. Как отмечалось, при абсолютном позиционировании элемент изымается из стандартного потока и отображается самостоятельно в своем собственном прямоугольном блоке, левый верхний угол которого помещается в определенную точку окна браузера. Свойства width и height задают ширину и высоту этого блока. Если они не заданы, то по горизонтали блок распространяется до правого края окна браузера, а по вертикали — на столько, на сколько необходимо для отображения содержимого элемента. Установка значений свойств width и height ограничивает размеры блока абсолютно позиционированного элемента. Если его содержимое не помещается в блок заданного размера, то оно просто не видимо пользователю. Динамическим изменением размеров блока можно сделать так, что будет видно все содержимое элемента.

Относительное позиционирование

Относительно позиционированные и *статически позиционированные* элементы после изъятия из исходного текста документа всех абсолютно позиционированных элементов (вместе с их потомками), образуют непрерывный поток отображения, в котором каждый последующий элемент позиционируется относительно конца предыдущего.

Относительно позиционированные элементы, являющиеся потомками абсолютно позиционированных элементов, также позиционируются в конец своего элемента-родителя.

Рис. 10.8, *а* демонстрирует отображение в окне браузера последовательности относительно позиционированных элементов страницы:

```
<SPAN STYLE="position:relative;
background-color: gray">
Это изображение </SPAN>
<IMG SRC="Ж.gif" STYLE="position: relative;">
<SPAN STYLE="position:relative;
background-color: lightgrey">
является изображением буквы "Ж".
</SPAN>
```

Каждый элемент потока позиционируется в конец предыдущего. Если позиционирование элемента-изображения изменить на абсолютное, то он будет выведен из стандартного потока отображения, а оставшиеся относительно позиционируемые элементы будут отображаться друг за другом в едином потоке (рис. 10.8 *б*). Здесь же видно, что последовательность отображения

элементов определяется их заданием в исходном файле документа: элемент `<IMG>` задан вторым, поэтому он отображается раньше третьего элемента и перекрывается последним.

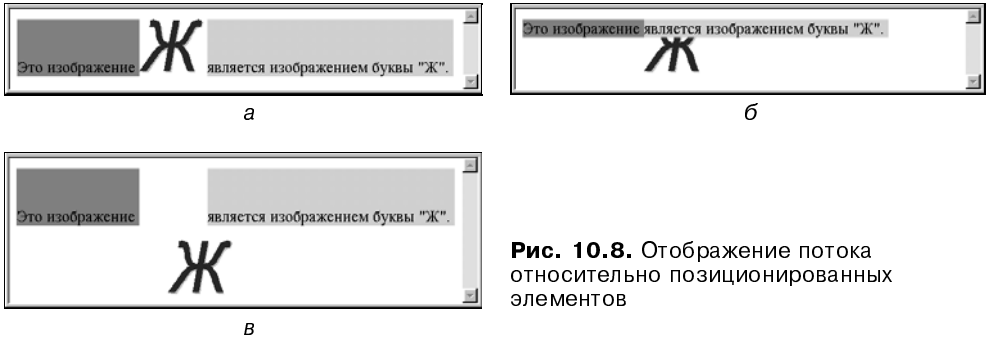


Рис. 10.8. Отображение потока относительно позиционированных элементов

До сих пор мы не задавали значений свойств `top` и `left` относительно позиционируемых элементов. Если для какого-либо элемента из стандартного потока определены значения этих свойств, то этот элемент смещается вниз и вправо на заданные величины относительно правого верхнего угла блока предыдущего элемента в потоке, а следующий элемент отображается так, как будто предыдущий смещенный элемент остается не смещенным. Это правило иллюстрируется на рис. 10.8, *в*, где отображен предыдущий фрагмент, в котором параметр `STYLE` элемента `<IMG>` имеет значение "`position: relative; top: 100px; left: 40px;`".

Статическое позиционирование

В стандартном потоке *статически позиционированные* элементы ведут себя аналогично относительно позиционированным: они отображаются непосредственно сразу после предыдущего элемента в потоке. Единственное их отличие от относительно позиционируемых элементов заключается в том, что для них нельзя установить значения свойств `top` и `left`, и тем самым сместить их, например, со строки абзаца вверх или вниз.

Визуальные эффекты

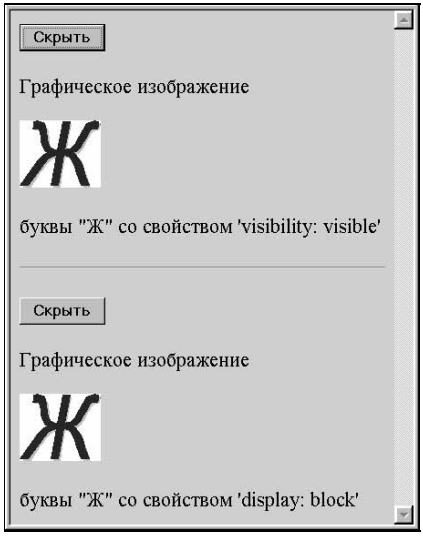
Несколько свойств каскадных таблиц стилей позволяют организовать, совместно со встроенными сценариями, динамическое отображение и скрытие элементов страницы HTML.

Свойство `visibility` управляет отображением элемента. Если его значение равно `visible` (значение по умолчанию), то элемент отображается, если оно установлено равным `hidden`, то элемент не отображается. Когда для скрытия элемента используется его свойство `visibility`, то элемент не изымается из

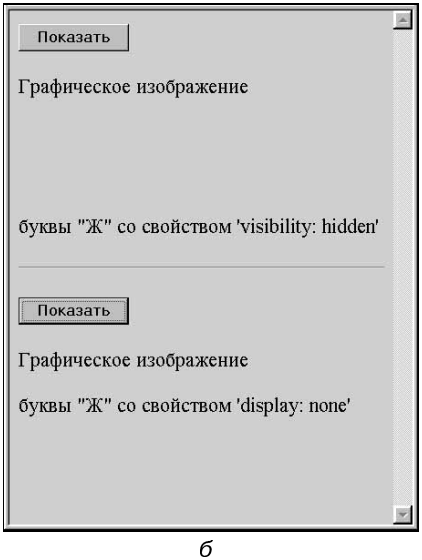
потока отображения. Это означает, что соответствующий ему блок занимает надлежащее положение на странице, но содержимое этого блока (элемент) не отображается.

Подобное поведение отличается от поведения объекта со свойством `display` равным `none`. В последнем случае элемент не только не отображается, но и изымается из потока отображения, и на странице нет пустого блока, соответствующего этому элементу.

Два разных поведения скрываемых элементов, связанных с использованием двух разных свойств, показаны на рис. 10.9.



а



б

Рис. 10.9. Поведение элементов при их скрытии

На рис. 10.9, *а* показано отображение двух элементов `<IMG>`, источником изображения которых является графический файл буквы Ж. Верхний элемент определен со свойством `visibility` равным `visible`, а нижний элемент — со свойством `display` равным `block`. При нажатии соответствующих кнопок свойство `visibility` первого изображения становится равным `hidden`, а свойство `display` второго изображения изменяется на `none`. На рис. 10.9, *б* показан результат отображения скрытых изображений. Блок первого изображения остается в потоке отображения, и поэтому только исчезает картинка. Второе изображение изымается из потока отображения, и поэтому никакого зарезервированного для картинки места не остается.

Свойство `clip` позволяет обрезать видимое изображение абсолютно позиционированного элемента. Каждый такой элемент отображается в прямоугольном блоке определенной ширины и высоты, которые определяются

либо шириной и высотой самого элемента, либо его свойствами width и height. Та часть элемента, которая отображается в блоке, является его видимым изображением. Блок отображения не обязательно должен соответствовать размерам самого элемента: он может вмещать весь элемент, а может вмещать только его часть. Например, блок отображения элемента <IMG> может быть меньше размеров графического изображения, представляемого этим элементом. Свойство clip воздействует *только* на блок отображения элемента. Его значением может быть auto (никакого отсечения не производится) или границы прямоугольника видимого изображения элемента, которые задаются с помощью следующего параметра rect(<top> <right> <bottom> <left>). Величины <top>, <right>, <bottom> и <left> определяют, соответственно, верхнюю, правую, нижнюю и левую границы видимого изображения элемента относительно блока отображения. На рис. 10.10, *а* показано графическое изображение с размерами 89×74 пиксела, а на рис. 10.10, *б* то же изображение со свойством clip равным rect(10 70 60 10).

а *б*

Рис. 10.10. Свойство clip обрезает видимое изображение элемента

Свойство overflow определяет поведение элемента, когда размеры его содержимого не соответствуют размерам блока отображения, установленного свойствами top, left, width и height. Существует четыре значения этого свойства, определяющие поведение элемента:

- visible — заставляет элемент сжаться или увеличиться, чтобы полностью отобразиться в заданном блоке (для графического содержимого элемента) или увеличивает размеры блока отображения (для текстового содержимого);
- hidden — обрезает элемент в соответствии с размерами блока;
- auto — добавляет полосы прокрутки к блоку отображения в случае, если размеры содержимого элемента превосходят размеры блока отображения;
- scroll — добавляет полосы прокрутки к блоку отображения в любом случае.

На рис. 10.11 показан один и тот же раздел (элемент <DIV>) с разными значениями свойств overflow. Свойства width и eight раздела во всех вариантах его отображения одинаковы и соответствуют размерам первого элемента на рисунке (блок отображения).

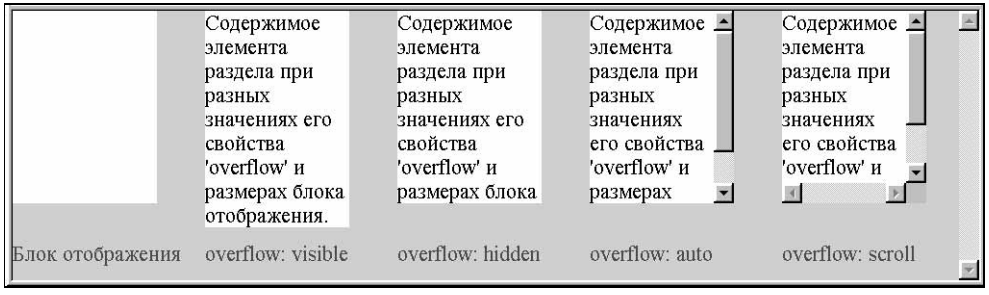


Рис. 10.11. Элементы с разными значениями свойства `overflow`

Все представленные выше свойства влияют на расположение блока отображения элемента в плоскости окна браузера. Однако каскадные таблицы стилей позволяют позиционировать каждый блок и в направлении, перпендикулярном плоскости экрана, что влияет на отображение элемента при перекрытии его другим элементом.

Обычно элемент, появляющийся позже другого в исходном тексте документа HTML, перекрывает ранее отображенные элементы. Свойство `z-index` задает *слой*, в котором располагается элемент при отображении. Если слой расположен ближе к пользователю (значение свойства `z-index` больше), то элемент перекрывает любой другой элемент с меньшим значением слоя, даже если последний и отображается позже. Следующие фрагменты документа HTML иллюстрируют использование свойства `z-index`, а на рис. 10.12 показаны результаты их отображения в окне браузера:

```
<!—Фрагмент 1-->
<IMG STYLE="position:absolute; top:80px; left:0px; z-index:auto"
            SRC=Ж.gif>
<DIV STYLE="position:absolute; top:115px; left:10px; width:150px;
            color:white; background-color:blue; z-index:auto">
    Две буквы "Ж"</DIV>
<IMG STYLE="position:absolute; top:120px; left:50px; z-index:auto"
            SRC=Ж.gif>

<!—Фрагмент 2-->
<IMG STYLE="position:absolute; top:260px; left:0px; z-index:3"
            SRC=Ж.gif>
<DIV STYLE="position:absolute; top:295px; left:10px; width:150px;
            color:white; background-color:blue; z-index:1">
    Две буквы "Ж"</DIV>
<IMG STYLE="position:absolute; top:300px; left:50px; z-index:auto"
            SRC=Ж.gif>
```

Если значением свойства `z-index` является `auto`, то элемент перекрывает все элементы с таким же значением этого свойства, но его перекрывает любой элемент со значением свойства `z-index`, отличным от `auto`.

Первый фрагмент иллюстрирует значение `auto`. Второй элемент в потоке отображения перекрывает первый, а третий — все предыдущие. Во втором фрагменте третья буква Ж перекрывается всеми предыдущими элементами, так как значение ее свойства `z-index` равно `auto`, а у предыдущих элементов это свойство определяет номер слоя. Первая буква Ж перекрывает текст, так как значение ее свойства `z-index` больше значения этого же свойства текста, хотя в потоке отображения она идет ранее.

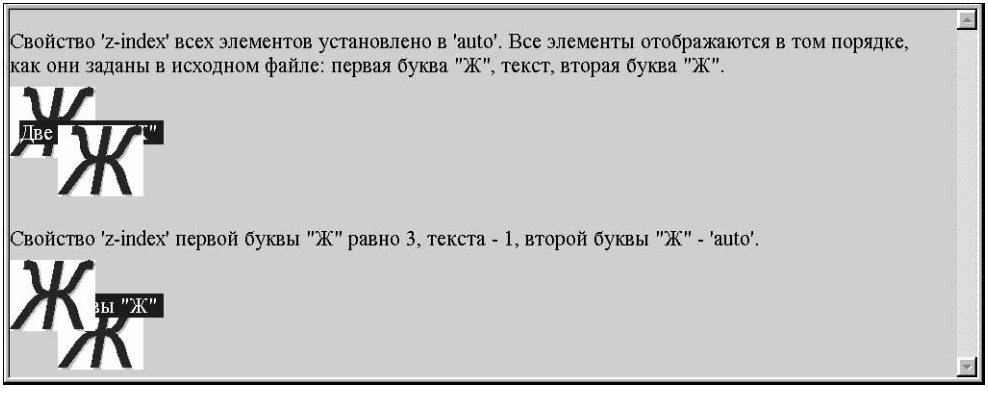


Рис. 10.12. Перекрытие элементов при отображении

Отображение списков

Четыре свойства каскадных таблиц стилей влияют на отображение списков, задаваемых тэгами `<OL>` и `<UL>`.

Свойство `list-style-type` определяет тип маркера списка, если значение свойства `list-style-image` установлено равным `none` или графическое изображение, определенное в этом свойстве функцией `url()`, не доступно. Допустимые значения этого свойства показаны в табл. 10.6.

Таблица 10.6. *Значения свойства* `list-style-type`

Значение	Вид маркера
`none`	Маркер не отображается
`disk`	Закрашенный кружок
`circle`	Не закрашенный кружок
`square`	Закрашенный квадрат
`decimal`	Арабская цифра с точкой
`lower-roman`	Римская строчная цифра с точкой

Таблица 10.6 *(окончание)*

Значение	Вид маркера
upper-roman	Римская прописная цифра с точкой
lower-alpha	Латинская строчная буква с точкой
upper-alpha	Латинская прописная буква с точкой

Свойство `list-style-image` определяет графическое изображение, которое будет использоваться в качестве маркера списка, если оно доступно. Значениями этого свойства могут быть либо `none` (не задается никакое изображение для маркера), либо полный или относительный адрес графического файла, задаваемый с помощью функции `url()`:

```
OL {list-style-image: url(http://www.bhv.com/list.gif)}
```

Свойство `list-style-position` определяет положение маркера в списке: в составе абзаца пункта списка (значение `inside`) или выдвинутым влево от него (значение `outside`):

```
<DIV STYLE="position:absolute; top: 0px; left: 20px; width: 300px">
<UL STYLE="list-style-position: inside"> Список 1
  <LI> Абзац первого перечисления в списке. Маркер в составе абзаца.
  <LI> Абзац второго перечисления в списке. Маркер в составе абзаца.
</UL>
</DIV>

<DIV STYLE="position:absolute; top: 0px; left: 400px; width: 300px">
<UL STYLE="list-style-position: outside"> Список 2
  <LI> Абзац первого перечисления в списке. Маркер впереди абзаца.
  <LI> Абзац второго перечисления в списке. Маркер впереди абзаца.
</UL>
</DIV>
```

Результаты отображения этих двух списков показаны на рис. 10.13.

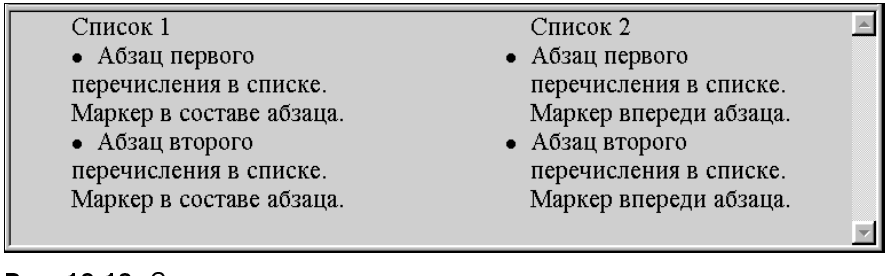


Рис. 10.13. Списки с разным расположением маркеров

Как и для большинства свойств каскадных таблиц стилей, так и для этих свойств определено свойство, в котором можно одновременно определить значения всех свойств отображения списка. Значением свойства `list-style` является список значений свойств `list-style-type`, `list-style-image` и `list-style-position`:

```
OL {list-style: circle url(list.gif) inside}
```

Примечание

Свойства группы "Фильтры и переходы" используются для создания динамических визуальных эффектов в HTML-документах, и поэтому их описание будет дано ниже в разделе "*Динамический HTML в Internet Explorer*".

Объектная модель документа

Объектная модель документа (Document Object Model — DOM) — это то, без чего не возможен динамический HTML. Именно последний явился катализатором в разработке спецификаций DOM в рамках Консорциума W3, и в то же время, он является конечной целью программной реализации объектной модели документа. Объектная модель документа связывает в единое целое HTML, язык сценариев и каскадные таблицы стилей, предоставляя разработчикам Web-документов инструмент с совершенно новыми качествами — динамический HTML.

По своей сути DOM является интерфейсом прикладного программирования (API) для документов HTML. Она определяет логическую структуру документа и способ доступа и манипулирования составляющими документ элементами. Все, что определено в документе тэгами языка разметки страниц, становится доступным для изменения, удаления и добавления. Программист может создавать документы, свободно перемещаться по их структуре и добавлять, изменять или удалять элементы и/или их содержимое.

Одной из основных целей разработки рекомендаций Консорциумом W3 являлось предоставление независимого от языка и платформы стандартного интерфейса прикладного программирования. В настоящее время многие разработчики редакторов и браузеров HTML-документов применяют рекомендации Консорциума W3 по объектной модели документа в своих продуктах.

Динамический HTML в Internet Explorer 4.01 реализован на базе объектной модели DHTML, разработанной фирмой Microsoft и вошедшей в качестве подмножества в объектную модель Консорциума W3. Последняя версия браузера Internet Explorer 5.0 полностью реализует объектную модель документа, совместимую с объектной моделью DHTML.

Целью данного раздела является знакомство читателя с основными концепциями и понятиями объектной модели документов.

Структура документа

В объектной модели документа любой документ представляется в виде логической древовидной структуры. Например, следующий фрагмент документа HTML:

```
<BODY>
.       .       .
<P ID='p1'> В блоковый элемент, каким является абзац,
можно добавлять <B ID='b1'>встраиваемые элементы</B> и
даже другие блоковые элементы <IMG ID='img1' SRC="my.gif">
</P>
<IMG ID='img2' SRC="my-1.gif">
.       .       .
</BODY>
```

будет представлен в объектной модели документа логической структурой, показанной на рис. 10.14.

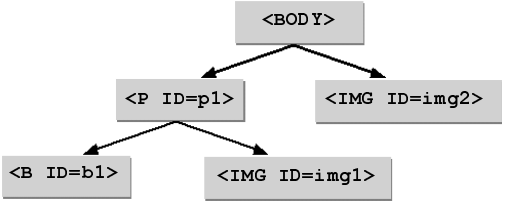


Рис. 10.14. Логическая структура фрагмента документа

Для сложного документа логическая структура будет, естественно, сложнее. В ней может оказаться много "деревьев", которые в сумме будут представлять уже некоторый "лес".

Для понимания объектной модели документов важно осознавать, что логическая древовидная структура представления документа никак не связана с реализацией этой модели именно в виде древовидной структуры. Рекомендации не регламентируют способ реализации модели, она может быть произвольной. Основное — это принцип *структурного изоморфизма*: две реализации объектной модели документа, используемые для представления одного и того же документа, создадут одну и ту же структурную модель с одинаковыми объектами и их связями.

Другой важный аспект модели документов — она оперирует с объектами в полном соответствии с традиционными объектно-ориентированными технологиями: все элементы документа представляются в виде объектов. В узлах структурной логической схемы находятся объекты, а не данные, со всеми присущими объектам свойствами и поведением.

Объектная модель документов, таким образом, определяет:

- интерфейсы и объекты, используемые для представления документа и манипулирования с ним;
- семантику (смысл) этих интерфейсов и объектов, включая и поведение, и параметры;
- "родственные" связи и взаимодействие между этими интерфейсами и объектами.

Основное назначение *реализации* объектной модели документов — предоставить возможность доступа и манипулирования элементами документа из программы с помощью объектов, выстроенных в некоторую иерархическую структуру, а также обеспечить взаимодействие между объектами. Поэтому любая реализация модели включает в свою очередь и управление событиями, представленными также в виде объектов. Разработанные на настоящий момент рекомендации Консорциума W3 не затрагивают вопросов управления событиями, но работа в этом направлении продолжается, и, возможно, в ближайшее время могут появиться рекомендации по объектной модели документа, уровень 2.

В связи с этим хотелось бы отметить ведущую роль фирмы Microsoft, объектная модель DHTML которой легла в основу рекомендаций, разработанных Консорциумом W3. Поэтому следует ожидать, что и стандартизация событий объектной модели документов также будет основываться на реализованной в Internet Explorer событийной модели.

Примечание

В другом популярном браузере Netscape Navigator динамический HTML реализован с использованием собственных технологий фирмы Netscape. Каскадные таблицы стилей в этом браузере поддерживаются не в полном объеме в соответствии с рекомендациями Консорциума W3, а динамический механизм изменения содержимого документа реализован на нестандартных технологиях слоев и динамических шрифтов. В последнем разделе этой главы будет дано краткое описание динамического HTML фирмы Netscape.

Объектная модель DHTML в MS Internet Explorer 4.0

Объектная модель DHTML предоставляет разработчикам Web-документов прямой программируемый доступ ко всем элементам документа, а совместно с событийной моделью подобный подход позволяет браузеру обрабатывать ввод пользователя, выполнять встроенные сценарии и динамически менять содержимое документа, не перезагружая его.

Каждый элемент HTML-документа является в этой модели программируемым. Это означает, что к каждому элементу можно привязать сценарий, выполняемый в зависимости от того, какое действие над элементом произвел

пользователь: поместил ли курсор мыши на элемент, перемещает курсор мыши в границах элемента, нажал или отпустил кнопку мыши и т. д. Результатом выполнения указанных действий является генерация определенного события, которое может быть привязано к конкретному сценарию, выполняемому в случае возникновения события.

Иерархия объектов

Иерархическая структура объектов модели DHTML похожа на иерархическую структуру объектов JavaScript, но дополнительно позволяет обращаться и устанавливать свойства каскадных таблиц стилей.

Каждый объект в иерархии можно использовать в сценариях, но для этого необходимо получить доступ к соответствующему объекту. В иерархической объектной модели одну из важных ролей при доступе к объектам играют два набора: `all` и `children`. В первом содержатся ссылки на *все объекты, расположенные ниже объекта в иерархии*, тогда как второй содержит ссылки на *все объекты, непосредственно порождаемые данным.* Например, набор `all` объекта `body` приведенного выше фрагмента документа *(см. выше раздел "Структура документа")* содержит ссылки на все объекты документа, в том числе и на `p1`, `img1`, `b1` и `img2`. Набор `children` будет ссылаться только на два объекта, непосредственно порождаемых элементом `body`: `p1`, `img2`, при условии, что приведенный фрагмент является законченным документом, и больше не содержит никаких элементов.

На вершине всей иерархии объектов расположен объект `document`, от которого происходят, в конечном итоге, все объекты, представляющие элементы HTML. Этот объект также имеет набор `all` (содержащий ссылки на *все* объекты документа), который можно использовать для доступа к любому объекту страницы.

Например, чтобы обратиться к объекту, представляющему первый абзац предыдущего примера, в сценарии JavaScript это можно сделать следующими способами:

```
document.all.p1
document.all['p1']
document.all[0]
```

Примечание

Набор `all` объекта `document` всегда содержит ссылки на объекты `HTML`, `HEAD`, `TITLE` и `BODY`, даже если соответствующие тэги отсутствуют в документе. Кроме перечисленных объектов в этот набор включаются также комментарии и неизвестные или ошибочные тэги.

Имя элемента HTML, по которому на него можно ссылаться в сценарии, задается либо в параметре `ID`, либо в параметре `NAME` тэга элемента. Каждый

объект, входящий в набор, определяет свойство этого набора с таким же именем, что и имя объекта, соответствующего определенному элементу HTML-документа. Первая ссылка на объект первого абзаца страницы в приведенном примере как раз и использует это свойство наборов и объектов. Последнюю запись можно использовать, если точно известно, что первый абзац является первым элементом набора `all`.

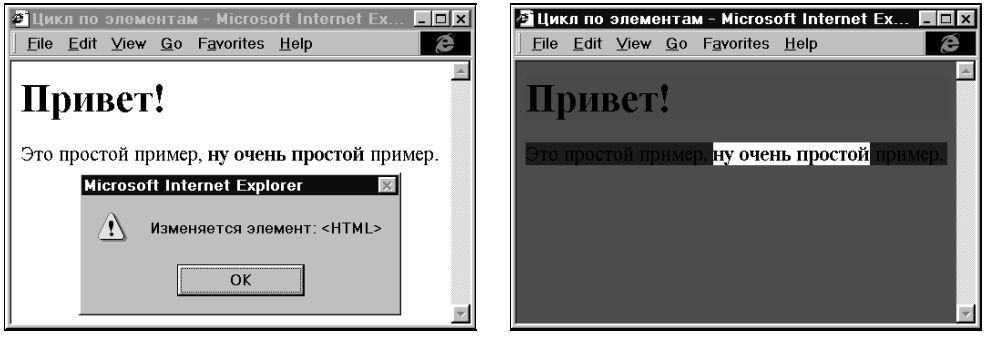
Можно просто ссылаться на элемент, указав его имя, определенное в параметре `ID` или `NAME`. Таким образом, на первый абзац можно сослаться и просто как `p1`.

При программировании достаточно полезным свойством наборов является свойство `length`, в котором хранится общее число объектов в наборе. Его обычно используют, если заранее не известно число элементов в наборе. Сценарий примера 10.2 демонстрирует использование этого свойства для организации цикла по всем элементам страницы HTML. Этот сценарий отображает диалоговое окно и сообщает пользователю, какой элемент он собирается изменить.

Пример 10.2. Изменение цвета фона всех элементов HTML-страницы

```
<HTML>
<HEAD><TITLE>Цикл по элементам</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var mColor = new Array('red','green','blue','white')
function changeAllElements() {
   for (i=0; i<document.all.length; i++) {
     alert("Изменяется элемент: " + "<" + document.all[i].tagName + ">");
     document.all[i].style.backgroundColor = mColor[i%4];
   }
}
</SCRIPT>
</HEAD>
<BODY onload="changeAllElements()>
<H1>Привет!</H1>
<P>Это простой пример, <B>ну очень простой</B> пример.
</BODY>
</HTML>
```

При загрузке документа вызывается функция `changeAllElements()`, которая в цикле просматривает набор `all` объекта `document` и отображает диалоговое окно с сообщением, что будет изменен элемент, задаваемый соответствующим тэгом. В качестве имени элемента в функции `alert()` используется значение свойства `tagName` объекта, в котором как раз и хранится имя тэга. На рис. 10.15 показано окно браузера в начале выполнения сценария (*а*) и после выполнения сценария (*б*).



а б

Рис. 10.15. Изменение параметров форматирования элементов HTML

В начале работы сценария документ отображается со стандартными установками форматирования браузера: черный текст на белом фоне. В процессе выполнения сценария поочередно отображаются диалоговые окна с предупреждением об изменении элементов страницы. Так как к некоторым элементам страницы свойство, устанавливающее цвет фона элемента `background-color`, не применимо, то его установка оператором JavaScript просто игнорируется. Изменения начинаются с элемента `<BODY>`. Сначала цвет фона всей страницы становится красным, затем заголовок отображается на зеленом фоне, текст абзаца — на синем, и в конце сценария текст тэга `<B>` отображается на белом фоне.

В этом примере продемонстрирована техника доступа к свойствам каскадных таблиц стилей форматирования HTML-элементов из программы на языке сценариев JavaScript. Все установленные для объекта свойства каскадных таблиц стилей хранятся в наборе `style` этого объекта и являются, как обычно, свойствами этого набора. Например, обратиться к свойству `color` какого-либо HTML-элемента из сценария JavaScript можно с использованием следующего синтаксиса:

```
<ссылка_на_объект>.style.color
<ссылка_на_объект>.style['color']
```

Здесь `<ссылка_на_объект>` определяет ссылку на объект, о которой речь шла выше.

В каскадных таблицах стилей свойства форматирования имеют названия, в которых используется дефис: `background-color`, `margin-top`, `border-left` и т. д. Имя такого свойства в сценарии определяется следующим образом:

1. Удаляется из названия свойства дефис (-).
2. Часть имени свойства после дефиса присоединяется к предшествующей части с прописной буквы.

3. Предыдущие два пункта повторяются для всех вхождений дефиса в название свойства.

Например, свойство `border-right-width` в сценарии JavaScript будет выглядеть `borderRightWidth`, на свойство `background-color` можно ссылаться как `backgroundColor`.

Предупреждение

Язык JavaScript чувствителен к регистру, поэтому следует внимательно задавать имя свойства каскадных таблиц стилей. Не поставленная прописная буква в имени свойства приводит к тому, что оно просто игнорируется, и, соответственно, не меняет параметры форматирования элемента HTML-страницы.

Иногда необходимо из всего множества объектов страницы выделить подмножество объектов, соответствующих некоторому типу элементов HTML. На странице обычно содержится несколько заголовков, например, первого уровня. В наборе `all` они расположены вперемежку с остальными объектами, но если необходимо выделить их из всего множества элементов, то подобную операцию можно осуществить, используя метод `tags()` набора `all`:

```
var paragraphs=document.all.tags('P')
for (i=0; i< paragraphs.length; i++)
     paragraphs(i).style.color = 'blue';
```

Параметром метода `tags()` является наименование типа тэга, все экземпляры которого необходимо выделить в отдельный набор. В дальнейшем работа со вновь созданным набором осуществляется обычным способом. Так как он содержит объекты, то можно, например, изменить цвет текста всех объектов выделенного типа на HTML-странице, как показано в приведенном фрагменте.

Свойства и методы объектов

Большинство свойств объектов соответствуют параметрам представляемых ими элементов HTML-документа и имеют такие же имена, что и имена параметров. В сценарии можно получить значения интересующих параметров элемента или, наоборот, изменить их установку. Динамическое изменение свойств объектов, и, соответственно, представляемых ими элементов HTML, является основной концепцией динамического HTML.

Имена некоторых свойств объектов отличаются от имен параметров, но обычно достаточно близки к именам представляемых параметров. Например, свойство `className` соответствует параметру `CLASS`. Подобное несоответствие связано, в основном, с именами параметров, которые могут конфликтовать с зарезервированными ключевыми словами основных языков сценариев.

Некоторые свойства объектов доступны только для чтения. Это означает, что можно получить их значение, но нельзя изменить. Примером подобного свойства является свойство `tagName`, представляющее имя тэга HTML-элемента.

В объектной модели существуют свойства, представляющие целое множество свойств объекта. По существу, такие свойства являются наборами и к ним применима описанная выше технология работы с наборами. Свойство `style` объекта — одно из таких свойств. Оно содержит все свойства каскадных таблиц стилей, применимых к соответствующему элементу, и которые можно задавать в параметре `STYLE` тэга элемента. В приведенном выше фрагменте это свойство объекта, представляющего абзац, использовалось для изменения цвета текста.

Некоторые свойства не соответствуют никаким параметрам тэгов. Эти свойства предоставляют дополнительную информацию об элементе и обычно являются свойствами только для чтения. С одним из подобных свойств мы уже знакомы. Свойство `tagName` не соответствует никакому параметру и предоставляет информацию о типе тэга элемента. Другим примером является свойство `sourceIndex`, значением которого является индекс элемента в наборе `all`.

Методы `getAttribute()`, `setAttribute()` и `removeAttribute()` позволяют, соответственно, получить и установить значение параметра элемента, не обращаясь к его свойствам, или удалить параметр. Для методов `getAttribute()` и `removeAttribute()` параметром является строка, задающая имя параметра, в метод `setAttribute()` кроме имени параметра необходимо передать и его значение. Следующий фрагмент сценария использует все три метода для получения и установки значения параметра `ALIGN` объекта `p1`, а также для удаления этого параметра из тэга элемента, соответствующего объекту `p1`:

```
alert("Значение параметра ALIGN равно: " + p1.getAttribute('align'));
p1.setAttribute('align','right')
alert("Значение параметра ALIGN равно: " + p1.getAttribute('align'));
p1.removeAttribute('align')
alert("Значение параметра ALIGN равно: " + p1.getAttribute('align'));
```

Если объект `p1`, например, будет соответствовать абзацу с параметром `ALIGN`, равным `center`, то при выполнении приведенного фрагмента сначала отобразится диалоговое окно, в котором значением параметра `ALIGN` будет `center`. В следующем диалоговом окне это значение изменится на `right`, а в последнем — значение будет не определено (будет выведена пустая строка).

В соответствии с устанавливаемыми значениями параметра `ALIGN` будет изменяться и отображение абзаца в окне браузера. При удалении этого параметра абзац будет отображаться с использованием значения по умолчанию (`left`).

Примечание

Не следует думать, что метод удаления параметра физически удаляет из файла текст задания параметра. Он только меняет его установленное значение на неопределенное. Параметр будет иметь это значение, пока другой сценарий не изменит его. В исходном файле страницы его установка не изменяется.

Эти три метода не чувствительны к регистру, т. е. для них неважно, строчными, прописными или и теми, и другими заданы имя параметра и его значение. Если для установки значения параметра или для задания его имени важен регистр, то дополнительный, последний параметр, принимающий значения `true` или `false`, определяет чувствительность этих методов к регистру.

Иногда возникает необходимость показать в окне браузера часть документа, расположенного достаточно далеко от того места страницы, где в данный момент находится читатель. Конечно, можно поместить на страницу ссылку, по которой читатель и сможет осуществить переход. А если подобную процедуру необходимо произвести из сценария? В этом случае следует воспользоваться методом `scrollIntoView()` элемента, который должен быть отображен в окне браузера. Метод прокручивает содержимое окна браузера, чтобы указанный элемент оказался вверху или внизу окна. Например, в следующем сценарии осуществляется быстрый переход к третьему заголовку первого уровня документа:

```
var myH1=document.all.tags['H1'];
if( myH1.length > 0) myH1[2].scrollIntoView(true);
```

Объектная модель позволяет осуществлять доступ и изменять не только значения параметров элементов, но и их содержимое. Это возможно с помощью следующих свойств объектов: `innerHTML`, `innerText`, `outerHTML` и `outerText`.

Свойства с префиксом `inner` применяются к элементам-контейнерам (например, `DIV`, `SPAN` и `H1`) и заменяют содержимое HTML-элементов внутри контейнера, а свойства с префиксом `outer` применяются ко всем HTML-элементам в теле документа и могут использоваться для замены всего элемента и его содержимого.

Свойства с суффиксом `Text` работают с текстовым содержимым HTML-элемента без ограничивающих тэгов, тогда как свойства с суффиксом `HTML` оперируют с представлением элемента на языке HTML. Например, оператор

```
contents = p1.outerText
```

присвоит переменной строку, являющуюся текстом абзаца `p1`. Если абзац в тексте документа определен следующим образом:

```
<P NAME=p1 ID=p1>
В блоковый элемент, каким является абзац,
можно добавлять <B ID='b1'>встраиваемые элементы</B>
```

```
и даже другие блоковые элементы <IMG ID='img1' SRC="m1.gif">.
Как только что включенное изображение.
</P>
```

то в результате выполнения предыдущего оператора переменная `contents` будет содержать строку:

```
"В блоковый элемент, каким является абзац,
можно добавлять встраиваемые элементы и
даже другие блоковые элементы .
Как только что включенное изображение."
```

Если в операторе присваивания заменить свойство `outerText` на `outerHTML`, то переменная `contents` будет содержать полное описание абзаца на языке HTML, представленное выше, со всеми входящими тэгами, параметрами и их значениями.

Примечание

Ознакомиться с полным описанием всех свойств и методов объектной модели DHTML можно на сервере разработчика фирмы Microsoft по адресу: **http://msdn.microsoft.com/workshop/author/**.

Событийная модель

В объектной модели DHTML с *каждым* элементом страницы можно связать определенное действие пользователя: щелчок кнопкой мыши, нажатие клавиши клавиатуры, перемещение в области элемента курсора мыши и т. д. Эта технология основана на фундаментальном понятии *события* в операционных системах с графическим интерфейсом пользователя. Каждое действие пользователя является причиной возникновения сообщения в операционной системе, которые представляются объектами в объектной модели DHTML. Свойства объектов-событий можно использовать во встраиваемых сценариях для получения информации о событии. При возникновении любого события динамически создается свойство `event` объекта `window`, входящего в объектную модель и представляющего окно браузера. Это свойство и является объектом, соответствующим сгенерированному событию.

Цикл жизни события

Любое событие имеет свой *"жизненный" цикл*: от момента возникновения действия или условия, являющегося причиной генерирования события, до выполнения последнего оператора обработчика события или финальных действий браузера. Цикл жизни любого типичного события включает следующие этапы:

1. Происходит действие пользователя или возникает условие, которое возбуждает событие.
2. Тотчас же корректируется объект `event`, чтобы отразить параметры возникшего события.
3. Событие генерируется — это и есть истинное сообщение о возникшем событии.
4. Вызывается обработчик событий элемента-источника события, который выполняет определенные программистом действия и завершает свою работу.
5. Событие передается вверх по иерархии объектов (bubble up) и вызывается обработчик события объекта, являющегося родителем объекта-источника события. Это "всплытие" вверх по иерархии объектов продолжается, пока не будет достигнут самый верхний объект иерархии — объект `window`, или обработчик события какого-либо объекта не аннулирует событие.
6. Выполняются заключительные действия по умолчанию, если таковые определены, но при условии, что событие не было аннулировано.

Если для элемента-источника события не определен обработчик событий, то в иерархии объектов определяется его родитель, и обработчик событий родителя выполняет соответствующие действия по обработке события. И так происходит до корневого объекта иерархии.

Какие удобства предоставляет подобная технология обработки событий? Прежде всего, нет необходимости для каждого элемента писать процедуру обработки события и присоединять ее к нему. Достаточно написать одну процедуру для элемента-родителя, и она будет обрабатывать события, возбуждаемые всеми порожденными родителем элементами. Это позволяет централизованно обрабатывать наиболее часто возникающие события, и, как результат, требует меньше усилий и времени для написания и поддержки кода процедур обработки событий.

Пример 10.3 демонстрирует технику передачи события вверх по иерархии объектов. В нем щелчки кнопкой мыши на всех элементах страницы обрабатываются централизованно обработчиком события элемента `<BODY>`, который является родителем всех элементов страницы.

Пример 10.3. Передача обработки события родителю

```
<HTML>
<HEAD><TITLE>Всплывание события</TITLE>
</HEAD>
<BODY ID='body' onclick="alert('Не надо щелкать!');">
<H1 ID='head1'>Привет!</H1>
<P ID='parag1'>Это простой пример, <B ID='bold1'>ну очень простой</B>
пример.
```

```
</BODY>
</HTML>
```

Щелчок на любом элементе документа приводит к отображению диалогового окна предупреждений из процедуры обработки события `Click` объекта `body`.

Если к какому-нибудь элементу добавить собственный обработчик событий, то будут выполнены две процедуры: самого элемента и элемента родителя. Если элемент расположен достаточно глубоко в иерархии объектов, и каждый элемент, расположенный выше него, имеет также собственный обработчик событий, то неужели событие будет обрабатываться всеми обработчиками? Да, именно это и произойдет, если только какой-то обработчик не аннулирует "всплывающее" вверх по иерархии событие. Объект `event` имеет свойство `cancelBubble`, которое позволяет аннулировать событие, если установить его значение равным `true`. После этого соответствующее событие не существует, и обработчики этого события для всех, расположенных выше элементов, не вызываются. В примере 10.4 обработчик щелчка мыши выделенного элемента `<B>` аннулирует данное событие. Это приводит к тому, что при щелчке на нем никакого диалогового окна с сообщением не отображается.

Пример 10.4. Аннулирование события

```
<HTML>
<HEAD><TITLE>Аннулирование события</TITLE>
</HEAD>
<BODY ID='body'  onclick="alert('Не надо щелкать!');">
<H1 ID='head1'>Привет!</H1>
<P ID='parag1'>Это простой пример,
<B ID='bold1' onclick="window.event.cancelBubble=true" >ну очень про-
стой</B> пример.
</BODY>
</HTML>
```

Аннулирование события достаточно частое действие в сценариях обработки событий, и будет интенсивно использоваться далее в примерах раздела *"Динамический HTML в Internet Explorer"*.

Присоединение обработчика события к элементу страницы осуществляется установкой значения параметра обработки события тэга элемента или соответствующего свойства объекта. О том, как это можно сделать, подробно рассказывалось в главе 9 в разделе *"Язык создания сценариев JavaScript"*.

Каждый элемент HTML может быть источником многих событий. В табл. 10.7 представлены события, которые могут генерировать и обрабатывать все элементы страницы, и соответствующие им параметры обработки событий.

Таблица 10.7. *Общие события всех HTML-элементов*

Параметр	Условие возникновения
`onmouseover`	Курсор мыши перемещается в область элемента (т. е. находится внутри элемента)
`onmouseout`	Курсор мыши выходит за пределы элемента
`onmousedown`	Нажата любая кнопка мыши, когда курсор находится в пределах элемента
`onmouseup`	Отпущена ранее нажатая любая кнопка мыши, когда курсор находится в пределах элемента
`onmousemove`	Курсор мыши перемещается в пределах области элемента
`onclick`	Щелчок левой кнопкой мыши на элементе
`ondblclick`	Двойной щелчок левой кнопкой мыши на элементе
`onkeypress`	Нажата и отпущена клавиша клавиатуры. Если клавиша удерживается, то генерируется серия события `Keypress`
`onkeydown`	Нажата клавиша клавиатуры. Генерируется только одно событие, даже если клавиша удерживается
`onkeyup`	Отпущена ранее нажатая клавиша клавиатуры

Объект *event*

Как уже отмечалось, объект `event` создается автоматически всякий раз, когда возникает какое-либо событие. Этот объект не зависит от используемого языка создания сценария, и его использование в процедурах обработки событий для получения информации о сгенерированном событии является предпочтительным способом получения достоверной информации о событии.

Каждое событие характеризуется параметрами, которые передаются в сценарий через свойства объекта `event`. Существуют параметры, общие для всех типов событий (например, координаты курсора мыши в окне браузера) и специфические для определенного события (например, код нажатой клавиши для событий клавиатуры). Свойства объекта `event`, как и сам он, являются динамическими и создаются в зависимости от типа произошедшего события. При описании свойства, если не оговорено противное, подразумевается, что оно является общим для всех типов событий.

Свойство `srcElement` определяет элемент документа, явившийся источником события. Оно может быть полезным при централизованной обработке событий элементом, расположенным выше в иерархии объектов документа истинного "виновника" события, и, в зависимости от типа элемента, программа-обработчик может предпринять соответствующие действия.

Важное свойство cancelBubble, аннулирующее событие и прекращающее передачу его на обработку вверх по иерархии объектов, рассмотрено немного ранее в этом же разделе.

Свойство returnValue является булевым и возвращает значение true или false после завершения выполнения процедуры обработки события. При передаче события вверх по иерархии значение этого свойства можно использовать для альтернативной обработки события. Кроме того, если в обработчике события элемента, для которого определены действия по умолчанию, это свойство устанавливается равным false, то это отменяет выполнение действий по умолчанию. Одним из таких элементов является тэг <A>, действием по умолчанию которого является переход по ссылке, задаваемой параметром HREF.

По значениям свойств altKey, ctrlKey и shiftKey элемента-источника события определяется, была ли нажата, соответственно, клавиша <Alt>, <Ctrl> или <Shift> в момент возникновения события. Значение свойства равно true, если клавиша была нажата, и false — в противном случае.

Следующий фрагмент сценария отменяет переход по любой связи в документе, если при щелчке на ней была нажата клавиша <Shift>:

```
document.onclick=click;
function click() {
  if((window.event.srcElement.tagName=='A') && window.event.shiftKey) {
     window.event.returnValue=false;
  }
}
```

Предупреждение

Установка значения свойства returnValue равным false не аннулирует событие. Оно продолжает "всплывать" по иерархии объектов. Отменяется только действия по умолчанию элемента, являющегося источником события.

Для событий мыши определены свойства, значениями которых являются координаты положения курсора в момент возникновения события.

Свойства clientX и clientY представляют координаты относительно области отображения браузера, offsetX и offsetY являются координатами относительно элемента-контейнера, в котором расположен элемент-источник события, screenX и screenY — абсолютные координаты курсора мыши, т. е. координаты экрана монитора. Все значения этих свойств определены в пикселах.

Свойства x и y определяют положение курсора мыши по горизонтали и вертикали относительно позиционированного контейнера, содержащего элемент-источник события. Если ни один контейнер не позиционирован, то положение определяется относительно тела документа <BODY>.

Полезное свойство событий мыши `button` определяет нажатую кнопку мыши. Его возможные значения представлены в табл. 10.8.

Таблица 10.8. *Значения свойства* `button`

Значение	Нажаты кнопки мыши
0	Ни одна
1	Левая
2	Правая
3	Одновременно левая и правая
4	Средняя
5	Одновременно левая и средняя
6	Одновременно правая и средняя
7	Все три одновременно

Свойства `toElement` и `fromElement` применимы только к событиям `onmouseover` и `onmouseout`. Они определяют, от какого элемента и к какому перемещался курсор мыши, и полезны при анализе этих действий для вложенных элементов.

Некоторые элементы на HTML-странице могут получить фокус. В каждый момент времени только один элемент может обладать фокусом, и ввод данных с клавиатуры направляется именно этому элементу. Наиболее часто используемыми элементами с фокусом являются `<BUTTON>` и некоторые типы элемента `<INPUT>`. Для таких элементов при получении ими фокуса генерируется событие `onfocus`, а при потере фокуса — событие `onblur`. Элемент получает фокус при щелчке на нем кнопкой мыши или перемещением на этот элемент с помощью клавиш клавиатуры.

При выделении на странице части документа возникают события, регистрирующие эти действия. Событие `onselectstart` генерируется, когда пользователь нажимает кнопку мыши при расположении курсора в области документа. Если после этого нажатия он перемещает курсор мыши по документу (не отпуская нажатую кнопку), то инициируется событие `onselect`, регистрирующее выделение части документа. У этого события есть действия по умолчанию: визуально отметить выделенную часть документа изменением ее фона. Это действие можно отменить, установив значение свойства `returnValue` события равным `false` в процедуре обработки этого события.

Для самого документа существуют два события, отмечающие некоторые стадии его обработки браузером. На самом деле эти события относятся к объекту `window`, находящемуся на вершине иерархии объектов, но обработчики

этих событий задаются в тэге `<BODY>` документа. Событие `onload` происходит сразу же после того, как были загружены в окно браузера все элементы страницы. Его можно использовать для выполнения действий при первоначальной или повторной загрузке страницы. Событие `unload` генерируется до начала выгрузки документа, когда пользователь желает загрузить другой документ, и в процедуре обработки этого события можно, например, напомнить пользователю о выполнении некоторых действий перед окончательной выгрузкой страницы.

Существует еще большой набор полезных и важных событий в объектной модели DHTML, с которыми можно познакомиться на сервере разработчика фирмы Microsoft по адресу **http://msdn.microsoft.com/workshop/author/**.

Объектная модель документа в MS Internet Explorer 5.0

Объектная модель, реализованная в Internet Explorer 5.0, полностью соответствует рекомендациям REC-DOM-Level-1-19981001 Консорциума W3. Эта модель основана на объектной модели DHTML, рассмотренной в предыдущем разделе, и включает дополнительные возможности динамического управления содержимым документов HTML.

В данном разделе основное внимание будет уделено привнесенным в объектную модель DHTML новшествам, дополнениям и изменениям.

Объектная модель документа предоставляет программисту большие удобства при работе с иерархической структурой объектов документа. Она позволяет:

- Перемещать часть структуры документа в другое место, не разрушая и не создавая ее заново.
- Создавать новые элементы и присоединять их к структуре документа в любом ее месте.
- Организовывать и манипулировать новыми или существующими ветвями структуры фрагмента документа до помещения объектов в структурное дерево документа.

Объектная модель документа представляется *узлами* (node), расположенными в виде иерархической структуры дерева. Концепция объектной модели не привязана ни к какому конкретному представлению документа (HTML, XML, SGML). Она всего лишь описывает логическую организацию документа. Ее реализация в конкретной системе представления документов ставит в соответствие узлам реальные элементы. В объектной модели документа, реализованной для HTML, в узлах могут находиться любые HTML-элементы или текст, называемые *узловыми элементами*.

К узлам, и даже целым ветвям, можно получить доступ из сценария JavaScript, встроенного в документ. Концепция объектной модели документа

позволяет изменить узел или целую ветвь структуры, не разрушая ее. Это приводит к более простому и ясному коду по сравнению с кодом, реализующим изменение структуры документа в объектной модели DHTML.

Каждый узловой элемент порождается другим узловым элементом и может сам выступить родителем других элементов (правда, не все элементы могут быть родителями других элементов). Рассмотрим на примере таблицы HTML, как в объектной модели документа можно быстро изменить часть документа, не затрагивая документ в целом.

Пусть в каком-либо документе задана таблица из трех рядов:

```
<TABLE ID="Table">
<TR>
<TD>Ячейка 1 первого ряда</TD><TD>Ячейка 2 первого ряда</TD>
</TR>
<TR>
<TD>Ячейка 1 второго ряда</TD><TD>Ячейка 2 второго ряда</TD>
</TR>
<TR STYLE="background-color: #CFCFCF;">
<TD>Этот ряд перемещаем</TD><TD>Этот ряд перемещаем</TD>
</TR>
</TABLE>
```

Необходимо переместить последний ряд на место предшествующего. Для начала приведем текст сценария, ориентированный на объектную модель DHTML:

```
function fncInterchange(row){
var rowMove=row.rowIndex;
   var rowMove_Cell1=Table.rows[rowMove].cells[0].innerHTML;
   var rowMove_Cell2=Table.rows[rowMove].cells[1].innerHTML;
   Table.deleteRow(rowMove);
   rowMove --;
   Table.insertRow(rowMove);
   Table.rows(rowMove).style.backgroundColor="#CFCFCF";
   Table.rows(rowMove).insertCell(0);
   Table.rows(rowMove).insertCell(1);
   Table.rows(rowMove).cells[0].innerHTML+= rowMove_Cell1;
   Table.rows(rowMove).cells[1].innerHTML+= rowMove_Cell2;
}
```

В переменной `rowMove` сохраняется индекс перемещаемого ряда, передаваемого в качестве параметра. Переменные `rowMove_Cell1` и `rowMove_Cell1` хранят содержимое двух ячеек перемещаемого ряда. После этих подготовительных действий ряд удаляется методом `rowDelete()` объекта `Table`, и в таблицу вставляется новый ряд перед рядом, предшествующим удаленному. Послед-

ние операторы функции добавляют две ячейки в новый ряд и помещают в них содержимое соответствующих ячеек удаленного ряда.

Как видим, в объектной модели DHTML пришлось видоизменять структуру документа: удалять и добавлять ряды и ячейки. Иначе обстоит дело в объектной модели документа. Методом `swapNode()` можно просто поменять местами узлы-элементы в структуре документа:

```
function fncInterchange(row){
  row.swapNode(row.previousSibling);
}
```

Параметром этого метода является элемент, с которым необходимо поменять местами текущий элемент. Свойство `previousSibling` определяет ближайшего предыдущего брата элемента. На сколько код стал компактнее и понятнее по сравнению с кодом объектной модели DHTML!

Примечание

Этот пример будет выполняться только в Internet Explorer 5.0 или другом браузере, поддерживающим объектную модель документа (DOM).

Добавив в обработчик событий `onclick` какого-либо ряда таблицы вызов функции `fncInterchange()`, можно простым щелчком на соответствующей строке таблицы переместить ее на одну строку выше.

При реализации объектной модели документов в IE5.0 в существовавшую в IE4.0 объектную модель DHTML были добавлены свойства, методы и наборы, позволяющие с максимальным удобством и полнотой использовать все преимущества, предоставляемые этой объектной моделью.

Если все HTML-элементы документа имеют параметр `ID`, то разработчику документа достаточно просто в сценариях определять объекты таких элементов (значение параметра `ID` элемента является символическим именем соответствующего ему объекта в языках сценариев). Но идентифицировать все элементы страницы дело достаточно утомительное, поэтому в объектную модель документа включены набор `childNodes` и свойства `parentNode`, `nextSibling`, `previousSabling`, `firstChild` и `lastChild` объектов, позволяющие легко перемещаться по древовидной иерархии объектов и идентифицировать необходимые для работы объекты.

На примере структуры, создаваемой вложенными списками, продемонстрируем использование перечисленных выше свойств:

```
<UL ID="Parent">
   <LI ID="Node1">Узел 1
   <LI ID="Node2">Узел 2
      <UL>
         <LI ID="Child1">Потомок 1
```

```
        <LI ID="Child2">Потомок 2
        <LI ID="Child3">Потомок 3
      </UL>
   <LI ID="Node3">Узел 3
</UL>
```

В объектной модели документов этот фрагмент будет представлен в виде структуры с отношениями "родства", показанной на рис. 10.16.

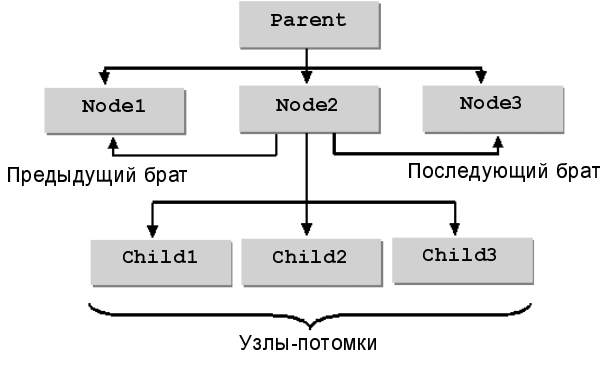


Рис. 10.16. Структура вложенных списков с родственными отношениями

Элементы с именами `Node1`, `Node2` и `Node3` являются узлами-потомками элемента-родителя с именем `Parent`, который в объектной модели также представляется узлом. В наборе `childNodes` объекта `Parent` хранятся ссылки на всех потомков этого элемента (`Node1`, `Node2` и `Node3`). Свойство `parentNode` объектов-потомков возвращает имя родителя объекта, поэтому значением этого свойства объектов `Node1`, `Node2` и `Node3` будет `Parent`. Объекты `Node1`, `Node2` и `Node3` являются ближайшими родственниками одного поколения и открываются друг другу с помощью свойств `previousSibling` (предыдущий ближайший родственник) и `nextSibling` (следующий ближайший родственник). Набор узла-объекта `Node2` покажет всех его потомков (`Child1`, `Child2` и `Child3`).

Объектная модель DHTML также раскрывает все родственные отношения, но для некоторых отношений требуется небольшое программирование, и она использует собственные свойства элементов, отличные от свойств объектной модели документа.

Определить родителя объекта в объектной модели DHTML можно с помощью свойства `parentElement`, которое соответствует свойству `parentNode` узла объектной модели документа.

Предыдущего ближайшего родственника так легко, как в объектной модели документа, в модели DHTML не найти. Необходимо выполнить ряд действий:

```
function fnGetSibling(Obj){
var Parent=Obj.parentElement;
```

```
var iLength=Parent.children.length;
  for(var i=0;i < iLength;i++){
    if(Parent.children[i]== Obj){
      return Parent.children[i-1];
      break;
    }
  }
}
```

Здесь приходится организовывать цикл по набору `children` объекта-родителя, пока не встретится исходный объект, для которого ищется ближайший родственник.

Для установки или получения содержимого объекта в объектной модели документа используется свойство `nodeValue`, тогда как в модели DHTML следует применять свойства `innerHTML` и `innerText`:

```
Node.childNodes[0].nodeValue="Новое содержимое"; // Объектная модель
Node.innerHTML="Новое содержимое";               // Модель DHTML
```

Отметим отличие этих двух моделей при доступе к текстовому содержимому объектов. В объектной модели документа текст является объектом `textNode`, порождаемым элементом, который содержит этот текст. Поэтому доступ к текстовому содержимому объекта осуществляется через набор `childNodes`.

В модели DHTML добавление элементов в структуру документа связано с изменением значений свойств `innerHTML` и `outerHTML` объектов-контейнеров, или применением внешних для объекта методов, например, `insertRow` и `insertCell` для элемента `TABLE`. В объектной модели документа можно создать любой HTML-элемент, задать значения его параметров, а затем встроить в существующую иерархию документа методами `createElement()` и `createTextNode()` объекта `document`.

Примечание

Свойства объектов, соответствующие некоторым параметрам тэгов HTML и являющиеся неизменяемыми свойствами, можно изменять только для самостоятельно существующих объектов, т. е. объектов созданных, но не присоединенных к иерархии документа.

Примечание

В объектной модели DHTML можно использовать свойство `createElement()`, но его действие ограничено созданием только элементов `AREA`, `IMAGE` и `OPTION`.

Если не надо задавать параметры вновь создаваемого элемента или используются их значения по умолчанию, то для создания нового элемента достаточно только его HTML-имя:

```
var newParagraph = document.createElement('P');
```

При создании новых объектов и встраивания их в существующую структуру документа следует формировать объекты с правильной структурой, соответствующей их представлениям в объектной модели документа. Не соблюдение этих правил может привести к неправильно сформированному документу и, в конечном итоге, к его неправильному отображению браузером.

Любая таблица объектной модели документа обязательно состоит, по крайней мере, из двух узлов: `TABLE` и `TBODY`. Поэтому при динамическом создании таблиц не следует забывать об этом обстоятельстве:

```
var Table=document.createElement('TABLE');
var TBody=document.createElement('TBODY');
var Row=document.createElement('TR');
var Cell1=document.createElement('TD');
var Cell2=Cell1.cloneNode();
Row.appendChild(Cell1);
Row.appendChild(Cell2);
Table.appendChild(TBody);
TBody.appendChild(Row);
document.body.appendChild(Table);
Cell1.innerHTML='Ячейка 1';
Cell2.innerHTML='Ячейка 2';
```

Процедура создания таблицы, собственно говоря, повторяет задание тэгов в коде HTML-документа (не пропуская тэгов, вставляемых по умолчанию). Обратим внимание на метод `appendChild(элемент)`, который применим к большинству элементов объектной модели. Он добавляет подчиненный элемент к объекту, для которого вызывается, создавая, таким образом, древовидную структуру. В нашем примере этим методом в строку таблицы были добавлены две ячейки, в тело таблицы добавлена строка, а само тело было добавлено к объекту таблицы `Table`.

Для создания объекта, соответствующего второй ячейке таблицы, использован метод `cloneNode()`, который копирует объект, для которого он применяется, его параметры и набор `childNodes`, если в качестве параметра задано значение `true`. Если параметр метода не задан, то используется значение по умолчанию `false`, при котором набор ссылок на порождаемые объекты не копируется.

Для включения вновь созданной структуры в документ ее необходимо добавить к объекту `body` методом `appendChild()`.

Для манипуляции узлами в объектной модели используются методы `removeNode()`, `replaceNode()` и `swapNode()`.

Метод `removeNode()` удаляет объект, для которого он вызван, из структуры документа. Его единственный параметр может принимать булевы значения `true` или `false`. Значение `true` предписывает удалить и все порожденные

данным объектом объекты, тогда как значение `false` (умалчиваемое) удаляет только сам объект, оставляя в документе все подчиненные ему объекты.

Замену одного объекта другим можно осуществить методом `replaceNode()`, вызываемым для замещаемого объекта. Замещающий объект передается в качестве параметра метода. При замене объекта замещаемый объект удаляется из структуры документа.

Поменять местами два объекта в иерархии документа позволяет метод `swapNode()`. Меняются местами объект, метод которого вызывается, и объект, определяемый параметром метода.

Примечание

Изменение структуры документа требует внимательности и осторожности, так как может привести к неправильной модели документа. Например, удаление из списка верхнего объекта, соответствующего тэгу `<OL>`, приведет к ошибке при интерпретации оставшихся объектов `<LI>`.

В данном разделе мы кратко охарактеризовали объектную модель документа, реализованную в браузере Internet Explorer 5.0, и совместимую с моделью DHTML. Подробнее ознакомиться с объектной моделью документа можно на сервере разработчика фирмы Microsoft по адресу **http://msdn.microsoft.com/workshop/author/**.

Динамический HTML в Internet Explorer

Напомним, что под динамическим HTML понимается технология, позволяющая изменять содержимое отображаемой в окне браузера HTML-страницы без ее перезагрузки. Эта технология использует каскадные таблицы стилей, язык сценариев (например, JavaScript) и обычный HTML, которые объединяются программным интерфейсом — объектной моделью документа (DOM).

Чтобы пользователь мог видеть все динамические эффекты, предусмотренные разработчиком Web-документа, его браузер должен поддерживать все перечисленные компоненты динамического HTML. В противном случае он увидит простую статическую страницу.

Если реализация каскадных таблиц стилей в двух наиболее популярных браузерах — Microsoft Internet Explorer и Netscape Navigator — соответствует рекомендациям Консорциума W3, то относительно объектной модели документа этого сказать нельзя. Объектная модель браузера Internet Explorer согласуется с рекомендациями Консорциума, тогда как в Netscape Navigator реализована собственная модель документа, выполняющая такие же функции, что рекомендованная Консорциумом модель, но немного отличающаяся от нее. Поэтому динамические эффекты, реализованные для одного брау-

зера, могут не выполняться в другом. Особенно это относится к динамическому перемещению элементов и использованию фильтров.

В данном разделе на примерах показано создание основных динамических эффектов в Internet Explorer: отображение скрытой информации, перемещение информации на HTML-странице, создание визуальных эффектов (постепенное отображение информации, использование фильтров для отображения графики) и т. п.

Кроме перечисленных динамических эффектов, реализуемых средствами, рекомендованными Консорциумом, здесь же описаны основные аспекты связывания баз данных с HTML-страницами, и динамического просмотра содержимого баз данных. Эта возможность реализована и поддерживается только в Internet Explorer.

Примечание

Реализации динамического HTML в Netscape Navigator посвящен следующий раздел данной главы.

Динамическое изменение документа

Под динамическим изменением документа понимается изменение его отображения в окне браузера как результат перемещения курсора мыши над элементами страницы, которые, изменяя свой внешний вид, подсказывают пользователю, что можно увидеть скрытую часть документа и щелчком мыши инициировать выполнение некоторого действия. Например, при расположении курсора мыши над каким-либо текстом последний меняет цвет или размер и появляется всплывающая подсказка, что при щелчке кнопкой мыши можно увидеть элементы раскрывающегося списка.

Совет

При разработке динамических HTML-страниц следует всегда продумывать удобные и ясные подсказки пользователю, чтобы он смог полностью насладиться встроенными автором эффектами.

Раскрывающийся список

Создать простое меню можно на основе HTML-элемента `<UL>` и свойства `display` каскадных таблиц стилей, которое позволяет скрывать элементы страницы. Поместим на страницу вложенный список:

```
<UL ID="idList" NAME="idList">
  <LI TITLE="Щелкни и раскрой"
      STYLE="cursor: hand;"> Один
  <UL ID="idListOneA" NAME="idListOneA"
      STYLE="display: none; cursor: default;">
```

```
    <LI TITLE="Файл А">А
    <LI TITLE="Файл Б">Б
  </UL>
  <LI TITLE="Нераскрывающийся список"> Два
  <LI TITLE="Нераскрывающийся список"> Три
</UL>
```

Список `idList` составлен из трех элементов `<LI>` и вложенного списка `idListOneA`, который не отображается (его свойство `isplay` равно `none`) и будет использован для создания раскрывающегося списка при щелчке на первом элементе списка. Строка, заданная в параметре `TITLE`, отображается в виде всплывающей подсказки при расположении курсора мыши над элементом. Этот фрагмент отображается как простой статический список из трех элементов (вложенный список не отображается) с маркерами в виде закрашенных кружков.

Чтобы сделать список раскрывающимся, необходимо добавить к первому элементу внешнего списка и к элементам внутреннего списка обработчики событий и определить в них необходимые действия.

Прежде всего, запрограммируем изменение цвета этих элементов при расположении над ними курсора мыши, чтобы пользователь обратил внимание на их динамичность. Для этого добавим в тэги элементов параметры обработчиков событий:

```
ONMOUSEOVER="flashMe(this,'red')
ONMOUSEOUT="flashMe(this,'black')
```

В обработчиках событий вызывается функция `flashMe()`, изменяющая цвет элемента. Имя элемента (параметр `this`) и цвет передаются в качестве параметров функции:

```
function flashMe(eSrc, sColor) {
eSrc.style.color=sColor
idListOneA.style.color="black"
}
```

Обратим внимание на оператор установки черного цвета списка `idListOneA`. Если его не будет, то при раскрытии первого элемента внешнего списка цвет элементов вложенного списка `idListOneA` будет таким же, как и цвет элемента-родителя — красным (свойство `color` наследуется потомками).

Теперь остается добавить обработчик щелчка кнопки мыши в первый элемент списка `idList`, выполняющий функцию отображения вложенного списка `idListOneA`, если он скрыт, и скрывающий его, если он видим:

```
ONCLICK="toggleListOneA()
```

Исходный текст функции `toggleListOneA()` приведен ниже и не требует комментариев:

```
function toggleListOneA(){
  eTarget=idListOneA
  eTarget.style.display == "none" ? eTarget.style.display="block":
    eTarget.style.display="none"
  eTarget.display == "none" ? eTarget.display="block":
    eTarget.display="none"
}
```

Итак, у нас есть все функции, реализующие раскрывающийся список. Однако мы не учли одного обстоятельства. Если пользователь щелкнет на любом из элементов раскрывшегося списка, то список закроется. Вспомним "подъем" события по иерархии объектов. Хотя в элементе раскрывающегося списка нет обработчика события `ONCLICK`, в любом случае оно передается вверх по иерархии и обрабатывается всеми встречающимися обработчиками этого события. Поэтому процедура обработки события `ONCLICK` внешнего списка закроет внутренний (установит значение его свойства `display` равным `none`). Чтобы этого не происходило, следует в обработчиках событий элементов внутреннего списка отменить передачу события на обработку вверх по иерархии объектов:

```
ONCLICK="window.event.cancelBubble=true"
```

Окончательно исходный текст нашего раскрывающегося списка выглядит так:

Пример 10.5. Раскрывающийся список

```
<HEAD>
<SCRIPT LANGUAGE='JavaScript'>
<!--
function toggleListOneA(){
  eTarget=idListOneA
  eTarget.style.display == "none" ? eTarget.style.display="block":
    eTarget.style.display="none"
  eTarget.display == "none" ? eTarget.display="block":
    eTarget.display="none"
}
function flashMe(eSrc, sColor) {
  eSrc.style.color=sColor
  idListOneA.style.color="black"
}
//-->
</SCRIPT>
<STYLE TYPE="text/css"><!--
  H1 {background-color:lightgrey;
      font-family: Arial;
```

```
      font-size: 11pt;
      color: indianred;
     }
  #idListOne {
    list-style-image:url(item.jpg);
    color: black;
    list-style-position:inside;
    }
--></STYLE>
</HEAD>
<H1> Пример 2.4: Раскрывающийся список </H1>
<UL ID="idList" NAME="idList">
  <LI ONCLICK="toggleListOneA()
      ONMOUSEOVER="flashMe(this,'red')
      ONMOUSEOUT="flashMe(this,'black')
      TITLE="Щелкни и раскрой"
      STYLE="cursor: hand;"> Один
  <UL ID="idListOneA"
      NAME="idListOneA"
      STYLE="display:none; cursor: default;">
    <LI TITLE="Файл А"
        ONCLICK="window.event.cancelBubble=true"
        ONMOUSEOVER="flashMe(this,'red')
        ONMOUSEOUT="flashMe(this,'black')>А
    <LI TITLE="Файл Б"
        ONCLICK="window.event.cancelBubble=true"
        ONMOUSEOVER="flashMe(this,'red')
        ONMOUSEOUT="flashMe(this,'black')>Б
  </UL>
  <LI TITLE="Нераскрывающийся список"> Два
  <LI TITLE="Нераскрывающийся список"> Три
</UL>
```

Графический файл `item.jpg` в свойстве каскадных таблиц стилей `tem-style-image` задает изображение маркера в списках. Свойство `cursor` определяет тип курсора мыши, когда он располагается над элементом. В нашем примере курсор будет меняться на изображение руки (значение свойства `hand`).

На рис. 10.17 показаны результаты отображения раскрывающегося списка: в закрытом (*а*) и раскрытом состояниях (*б*).

Раскрывающийся список в Web-документах можно использовать в качестве простого меню, если в процедурах событий `ONCLICK` его элементов задать отображение каких-либо документов в отдельных окнах или во фрейме страницы.

а б

Рис. 10.17. Раскрывающийся список

Аналогичную технику можно применить для динамического отображения частей документа, связав отображение, например, какого-либо абзаца, с расположением курсора мыши на определенном слове документа, а его скрытие — с перемещением курсора от этого слова. Подобные приемы динамического раскрытия документа позволяют на одном экране представить краткую полную информацию, а детали показать в больших раскрывающихся частях документа.

Движущийся элемент

Положение абсолютно позиционированного элемента на странице легко изменить, установив новые значения его свойств `top` и `left` из встроенного сценария. Если организовать изменение значений этих свойств во времени, то элемент будет двигаться на странице по заданной траектории. Подобное поведение раздела (элемент `<DIV>`) реализовано в примере 10.6.

Пример 10.6. Движущийся раздел текста

```
<HTML>
<SCRIPT LANGUAGE="JavaScript">
<!--
var action, coef_px=-1, coef_pt=1, p_move=5;
function startMove() {
    Banner.style.left =  parseInt(document.body.offsetWidth)/3+200;
    Banner.style.top = 0;
    action = window.setInterval("move(),100);
}
function move() {
    px = parseInt(Banner.style.left);
    px = px + coef_px * p_move;
    Banner.style.left = px;
    if (px <= parseInt(document.body.offsetWidth)/3-200 ||
        px >= parseInt(document.body.offsetWidth)/3+200) {
        coef_px = -coef_px
    }
```

```
    pt = parseInt(Banner.style.top);
    pt = pt + coef_pt * p_move;
    Banner.style.top = pt;
    if (pt <= 0 ||  pt >= 200) {
        coef_pt = -coef_pt
    }
}
-->
</SCRIPT>
<STYLE TYPE="text/css"><!--
  H1 {background-color:lightgrey;
      font-family: Arial;
      font-size: 11pt;
      color: indianred;
     }
--></STYLE>
</HEAD>
<BODY onload="startMove()>
<H1> Пример 2.5: Изменение положения элемента </H1>
<P>Динамический HTML позволяет программно менять положение
элемента!</P>
<DIV ID="Banner" STYLE="position:absolute; width: 200px;
                        background-color: lightsteelblue;">
Добро пожаловать на страницу динамического HTML!</DIV>
</BODY>
</HTML>
```

В теле документа определяется абсолютно позиционированный раздел с именем `Banner`. Его ширина определена в 200 пикселов. При загрузке документа вызывается функция `startMove()`, в которой устанавливаются начальные значения свойств `top` и `left` раздела `Banner`. В этой функции методом `setInterval()` объекта `window` с интервалом в 100 миллисекунд вызывается функция `move()`, которая и реализует перемещение раздела во времени.

Каждый раз при вызове этой функции положение раздела меняется: значения его свойств `top` и `left` увеличиваются или уменьшаются на `p_move` пикселов в зависимости от достижения значениями этих свойств своих нижних или верхних границ. Таким образом, при каждом вызове функции `move()` изменяется положение раздела, что внешне представляется как его движение по странице документа.

Обратим внимание на то, что значения свойств каскадных таблиц стилей необходимо преобразовывать в числовое представление функцией `parseInt()`, так как они хранятся в символьном виде. На рис. 10.18, *а*, *б* показаны два различных момента отображения документа примера 10.6.

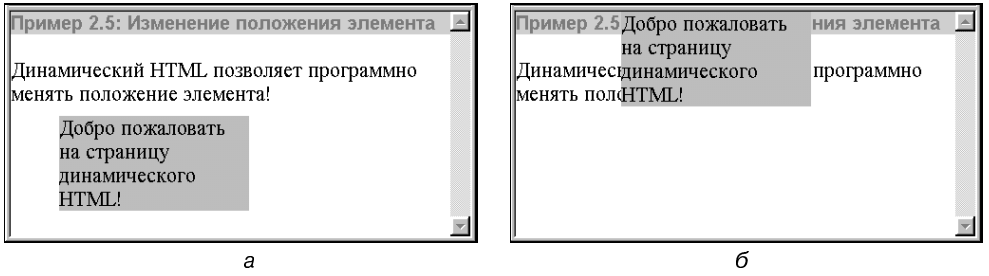


Рис. 10.18. Движущийся раздел документа

Эффекты перемещения частей текста или графических изображений можно использовать для привлечения внимания читателя к важной информации.

Поиск в документе

Если документ большой и содержит разнообразную информацию, можно предоставить читателю возможность поиска задаваемых им слов и автоматического создания указателей на найденные вхождения или перехода к первому вхождению слова.

Пример этого раздела демонстрирует технику поиска в документе и изменение содержимого документа. В текстовом поле пользователь задает слово или фразу, которую необходимо найти в документе. Результаты поиска отображаются на странице. Если найдено соответствующее выражение, то автоматически создается кнопка перехода на первое вхождение введенной в текстовом поле фразы.

Пример 10.7. Поиск, изменение и переход

```
<HTML>
<SCRIPT>
function findme(cValue) {
  if (cValue == null || cValue == '')
     { alert("Задайте слова для поиска"); return; }
  var rng = document.body.createTextRange();
  for (i=0; rng.findText(cValue)!=false; i++) {
    rng.collapse(false);
  }
  spanFound.innerHTML = "Найдено <SPAN STYLE='font-weight: bold'>" + i +
                        "</SPAN> вхождений слова:";
  if(i != 0) {
    search.innerHTML = '<BUTTON' +
      ' style="width:100; text-align: center"' +
      'onclick="findme(txtToFind.value)>Поиск' +
      '</BUTTON> ' +
```

```
      '<BUTTON style="width: 100; text-align: center"' +
      'onclick="move(txtToFind.value)>Перейти</BUTTON>'
  }
}

function move(cValue) {
       var rng = document.body.createTextRange();
       if (rng.findText(cValue) ==true) {
          rng.select();
          rng.scrollIntoView();
        }
}
</SCRIPT>
</HEAD>
<BODY>
<DIV ID="buttons" STYLE="position: absolute; top: 20"> 
<SPAN ID="spanFound">Введите слово для поиска: </SPAN>
<INPUT TYPE=TEXT ID=txtToFind VALUE="">
<SPAN ID="search">
<BUTTON style="width: 100; text-align: center"
       onclick="findme(txtToFind.value)>Поиск
</BUTTON></SPAN>
</DIV>

<DIV style="position: absolute; top:60; font-size:14pt;">

Правда и то: если никто не хочет задумываться, то, казалось бы, тем легче
русскому литератору. Да, легче действительно; и горе тому литератору и
издателю, который в наше время задумывается. Еще горше тому, кто сам
захотел бы учиться и понимать; но еще горше тому, который объявит об этом
искренно; а если заявит, что уже капельку понял и желает высказать свою
мысль, то немедленно всеми оставляется...

<P>Ф.М. Достоевский "Дневник писателя за 1873 год"
</DIV>
</BODY>
</HTML>
```

В этом примере в разделе `buttons` задается встроенный элемент `<SPAN>` с именем `spanFound`, в который выводится информация о числе найденных вхождений фрагментов текста, задаваемых в текстовом поле `txtToFind`, а также кнопка **Поиск** для инициализации процедуры поиска. Эта кнопка содержится в элементе `<SPAN>` с именем `search`, в который, в случае успешного поиска, добавляется кнопка перехода к первому вхождению фразы.

При нажатии на кнопку **Поиск** вызывается процедура `findMe(cValue)`, параметром которой служит введенная пользователем в текстовом поле `txtToFind`

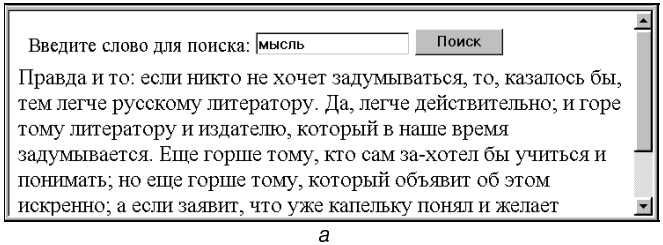
фраза. В процедуре методом `createTextRange()` объекта `body` создается объект `TextRange`, в котором хранится текстовое содержимое всей страницы.

Методом `findText()` этого объекта в цикле определяется число вхождений введенной фразы. Вызов метода `collapse()` необходим, чтобы установить начало следующего цикла поиска в конец предыдущего найденного вхождения.

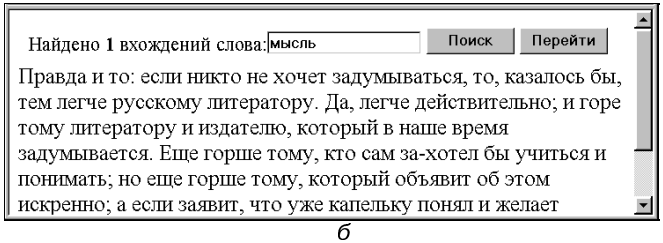
Переустановка значения свойства `inner` объекта `spanFound` приводит к изменению его содержимого. Теперь в нем содержится текст, отражающий количество найденных вхождений заданной фразы.

Если число вхождений не нулевое, то создается кнопка **Перейти**, при нажатии на которую происходит переход к первому вхождению в документ введенной пользователем фразы. Переход осуществляется функцией `move(cValue)`, которая ищет первое вхождение фразы, выделяет ее (метод `select()` объекта `rng`) и осуществляет, в случае необходимости, прокрутку страницы (метод `scrollIntoView()` объекта `rng`) до найденной фразы.

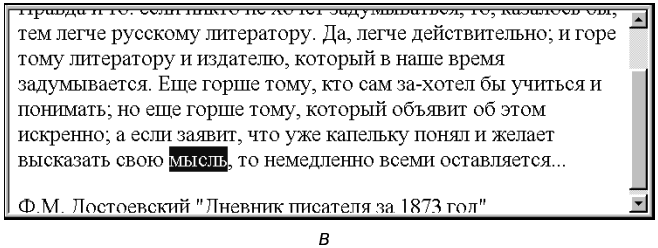
На рис. 10.19 показаны отдельные этапы работы со страницей: *а* — пользователь ввел слова для поиска; *б* — нашлось одно вхождение, и отобразилась кнопка перехода **Перейти**; *в* — пользователь перешел к первому вхождению слова в текст документа.



а



б



в

Рис. 10.19. Этапы поиска информации в документе

Возможности динамического HTML по работе с содержимым страницы не ограничиваются приведенными примерами. Немного фантазии, немного знаний, немного практики — и вы сможете создавать привлекательные и удобные для чтения страницы.

Фильтры и переходы

Создавать HTML-страницы с мультимедийными эффектами также просто, как и применять каскадные таблицы стилей. Разнообразные визуальные динамические эффекты: постепенное "проявление" изображения или текста, изменение контрастности графического изображения, "свечение" букв текста и т. п. — все это и многое другое можно увидеть при отображении страницы в Internet Explorer 4.0 и более поздних версий. Реализуются мультимедийные эффекты применением фильтров к элементам страницы и организацией переходов из одного визуального состояния к другому.

Фильтр — это некоторый алгоритм, преобразующий визуальное отображение элемента в окне браузера. Он может быть статическим или динамическим. *Статический фильтр* преобразует элемент, и после этого он отображается. *Динамический фильтр* воздействует во времени на визуальное отображение элемента, меняя его непосредственно на HTML-странице, что приводит к эффекту анимации. Динамический фильтр еще называют *переходом* из одного состояния отображения в другое.

Фильтры применяются не ко всем HTML-элементам, а только к тем, которые могут определять прямоугольный блок при отображении в окне браузера и не являются окнами, как, например, элемент `<IFRAME>`. В табл. 10.9 перечислены все элементы HTML, к которым могут применяться фильтры.

Таблица 10.9. *Элементы, к которым применяются фильтры*

Элемент	Фильтр применяется
`BODY`	Всегда
`BUTTON`	Всегда
`DIV`	Если заданы ширина (свойство `width`), высота (свойство `height`) или элемент абсолютно позиционирован
`IMG`	Всегда
`INPUT`	Всегда
`MARQUEE`	Всегда
`SPAN`	Если заданы ширина (свойство `width`), высота (свойство `height`) или элемент абсолютно позиционирован

Таблица 10.9 *(окончание)*

Элемент	Фильтр применяется
TABLE	Всегда
TD	Всегда
TEXTAREA	Всегда
TH	Всегда

Примечание

Фильтры можно применять и к элементам управления ActiveX, встраиваемым в HTML-страницу.

Фильтры не применяются к следующим элементам HTML-страницы: апплеты Java, IFRAME, SELECT, OPTION, P, EM, STRONG и ко всем заголовкам H1, H2 и т. д.

Применить фильтр к элементу просто: достаточно задать значение его свойства filter, следуя правилам задания свойств каскадных таблиц стилей filter: значение. Каждый фильтр, как отмечалось выше, реализует определенный алгоритм преобразования видимого отображения элемента, поэтому значение свойства filter задается в форме функции:

```
filter: имя_фильтра([параметры]);
```

а параметры фильтра, если они присутствуют, задаются с использованием синтаксиса именованных параметров функции:

```
имя_параметра=значение_параметра
```

Некоторым фильтрам требуется несколько параметров, задаваемых через запятую, а некоторым фильтрам параметры вообще не нужны, но круглые скобки должны присутствовать обязательно.

К элементу можно применить несколько фильтров одновременно. В этом случае они задаются в виде списка с пробелом в качестве разделителя:

```
<IMG ID=img1 SRC="пример1.gif"
      STYLE="filter: blur(strength=50) fliph()>
```

В данном примере к графическому изображению применяются два фильтра: первый (blur) размазывает изображение на глубину в 50 пикселов, а второй (fliph) просто зеркально его отображает в горизонтальном направлении.

В Internet Explorer реализовано большое число разнообразных фильтров. В табл. 10.10 представлены *все* фильтры с кратким описанием их воздействия на визуализацию элементов.

Таблица 10.10. *Доступные в Internet Explorer фильтры*

Фильтр	Описание
`alpha`	Устанавливает уровень непрозрачности объекта
`blendTrans`	Увеличивает или уменьшает контрастность отображения объекта
`blur`	Создает эффект размытия изображения
`chroma`	Делает прозрачными пикселы заданного цвета
`dropShadow`	Рисует сплошной силуэт объекта, смещенный в заданном направлении, создавая тем самым эффект объекта, расположенного над страницей и отбрасывающего на нее тень
`flipH`	Рисует объект в зеркальном отображении относительно горизонтальной плоскости
`flipV`	Рисует объект в зеркальном отображении относительно вертикальной плоскости
`glow`	Добавляет свечение вдоль внешних границ объекта, создавая эффект "возгорания" границ объекта
`gray`	Удаляет цветовую гамму объекта и отображает его в серых тонах
`invert`	Меняет оттенок, насыщение и яркость объекта на противоположные
`light`	Подсвечивает объект
`mask`	Закрашивает прозрачные пикселы объекта заданным цветом и создает прозрачную маску из непрозрачных пикселов объекта
`redirect`	Преобразует объект в объект `DAImage`, к которому можно применить все возможности технологии MS DirectAnimation
`revealTrans`	Показывает или скрывает объекты, используя 23 определенных в фильтре перехода
`shadow`	Рисует силуэт объекта вдоль одной из его границ в заданном направлении, создавая эффект тени
`wave`	Синусоидальное искривление объекта в вертикальном направлении, создавая эффект волнообразной поверхности
`xray`	Изменяет глубину цвета объекта и после этого отображает его в черно-белых тонах, имитируя рентгеновский снимок объекта

Ограниченные объемы книги не позволяют полностью описать каждый фильтр со всеми параметрами и нюансами использования, но некоторые, наиболее часто применяемые разработчиками Web-документов, фильтры будут описаны достаточно подробно.

Интересные динамические эффекты достигаются использованием фильтров совместно со сценариями. В процессе выполнения сценария можно устанавливать или изменять параметры применяемых к объектам фильтров,

можно назначать новые фильтры, создавать визуальные эффекты через определенные интервалы времени и делать многое другое.

Доступ к фильтрам и их параметрам в программируемых сценариях осуществляется, как обычно, с помощью объектной модели, предоставляемой браузером. В этой модели любой фильтр представляется в виде обычного объекта со своими свойствами и методами. Большинство свойств и методов соответствует параметрам фильтра, определяемым при задании фильтра в параметре `STYLE`. Некоторые свойства и методы доступны только из программируемого сценария.

Определить, какие фильтры применены к элементу, можно с помощью набора `filters` соответствующего объекта. В нем хранятся ссылки на все применяемые к элементу фильтры:

```
<IMG ID=picture SRC="picture.gif"
                STYLE="filter: wave(strength=100) gray() fliph()>
.   .   .   .   .
<SCRIPT>
strengthWave=picture.filters.wave.strength;
if( picture.filters['gray'].enabled && strengthWave >= 100) {
   picture.filters.wave.strength += 50
}
</SCRIPT>
```

В приведенном примере объект `picture` сценария соответствует элементу `<IMG>` HTML-страницы. Переменная `strengthWave` хранит значение параметра `strength` фильтра `wave`. Оператор условия проверяет, не отключался ли фильтр `gray` объекта `picture` ранее в каком-либо сценарии (значение свойства `enabled` фильтров равно `true`, если разрешено его применение к элементу). Если этот фильтр продолжает применяться, и параметр `strength` фильтра `wave` больше или равен 100, то этот параметр увеличивается на 50.

Можно ссылаться на фильтры объекта и с помощью числового индекса набора `filters`. Каждому фильтру, заданному в параметре или тэге `STYLE`, соответствует определенный числовой индекс этого набора. Все фильтры индексируются в порядке их перечисления в свойстве `filter` каскадных таблиц стилей, начиная с нуля. Так, в приведенном примере фильтру `wave` соответствует индекс 0, фильтру `gray` — 1 и фильтру `fliph` — 2. Поэтому получить значение параметра `strength` фильтра `wave` можно было бы и так:

```
strengthWave=picture.filters[0].strength;
```

Совет

Использование числового индекса при ссылке на фильтр элемента полезно, если к элементу применяются одновременно несколько фильтров одного и того же типа (естественно, с разными параметрами). Это единственный способ идентифицировать фильтры.

В наборе `style` объекта, в котором хранятся значения всех свойств каскадных таблиц стилей соответствующего объекту элемента HTML, хранится и значение свойства `filter` объекта, являющееся строкой, содержащей список применяемых к элементу фильтров. Если в предыдущем примере в сценарий добавить оператор

```
alert(picture.style.filter);
```

то в диалоговом окне, выводимом функцией `alert()`, будет отображена следующая строка:

```
wave(strength=100) gray() fliph()
```

Это свойство можно использовать для динамического изменения присоединенных к элементу фильтров. Добавить фильтр `flipv()` к элементу `picture` того же примера можно простым оператором:

```
picture.style.filter += ' flipv()';
```

Предупреждение

При добавлении новых фильтров через строковое значение свойства `filter` следует внимательно следить за синтаксисом этого свойства. При присоединении нескольких фильтров к элементу все они задаются в виде списка с пробелом в качестве разделителя, поэтому перед первым добавляемым фильтром всегда необходим пробел.

Общие свойства некоторых фильтров

Каждый фильтр имеет свойство `enabled`, которое разрешает (значение `true`) или запрещает (значение `alse`) применение присоединенного к объекту фильтра. Это свойство соответствует параметру `ENABLED` при задании фильтра в тэге HTML-элемента.

Свойство `strength` фильтров `blur`, `glow` и `wave` определяет интенсивность применения соответствующего фильтра и может изменяться от 0 до 255. Для `blur` это будет степень размытости изображения элемента, для `glow` — интенсивность "свечения" контура изображения объекта, для `wave` — амплитуда волны искажения. На рис. 10.20 показаны изображения одного и того же фрагмента текста, обработанного фильтрами `wave` различной интенсивности.

Примечание

Обратите внимание, что некоторые символы, расположенные близко к границам блока отображения элемента, при использовании фильтра `wave` преобразуются таким образом, что выходят за область отображения.

Фильтры, воздействующие на цветовую гамму элемента, имеют свойство (и, естественно, параметр) `color`. К ним относятся следующие фильтры: `chroma`, `dropShadow`, `glow`, `mask`, `shadow`. Как и в случае свойства `strength`,

свойство `color` для разных фильтров определяет разные цветовые эффекты, связанные с воздействием фильтра на объект. Для фильтров `dropShadow` и `shadow` это свойство задает цвет создаваемой тени, в фильтре `glow` оно определяет цвет "свечения" контура элемента, в фильтрах `chroma` и `mask` значением этого свойства задается цвет пикселов объекта, которые становятся прозрачными, или цвет, в который окрашиваются прозрачные пикселы объекта. Значение этого свойства задается шестнадцатеричным числом вида `#RRGGBB` или одним из зарезервированных слов, обозначающих определенные цвета, например `magenta`, `lightsteelblue` и т. д.

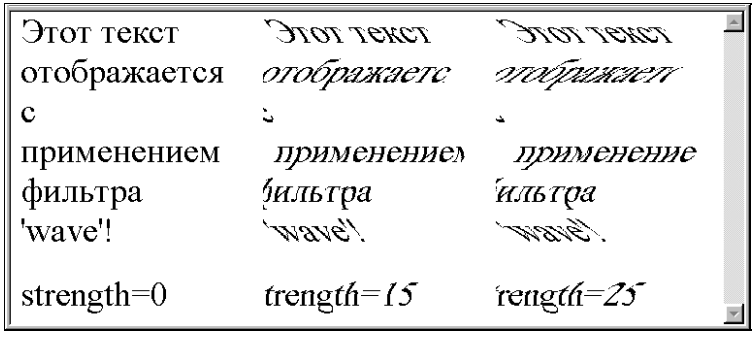


Рис. 10.20. Воздействие фильтра `wave` на отображение текста

Свойство `direction` фильтров `blur` и `shadow` определяет, соответственно, направление, в котором размывается изображение объекта, и направление падения тени от объекта. Значением этого свойства является угол, отсчитываемый от вертикали объекта по часовой стрелке с шагом в 45°. Отрицательные значения или значения, большие 360°, приводятся к эквивалентным значениям в диапазоне от 0° до 360°. Например, значение −45° соответствует 315°.

Булево свойство `add` определяет, включать (значение `true`) или не включать (значение `false`) исходное изображение объекта, к которому применяется фильтр, в отфильтрованный образ. Это свойство имеют только два фильтра: `blur` и `wave`. На рис. 10.21 показаны три графических файла буквы Ж: первый отображается без применения фильтров, второй — с применением фильтра размывания изображения `blur` интенсивности 100, третий — аналогичен второму, но с добавленным исходным изображением. В серых тонах трудно найти различия во втором и третьем изображениях, но в цвете видно, что в третьем изображении четче прорисовывается сама буква "Ж", а вдоль контура наблюдается размытие.

Перечисленные выше свойства полностью исчерпывают описание некоторых фильтров, так как они не имеют методов, или только свойств фильтров. Все такие фильтры приведены в табл. 10.11.

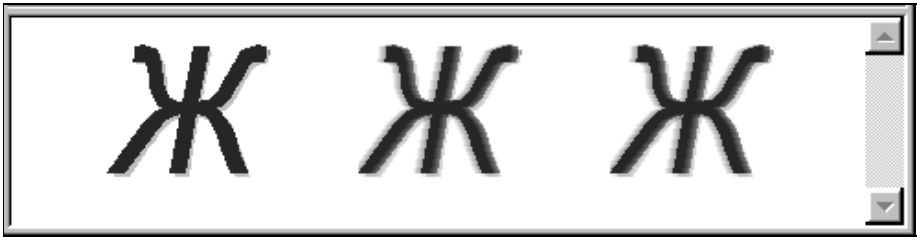


Рис. 10.21. Эффект от добавленного исходного изображения в фильтр

Таблица 10.11. *Свойства и методы некоторых фильтров*

Фильтр	Свойства	Методы
blur	add, direction, enabled, strength	Нет
chroma	color, enabled	Нет
flipH	Enabled	Нет
flipV	Enabled	Нет
glow	color, enabled, strength	Нет
gray	Enabled	Нет
invert	Enabled	Нет
light	Enabled	AddAmbient, addCone, addPoint, changeColor, changeStrength, clear, moveLight
mask	color, enabled	Нет
redirect	Enabled	ElementImage
shadow	color, direction, enabled	Нет
xray	Enabled	Нет

Не вошедшие в эту таблицу фильтры, а также методы фильтров `light` и `redirect`, описываются в следующих двух разделах.

Описание фильтров

Фильтр `wave` имеет свойства-параметры `add`, `enabled`, `strength`, `freq`, `phase` и `lightStrength`. Первые три свойства описаны выше. Свойства `freq` и `phase` определяют количество максимумов в синусоидальной волне искажения и фазу смещения волны, которая задается в процентах от фазы обычной синусоиды, имеющей начальное значение 0. Величина этого параметра, рав-

ная 100, соответствует фазе смещения в 360°. Все остальные значения лежат в интервале от 0 до 100. Например, значение `phase: 25` соответствует синусоиде с фазой смещения в 90°. Последнее свойство `lightStrength` определяет интенсивность освещения для имитации трехмерного эффекта поднятия гребней волны. На рис. 10.22 показаны изображения после обработки фильтром `wave` с различными параметрами.

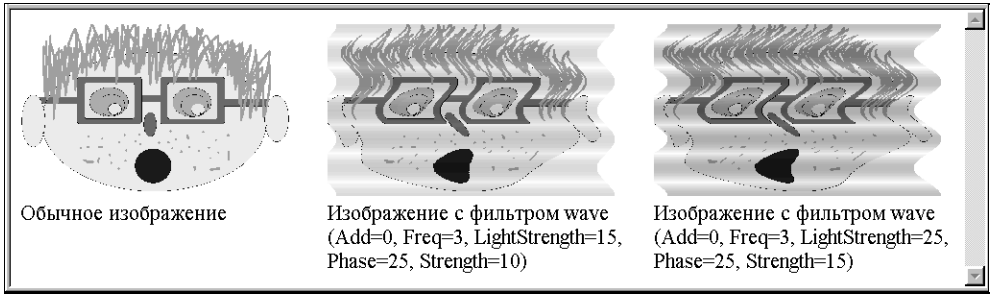


Рис. 10.22. Действие фильтра `wave`

Примечание

Чем сильнее интенсивность освещения, тем темнее горизонтальные полосы, имитирующие выпуклости и вогнутости синусоиды.

У фильтра `dropShadow` кроме свойств `color` и `enabled` имеется ряд свойств, задающих параметры падающей тени. Свойства `offX` и `offY` определяют количество пикселов в горизонтальном и вертикальном направлениях, на которые смещается тень от как бы приподнятого над плоскостью страницы объекта, а свойство `positive` определяет, должен ли фильтр создавать падающую тень от прозрачных пикселов объекта. Если значение этого свойства `true`, то тень создается от непрозрачных пикселов объекта, если `false` — от прозрачных. Положительное значение свойства `offX` смещает тень вправо, отрицательное — влево; положительное значение свойства `offY` смещает тень вниз, отрицательное — вверх. На рис. 10.23 показаны эффекты от применения этого фильтра.

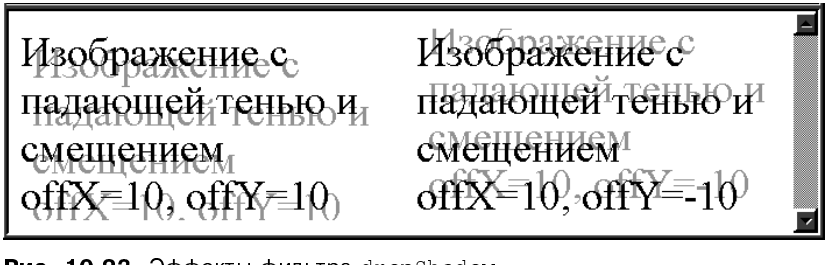


Рис. 10.23. Эффекты фильтра `dropShadow`

Совет

Если на странице есть прозрачный объект, то можно получить его тень, применив к нему фильтр `dropShadow` и установив значение его параметра `positive` равным `0`.

Фильтр `alpha` позволяет создавать эффекты прозрачности (невидимости) отображения или его части. Свойство `opacity` определяет степень прозрачности: значение `0` соответствует полной прозрачности (невидимости) изображения, а значение `100` — нормальному изображению. Параметр `style` определяет способ создания эффекта прозрачности. Его умалчиваемое значение равно `0` и соответствует изменению прозрачности всего изображения. Ненулевое значение означает, что прозрачность объекта не однородна и изменяется от пиксела к пикселу. При `style=1` параметры `startX`, `startY` и `finishX`, `finishY` определяют координаты точек прямой, на которой прозрачность равна значению параметра `finishOpacity`. Прозрачность точек, лежащих между границами объекта и этой прямой, меняется от значения, заданного свойством `opacity`, до значения, заданного свойством `finishOpacity`. При значении `style=2` степень прозрачности точек изображения меняется при перемещении из центра к эллипсу (вписанному в прямоугольный блок отображения элемента) от значения, определяемого свойством `opacity`, до значения, заданного параметром `finishOpacity`. Вне вписанного эллипса прозрачность соответствует `finishOpacity`. Если `style=3`, прозрачность аналогичным образом меняется от центра до границ изображения.

Комбинируя значения свойств фильтра `alpha` можно создавать разнообразные эффекты отображения объекта. Некоторые возможности показаны на рис. 10.24.

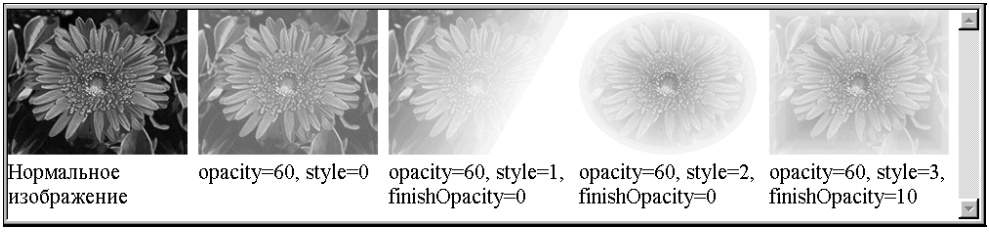


Рис. 10.24. Эффекты фильтра `alpha`

Фильтр `light`, возможно самый интересный из всех фильтров, имеет единственное свойство `enabled` и много методов, которые позволяют добавлять к изображению объекта разнообразные источники света.

Метод `addAmbient(iRed, iGreen, iBlue, iStrength)` добавляет рассеянный источник света, который представляет собой ненаправленный источник света, распространяющий параллельные лучи перпендикулярно к поверхности

страницы. Его параметры `iRed`, `iGreen`, `iBlue` задают, соответственно, насыщенность красной, зеленой и синей составляющей света и изменяются в диапазоне от 0 до 255. Параметр `iStrength` определяет интенсивность источника.

Направленный конический источник света добавляется методом

```
addCone(iX1,iY1,iZ,iX2,iY2,iRed,iGreen,iBlue,iStrength,iAngle)
```

где:

`iX1,iY1` — определяют координаты источника света в пикселах,

`iZ` — задает номер слоя,

`iX2,iY2` — определяют координаты точки направления света,

`iRed, iGreen, iBlue` — задают насыщенности соответствующих цветовых составляющих,

`iStrength` — интенсивность источника света,

`iAngle` — угол конуса распространения света.

Прямая, проходящая через точку источника света и точку направления света, является биссектрисой этого угла.

Точечный источник света добавляется к объекту методом

```
addPoint(iX1,iY1,iZ,iRed,iGreen,iBlue,iStrength)
```

Первые два параметра задают координаты источника света в пикселах, третий — номер слоя источника света. Остальные параметры имеют тот же смысл, что и в предыдущих методах добавления источников света.

На рис. 10.25 можно увидеть все три источника, присоединенные к графическому изображению.

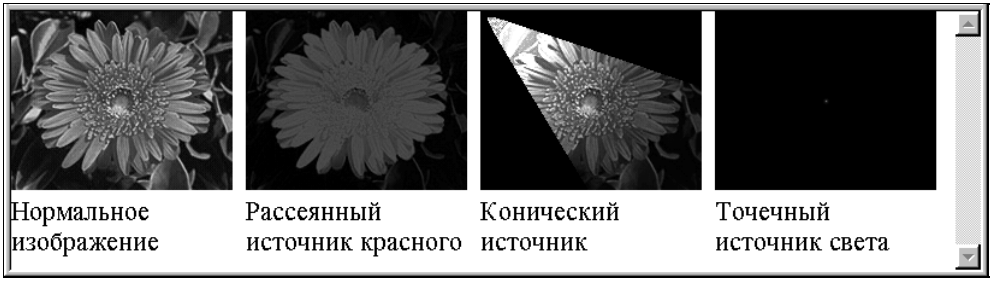


Рис. 10.25. Добавленные источники света

Все источники добавляются только из встроенного сценария. Для HTML-страницы, отображенной на рис. 10.25, исходный текст выглядит следующим образом:

Пример 10.8. Добавление источников света

```
<HEAD>
<SCRIPT>
window.onload=fnInit;
function fnInit(){
   var iX2=img2.offsetWidth/2;
   var iY2=img2.offsetHeight/2;
   var iX3=img3.offsetWidth/2;
   var iY3=img3.offsetHeight/2;
   img1.filters.light.enabled=true;
   img1.filters.light.addAmbient(255,0,0,255);
   img2.filters.light.enabled=true;
   img2.filters.light.addCone(0,0,3,iX2,iY2,255,255,255,255,20);
   img3.filters.light.enabled=true;
   img3.filters.light.addPoint(iX3, iY3, 3, 255, 255, 255, 255);
}
</SCRIPT>
</HEAD>
<DIV STYLE='position: absolute; top:0; left: 0; width: 190'>
<IMG SRC='flower.jpg'>Нормальное изображение</DIV>

<DIV STYLE='position: absolute; top:0; left: 195; width: 190'>
<IMG ID=img1 SRC='flower.jpg' STYLE="filter: light()>
Рассеянный источник красного
</DIV>

<DIV STYLE='position: absolute; top:0; left: 390; width: 190'>
<IMG ID=img2 SRC='flower.jpg' STYLE="filter:light();">
Конический источник
</DIV>

<DIV STYLE='position: absolute; top:0; left: 585; width: 190'>
<IMG ID=img3 SRC='flower.jpg' STYLE="filter:light(); height:190">
Точечный источник света
</DIV>
```

Примечание

При добавлении фильтра `light()` в параметре `STYLE` область отображения элемента становится черной, чтобы можно было видеть эффекты конического и точечного источников света.

Примечание

Точечный источник света может представлять собой один пиксел, если не подобран его цвет и цвет элемента. Лучше всего этот источник света виден на белом фоне.

Оригинальные эффекты с источниками света достигаются из встроенных сценариев. Большое количество подобных примеров можно найти на сервере разработчика фирмы Microsoft по адресу **http://msdn.microsoft.com/workshop/author/**.

Описание переходов

В Internet Explorer включено два перехода: `blendTrans` и `revealTrans`. Напомним, что переход позволяет переходить во времени от одного визуального состояния к другому, задавая длительность перехода. Переходы в Web-документах используются для создания эффекта анимации, например преобразования одного изображения в другое.

Свойство-параметр `duration` определяет интервал времени, необходимый для выполнения перехода. Его значением является действительное число с плавающей точкой в формате `секунды.миллисекунды`. В сценариях его можно изменять, если только не началось выполнение перехода.

Свойство `status`, доступное только из сценария, позволяет определить состояние перехода: `0` — переход остановлен, `1` — переход применен и `2` — переход выполняется.

Переход `blendTrans` задает постепенное исчезновение или появление изображения, увеличивая или уменьшая степень его прозрачности. Исчезновение или появление моделируется совместным использованием перехода и свойства `visibility` объекта. Следующий пример демонстрирует постепенное исчезновение одной картинки и появления другой.

Пример 10.9. Анимационный эффект

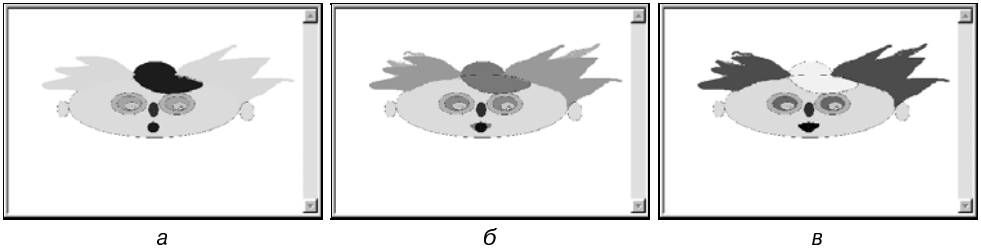
```
<HTML>
<SCRIPT language="JavaScript">
var fRunning = 0
function startTrans()
{
    if (fRunning == 0)
    {
        fRunning = 1
        img.filters.blendTrans.Apply();
        img.src = "Арнольд1.bmp";
        img.filters.blendTrans.Play()
    }
}
</SCRIPT>
<SCRIPT for="img" event="onfilterchange">
fRunning = 0
</SCRIPT>
```

```
</HEAD>
<BODY>
<IMG id="img" src="Арнольд.bmp"
style="filter:blendTrans(duration=8); width: 400; height: 250;"
onclick="startTrans()>
</BODY>
</HTML>
```

В этом примере к изображению `img` применяется переход `blendTrans`, который постепенно в течение 8 секунд делает прозрачным изображение файла `Арнольд.bmp`, а вместо него отображается содержимое файла `Арнольд1.bmp`. Инициализация перехода осуществляется щелчком мыши на изображении.

При начале выполнения перехода генерируется событие `onfilterchange`, в процедуре обработки которого задается начальное значение переменной `fRunning`.

В процедуре обработки щелчка мыши вызывается метод `applay()`, после которого можно устанавливать новые значения параметров объекта `img`. В примере задается новый файл изображения. На рис. 10.26 показано начальное изображение (*а*), промежуточное (*б*) и окончательное (*в*).



а *б* *в*

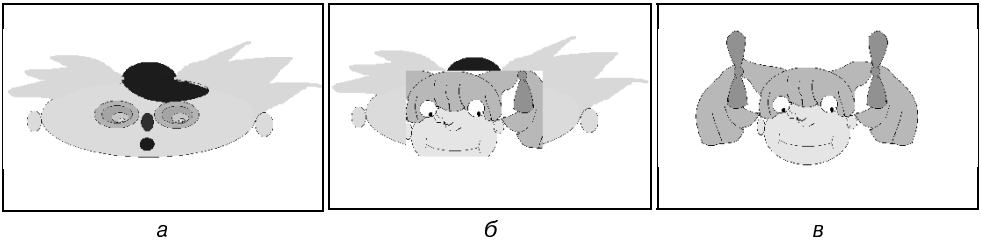
Рис. 10.26. Применение перехода `blendTrans`

Три метода переходов предназначены для управления выполнением перехода из сценария. Для их выполнения не требуется никаких параметров. Метод `apply()` "замораживает" видимое в данный момент изображение элемента и позволяет изменить значения параметров перехода и самого элемента, не требуя немедленного применения перехода. Чтобы выполнить переход после переустановки значений необходимых параметров, следует использовать метод `play()`. Остановить выполнение перехода для элемента в любой момент времени можно методом `stop()`.

Переход `revealTrans` позволяет, в отличие от перехода `blendTrans`, выполнить переходы из одного видимого состояния в другое разнообразными способами. Его свойство `transition` может иметь целое значение и определяет тип перехода. Допустимые значения представлены в табл. 10.12.

Таблица 10.12. *Типы переходов в фильтре* `revealTrans`

Переход	Отображение объекта	Переход	Отображение объекта
0	Сжимающийся прямоугольник	12	Пикселы отображаются в случайном порядке
1	Расширяющийся прямоугольник	13	Одновременно слева и справа к центру
2	Сжимающийся круг	14	Одновременно от центра налево и направо
3	Расширяющийся круг	15	Одновременно снизу и сверху к центру
4	Разворачивание снизу вверх	16	Одновременно от центра вверх и вниз
5	Разворачивание сверху вниз	17	От правого верхнего угла к левому нижнему
6	Разворачивание слева направо	18	От правого нижнего угла к левому верхнему
7	Разворачивание справа налево	19	От левого верхнего угла к правому нижнему
8	Вертикальные жалюзи	20	От левого нижнего угла к правому верхнему
9	Горизонтальные жалюзи	21	Случайными горизонтальными линиями
10	В шахматном порядке поперек	22	Случайными вертикальными линиями
11	В шахматном порядке вниз	23	Отображение пикселов случайным образом



а *б* *в*

Рис. 10.27. Применение перехода `revealTrans`

На рис. 10.27 (*а*, *б*, *в*) показаны результаты применения перехода `revealTrans` со значением параметра `transition=1` вместо перехода `blendTrans` в примере 10.9. В этом случае новый рисунок прорисовывается в течение 8 секунд

расширяющимся из центра прямоугольником. Промежуточное отображение перехода показано на рис. 10.27, *б*.

Мы привели достаточно простые примеры применения переходов. Более сложные примеры использования фильтров и переходов можно найти на сервере разработчика фирмы Microsoft по адресу **http://msdn.microsoft.com /workshop/author/**.

Связывание данных с документом

Привлекательной для разработчиков Web-приложений возможностью, реализованной в Internet Explorer 4.0 и выше, является возможность быстрого создания HTML-страниц, обрабатывающих данные из хранилища, расположенного на сервере. Необходимые для работы данные загружаются на машину пользователя, и последующая их обработка осуществляется уже на локальном уровне, не требуя дополнительного обращения к CGI-сценариям или сценариям сервера, реализованным средствами JavaScript или VBScript.

Реализованная в Internet Explorer *технология связывания данных* с HTML-страницей позволяет намного проще и дешевле решать такие задачи, как добавление и обновление данных, хранящихся на сервере. Она реализуется с помощью расширения языка HTML, а не традиционных для подобных решений HTML-форм, и в ее основе лежит разработанная фирмой Microsoft *архитектура привязки данных.*

Архитектура привязки данных

В основу решения привязки данных к HTML-странице положены четыре основных компонента:

- Объект, являющийся источником данных (Data Source Object — DSO)
- Объекты-потребители данных
- Программа-посредник, или агент, привязки данных
- Программа-посредник, или агент, повторения таблиц (табличный повторитель)

Объект-источник данных получает данные из сети и предоставляет их в распоряжение *объектам-потребителям* данных, являющимися стандартными HTML-элементами, через специальную *программу-посредника*, которая обеспечивает синхронизацию работы источника и потребителей данных. *Табличный повторитель* подключается, если обрабатывается целый набор записей из базы данных, который отображается в табличной форме на HTML-странице.

Привязка данных к HTML-странице осуществляется простым встраиванием в нее объекта-источника, который реализует открытую спецификацию, позволяющую разработчику определить:

- Используемый транспортный протокол для получения данных из сети.

  Им может быть любой протокол, например, стандартный протокол HTTP или простой файловый ввод/вывод. Также объект-источник определяет, синхронно или асинхронно осуществляется передача. Последний тип передачи является предпочтительным, так как позволяет немедленно взаимодействовать с пользователем, не дожидаясь окончания передачи данных.

- Необходимый набор данных из базы данных.

  Для этого может потребоваться задание строки ODBC-связывания и оператора SQL или простого указания URL-адреса базы данных.

- Как обрабатываются данные в сценарии.

  Так как объект-источник поддерживает данные на машине пользователя, он также определяет их сортировку и фильтрацию.

- Возможность корректировки и обновления данных.

Все перечисленные возможности объекта-источника реализуются через объектную модель, доступную для сценариев HTML-документа.

На объект-источник накладывается единственное ограничение, — он должен обеспечивать доступ к данным посредством либо OLE-DB, либо ODBC.

Объекты-потребители данных — это элементы HTML, способные отобразить на странице данные, поставляемые объектом-источником. К ним относятся как стандартные элементы HTML (`INPUT`, `SPAN`, `DIV`, `TABLE`), так и внедряемые в страницу Java-апплеты и элементы управления ActiveX. (Подробно они рассматриваются в главе 11 *"Встраиваемые компоненты"*.)

Используя специальные параметры, расширяющие стандартный HTML, Internet Explorer позволяет привязать соответствующий элемент к определенному столбцу данных из набора данных, поставляемых объектом-источником.

Элементы могут использовать какое-то одно значение из записи набора данных (например, элемент `INPUT`) или весь набор данных, отображая каждую запись в виде строки одной или нескольких таблиц (элемент `TABLE`).

Работа программы-посредника привязки данных и табличного повторителя совершенно невидима для пользователя. При загрузке страницы они определяют, содержит ли она объекты-источники данных и соответствующие им объекты-потребители. При наличии подобных объектов эти программы автоматически синхронизируют их работу. Когда объект-источник получает дополнительную порцию данных из сети, программы-посредники автоматически пересылают их соответствующим потребителям данных. И наоборот, если пользователь изменил данные в объекте-потребителе, то они автоматически пересылаются в объект-источник, который, в свою очередь, пересылает их в базу данных.

Разработчик страницы с привязанными к ней данными может отслеживать работу программы-посредника, перехватывая и обрабатывая события, генерируемые посредником при выполнении им определенных действий.

Для обеспечения привязки данных к HTML-странице в Internet Explorer включена поддержка дополнительных нестандартных параметров элементов-потребителей данных: DATASRC, DATAFLD, DATAFORMATAS и DATAPAGESIZE. Задание значений этих параметров для объекта-потребителя связывает его с определенным объектом-источником данных на странице и обеспечивает их правильное отображение.

Параметр DATASRC определяет имя источника данных, задаваемого в виде строки, начинающейся с символа (#), за которым сразу же следует имя объекта-источника, определенное в его параметре ID.

Параметр DATAFLD задает имя поля в записи, данные из которого привязываются к определенному элементу-потребителю данных на HTML-странице.

Обычно эти два параметра используются совместно, определяя отображение набора данных и конкретных полей его записей на странице. В следующем примере весь набор данных, поставляемый объектом-источником с именем myDSO, привязан к табличному потребителю данных, в ячейках которого отображаются значения поля author всех записей набора данных:

```
<TABLE DATASRC="#myDSO">
  <TR><TD><INPUT TYPE=TEXTBOX DATAFLD="author"></TD></TR>
</TABLE>
```

Примечание

Так как источник данных присоединен к табличному элементу, то при отображении в нем данных табличный повторитель создаст на странице таблицу, содержащую столько строк, сколько записей содержится в наборе данных, поставляемых объектом-источником.

При привязке данных к элементу, отображающему одно единственное значение, следует задавать оба параметра, как показано в следующем примере:

```
<SELECT DATASRC="#myDSO" DATAFLD="publisher">
   <OPTION>БХВ
   <OPTION>Питер
   <OPTION>Microsoft Press
</SELECT>
```

Здесь при выборе пользователем одной из опций в раскрывающемся списке программа-посредник установит текущую запись, поле publisher которой содержит выбранное значение.

Для элементов-потребителей единственного значения можно указать формат используемых данных в параметре DATAFORMATAS. В настоящее время значе-

нием этого параметра может быть только одно из двух значений: HTML и TEXT. Первое соответствует данным в формате HTML, второе — обычным текстовым данным. Данные отображаются в зависимости от заданного формата. Ниже представлены примеры привязки данных к элементам DIV и TEXTAREA с указанием формата:

```
<DIV DATASRC="#myDSO" DATAFLD="author" DATAFORMATAS="HTML"></DIV>
<TEXTAREA DATASRC="#myDSO" DATAFLD="book_name"
          DATAFORMATAS="TEXT"></TEXTAREA>
```

Примечание

Если задан определенный формат данных для полей, то в записях набора данных эти данные должны соответствовать соответствующим форматам.

Для оптимизации отображения набора данных, содержащего большое число записей, следует использовать параметр DATAPAGESIZE, определяющий максимальное число записей, отображаемых в таблице. По умолчанию в таблице отображаются все записи привязанного набора данных, и если записей много, то они могут не поместиться в окне отображения браузера. Возможность ограничения числа выводимых одновременно записей позволяет создавать более удобные в использовании страницы. Переход к отображению следующей порции записей осуществляется методом nextPage() объекта TABLE, предыдущей — методом previousPage().

Объекты-источники данных

При разработке приложений с использованием внешних данных следует, прежде всего, определить, в каком формате будут храниться эти данные. Они могут представляться в виде простого текстового файла с запятой в качестве разделителя (расширение CSV), HTML-файла или таблицы базы данных, поддерживающей технологию OLE-DB или ODBC (например, SQL Server, MS Access, FoxPro).

После того как определен формат представления внешних данных, необходимо выбрать подходящий объект-источник данных (DSO) и встроить его в страницу HTML. В Internet Explorer объектом-источником данных может быть апплет Java, элемент управления ActiveX или сама динамически подключаемая библиотека MSHTML.DLL браузера, реализующая его объектную модель.

Внешние источники данных

Internet Explorer поставляется с двумя элементами управления ActiveX. Первый, TDC, осуществляет доступ к данным, представленным текстовым форматом с разделителем (файл CSV), второй, RDS.DataControl, позволяет работать с таблицами реляционных баз данных.

Элементы управления ActiveX являются законченными программными объектами, которые позволяют получать/изменять значения своих свойств внешним программам или другим объектам. При их установке на компьютере пользователя в реестре Windows производится соответствующая запись, идентифицирующая каждый установленный в системе элемент управления ActiveX уникальным кодом. На HTML-страницу они встраиваются тэгом `<OBJECT>`, а тэг `<PARAM>` позволяет установить значения свойств элемента ActiveX.

Примечание

Более подробно элементы ActiveX и Java-апплеты рассматриваются в главе 11 *"Встраиваемые компоненты"*.

TDC (Tabular Data Control) является простым источником данных для HTML-страницы и используется в следующих случаях:

- Для отображения простых наборов данных, хранящихся в обычных текстовых файлах с разделителями.
- Если необходимо предоставить пользователю возможность просмотра простых наборов данных в режиме off-line. При работе с этим источником данные пересылаются с сервера на машину пользователя и сохраняются в кэш-областях. После завершения сеанса связи их можно просмотреть в автономном режиме.
- Если необходимо предупредить возможность прямого использования системы управления базами данных сервера (DBMS). В этом случае создается простой текстовый файл, представляющий набор данных из базы, доступ к которому и осуществляется объектом TDC. Заметим, что современные DBMS позволяют представлять данные в формате текстового файла с разделителем.

Его свойства и методы позволяют определять файл данных, а также осуществлять их фильтрацию и сортировку. Свойства, определяющие файл данных и его структуру, представлены в табл. 10.13.

Таблица 10.13. *Файловые свойства элемента управления TDC*

Свойство	Описание
`CharSet`	Определяет набор символов, используемый в файле данных. По умолчанию применяется набор latin1
`DataUrl`	Задает URL-адрес файла данных
`EscapeChar`	Определяет символ ESC в файле данных. Умалчиваемого значения нет
`FieldDelim`	Определяет символ-разделитель полей. По умолчанию используется запятая (,)

Таблица 10.13 *(окончание)*

Свойство	Описание
`Language`	Определяет язык, используемый для генерации файла данных. По умолчанию eng-us
`TextQualifier`	Задает необязательный символ, которыми окружено каждое поле
`RowDelim`	Определяет символ-разделитель рядов данных. По умолчанию используется символ новой строки
`UserHead`	Определяет, является ли первая строка файла данных информацией для заголовка таблицы (значение `TRUE`). По умолчанию значение равно `FALSE` — первая строка является строкой данных

Для работы с данными элемент управления TDC имеет два свойства и один метод.

Свойство `Filter` содержит выражение, включающее имя столбца, который должен быть отфильтрован, критерий фильтрации и значение, с которым должно сравниваться содержимое указанного столбца. В качестве критерия фильтрации используются последовательности символов `=`, `>=`, `<=`, `!=`, смысл которых очевиден.

Значением свойства `Sort` является список имен столбцов, разделенных точкой с запятой, по которым необходимо отсортировать данные файла в таблице. Знак (`+`) перед именем столбца означает сортировку в возрастающем порядке, минус (`-`) — в убывающем.

Действительная сортировка данных в таблице осуществляется методом `Reset()` элемента TDC.

Элемент управления RDS.DataControl (Remote Data Service) по своим возможностям превосходит элемент управления TDC. Он позволяет работать с наборами данных из реляционных таблиц, и его рекомендуется применять в случаях:

- Работы с данными, хранящимися в таблицах баз данных, поддерживающих технологии OLE-DB или ODBC
- Если необходимо получить набор данных с помощью SQL
- Если есть необходимость предоставить пользователю возможности корректировки и добавления данных непосредственно в таблицы баз данных
- Предоставления работы с данными в реальном режиме времени

Для стандартной работы с этим объектом достаточно задания трех его свойств: `Server`, `Connect` и `SQL`. Первое свойство задает URL-адрес сервера, на котором расположена база данных. Свойство `Connect` определяет системное имя набора данных (DSN), определенное на машине-сервере. Оно также требует, при необходимости, задания идентификатора пользователя и пароля доступа. Последнее свойство `SQL` определяет запрос к базе данных в виде

оператора SQL. Пример встраивания объекта RDS на HTML-странице показан ниже:

```
<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
          ID=dsoComposer HEIGHT=0 WIDTH=0>
   <PARAM NAME="Server" VALUE="http://bhvserver">
   <PARAM NAME="Connect" VALUE="dsn=library;uid=guest;pwd=">
   <PARAM NAME="SQL" VALUE="select * from title">
</OBJECT>
```

Здесь объект-источник привязывает набор данных, состоящий из всех полей таблицы `title` базы данных, расположенной на сервере по адресу **http://bhvserver**. Он связывается с ней посредством определенного в системе сервера имени источника данных `library`.

Этот объект имеет большой набор свойств и методов, позволяющий выполнять сложную фильтрацию и сортировку данных, а также корректировать и добавлять их в базу данных. Более подробно ознакомиться с объектом RDS.DataConnect можно на сервере Microsoft по адресу **http://www.microsoft.com/isapi/ado/**.

Сортировка данных в таблице

Предположим, что в файле `author.txt` содержатся данные по книгам, выпускаемым каким-то издательством. Этот файл представляет собой файл данных с запятой в качестве разделителя полей в записи. Каждая запись начинается с новой строки. В первой строке содержатся имена полей в записи. Фрагмент начала файла представлен ниже:

```
series, author, title
Мастер, Нортон П., Разработка приложений в Access 97
В подлиннике, Браун М., HTML 3.2
В подлиннике, Нортон Р., Windows 98
Мастер, Мюррей У., Создание переносимых приложений для Windows
. . . . . .
```

С помощью объекта-источника TDC и элемента `<SELECT>` можно организовать динамическое отображение в таблице данных из такого файла. В раскрывающемся списке `<SELECT>` пользователь выбирает необходимую ему серию книг и в таблице динамически изменяется содержимое, чтобы отобразить только книги выбранной серии. Исходный текст HTML-документа можно увидеть в примере 10.10.

Пример 10.10. Сортировка и фильтрация данных в таблице

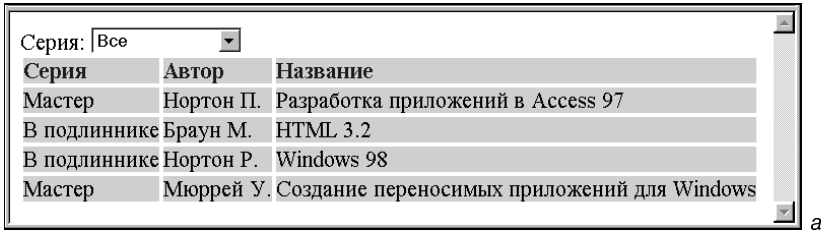
```
<STYLE TYPE="text/css">
DIV {background-color: lightgrey}
```

```
B {color: blue}
</STYLE>
<SCRIPT>
function tChange() {
  ser=new Array(3)
  ser[1]='Мастер'
  ser[2]='В подлиннике'
  authorDSO.Filter=''
  fld=s_change.value
  if(fld != 0) {
   authorDSO.Filter+='series='+ser[fld]
  }
   authorDSO.Reset()
}
function a_sort() {
  authorDSO.Sort='+author'
  authorDSO.Reset()
}
</SCRIPT>
</HEAD>
<OBJECT id="authorDSO"
        CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
       <PARAM NAME="DataURL" VALUE="author.txt">
       <PARAM NAME="UseHeader" VALUE="True">
</OBJECT>
<LABEL>Серия: </LABEL>
<SELECT ID="s_change" onchange="tChange()>
  <OPTION SELECTED VALUE=0>Все
  <OPTION VALUE=1>Мастер
  <OPTION VALUE=2>В подлиннике
</SELECT>
<TABLE ID="elemtbl" datasrc="#authorDSO">
   <THEAD><TR>
     <TD><DIV ID="series"><B>Серия</B></DIV></TD>
     <TD><DIV ID="author" onclick='a_sort()'><B>Автор</B></DIV></TD>
     <TD><DIV ID="title"><B>Название</B></DIV></TD>
   </TR></THEAD>
   <TBODY>
     <TR>
       <TD><DIV DATAFLD="series"></DIV></TD>
       <TD><DIV DATAFLD="author"></DIV></TD>
       <TD><DIV DATAFLD="title"></DIV></TD>
     </TR>
   </TBODY>
</TABLE>
```

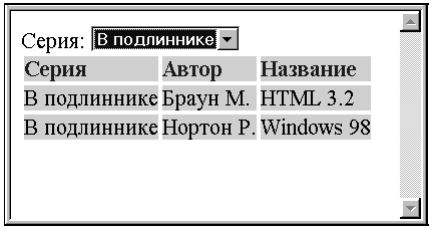
Объект-источник данных TDC включается в страницу тэгом `<OBJECT>`. Значения его свойств задаются в тэгах `<PARAM>`. Обязательно должен быть определен файл исходных данных. В нашем примере — это упомянутый выше файл `author.txt`. В параметре `CLASSID` задается уникальный регистрационный номер элемента управления ActiveX, в нашем примере элемента TDC.

При задании тела таблицы для ссылки на поля записей используются их символические имена, определенные в первой строке файла данных. Это можно делать, так как значение свойства `UseHeader` установлено равным `true`.

При выборе значения в раскрывающемся списке `s_change` генерируется событие `onchange` этого элемента и вызывается функция `tChange()`, которая устанавливает значение свойства `Filter` объекта `authorDSO` и выполняет метод `Reset()` для обновления отображаемых данных в соответствии с заданным критерием фильтрации. Последний задается только в случае, если выбрана конкретная серия книг (значение свойства `value` объекта `s_change` не равно нулю) в виде строки `series=Мастер` или `series=В подлиннике` в зависимости от осуществленного пользователем выбора. Если сделан выбор отобразить книги всех серий, то значение критерия фильтрации равно пустой строке. После этого вызывается метод `Reset()` объекта-источника данных, и обновляется содержимое таблицы. На рис. 10.28 *а*, *б* представлены два состояния таблицы, отображающей исходный файл данных `author.txt`.



а



б

Рис. 10.28. Сортировка (*а*) и фильтрация данных (*б*) на HTML-странице

Примечание

Динамическое отображение таблицы полезно, когда она содержит большое число записей, логически разбитых на мелкие группы, а пользователю как раз и необходимо видеть эти мелкие группы данных.

Источник данных MSHTML

Разработчики Web-документов могут задавать непосредственно в HTML-документе необходимый набор данных и обрабатывать его, используя сам браузер, точнее, — программы из его библиотеки динамической компоновки MSHTML.DLL. Следует отметить, что такой подход к отображению набора данных позволяет работать с ним исключительно как с набором данных, предоставляющим доступ только на чтение. Добавлять записи или корректировать их со страницы HTML нельзя. Отдельная HTML-страница, в которой определяется набор данных, называется страницей данных.

Когда MSHTML используется в качестве источника данных, она анализирует содержимое HTML-страницы и определяет элементы с параметрами `ID`. Содержимое разных элементов с одинаковыми идентификаторами определяет столбец набора данных, т. е. поле с именем, соответствующим значению параметра `ID`, разных записей набора. Одинаковые идентификаторы могут быть у разных элементов. Для MSHTML это не имеет никакого значения, — все они будут определять один столбец набора данных. Значение параметра `ID` используется в качестве значения параметра `DATAFLD` элемента, к которому привязываются данные.

Набор данных из предыдущего примера можно определить в файле `author.htm` следующим образом:

```
<DIV><SPAN ID=series>Мастер</SPAN>
<SPAN ID=author>Нортон П.</SPAN>
<SPAN ID=title>Разработка приложений в Access 97</SPAN></DIV>
<DIV><SPAN ID=series>В подлиннике</SPAN>
<SPAN ID=author>Браун М.</SPAN>
<SPAN ID=title>HTML 3.2</SPAN></DIV>
<DIV><SPAN ID=series>В подлиннике</SPAN>
<SPAN ID=author>Нортон Р.</SPAN>
<SPAN ID=title>Windows 98</SPAN></DIV>
<DIV><SPAN ID=series>Мастер</SPAN>
<SPAN ID=author>Мюррей У.</SPAN>
<SPAN ID=title>Создание переносимых приложений для Windows</SPAN>
</DIV>
```

Когда определен набор данных, можно использовать тэг `<OBJECT>` для привязки данных к HTML-странице:

```
<OBJECT ID=authorDSO DATA="author.htm">
</OBJECT>
```

Теперь набор данных можно обычным способом отображать в HTML-элементах. Динамическая обработка подобных наборов данных требует написания скриптлетов (программных компонентов, используемых в сценариях страниц HTML и написанных на языке VBScript или JScript) и выходит за рамки

настоящей книги. На сервере Microsoft по адресу **http://msdn.microsoft.com /workshop/author/databind/** можно найти пример динамической обработки данных, присоединенных к странице с помощью MSHTML.

Динамический HTML в Netscape Navigator

Заключительный раздел этой главы посвящен реализации динамического HTML в популярном браузере Netscape Navigator 4.0. Для создания динамических эффектов в нем также используются каскадные таблицы стилей, язык сценариев JavaScript, язык HTML и объектная модель документа, не соответствующая рекомендациям Консорциума W3. В ней для позиционирования HTML-элементов, представляемых объектами в модели документа, используются слои, а иерархия объектов отличается от рекомендованной Консорциумом W3. Так как эта объектная модель родилась из языка сценариев JavaScript, то мы будем называть ее *объектной моделью сценария*, чтобы не путать с объектной моделью документа — термином, предложенным Консорциумом W3.

В этой связи следует отметить, что многие динамические эффекты, реализованные в объектной модели документа Internet Explorer, не выполняются при просмотре HTML-документа в браузере Netscape Navigator, как, впрочем, и наоборот.

В данном разделе также приводятся отличия реализации динамического HTML в Netscape Navigator и Internet Explorer, а также рекомендации по созданию динамических HTML-документов, которые могут одинаково, или почти одинаково, отображаться в обоих браузерах.

Примечание

В этом разделе нам придется часто упоминать названия двух популярных браузеров. Для экономии места на странице и времени читателя при чтении длинных названий везде в этом разделе браузер Internet Explorer будет обозначаться IE, а Netscape Navigator — NN.

Применение каскадных таблиц стилей

Каскадные таблицы стилей в NN реализованы в соответствии с рекомендациями REC-CSS1 Консорциума W3 и их применение ничем не отличается от применения в IE. Единственное отличие заключается в том, что в NN реализовано два способа задания правил каскадных таблиц стилей. Первый способ соответствует рекомендациям Консорциума — задавать все правила в тэге `<STYLE>`, расположенном в головном разделе документа, в соответствии с предложенным там же синтаксисом. В NN добавлена возможность задания правил в тэге `<STYLE>` с использованием синтаксиса JavaScript и объектной модели сценариев.

Рассмотрим традиционное задание правила, определяющего цвет заголовка первого уровня:

```
<STYLE TYPE="text/css">
<!--
  H1 {color: red}
-->
</STYLE>
```

При использовании синтаксиса JavaScript это же правило будет выглядеть так:

```
<STYLE TYPE="text/javascript">
  document.tags.H1.color = 'red'
</STYLE>
```

Читатель, прочитавший раздел, посвященный объектной модели документа, или просто проницательный читатель без труда поймет, что объект `tags`, являющийся подобъектом объекта `document`, содержит все объекты, соответствующие типам HTML-элементов документа, а написанный оператор JavaScript присваивает свойству `color` объекта `H1` строку со значением `red`.

Как и в объектной модели IE, объект `document` соответствует всему документу и содержит все остальные объекты модели (является их родителем).

Сложные названия свойств каскадных таблиц стилей, содержащие дефисы, при использовании синтаксиса JavaScript преобразуются следующим образом: из названия просто убираются дефисы. Например, свойству `line-height` будет соответствовать название `lineheight`.

Обратим внимание на отличие в задании значения параметра `TYPE` тэга `<STYLE>`. Его значением является "`text/javascript`", что сообщает браузеру об использовании синтаксиса JavaScript. При этом содержимое тэга `<STYLE>` не надо комментировать, так как браузер, проанализировав значение параметра `TYPE`, просто пропустит весь тэг, если он не поддерживает этот синтаксис.

Примечание

В документации фирмы Netscape обычный способ задания правил каскадных таблиц стилей называется *CSS-стиль*, а с использованием синтаксиса JavaScript — *JavaScript-стиль* или *стиль сценария*. При необходимости мы также будем использовать эту терминологию.

В стиле сценария можно использовать оператор `with(объект)`, задающий объект, к которому применяются свойства и методы из тела этого оператора. Он удобен при определении нескольких значений свойств элемента одного типа. Следующий фрагмент определяет свойства заголовка первого уровня:

```
with(tags.H1) {
  color='red';
```

```
  textTransform = 'uppercase';
}
```

В каскадных таблицах стилей широко используется определение классов, позволяющих легко изменять отображение элемента одного и того же типа. В стиле сценария классы задаются следующим образом:

```
classes.GREENBOLD.all.color = "#44CC22"   // Эти два оператора определяют
classes.GREENBOLD.all.fontWeight = "bold" // класс GREENBOLD, применимый
                                          // ко всем элементам документа
classes.RED1.P.color = "red";        // Эти два оператора определяют
classes.RED1.P.fontWeight = "bold";  // класс RED1 элементов P

ids.BLUE1.color = "blue"; // Этот оператор определяет класс элемента
                          // с параметром ID="BLUE1"
```

Свойства контекстных элементов (правила с контекстными селекторами) задаются с помощью функции `contextual()`, параметрами которой служит список, определяющий контекстный селектор:

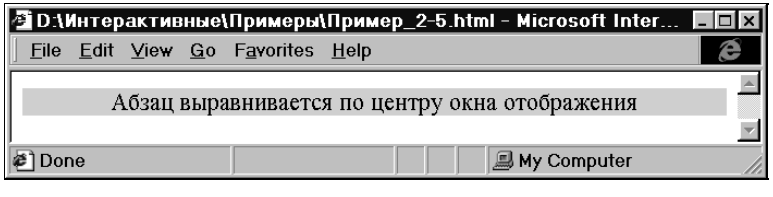
```
contextual(tags.H1, tags.EM).color = "green";
```

Приведенный оператор эквивалентен следующему правилу, записанному в CSS-стиле:

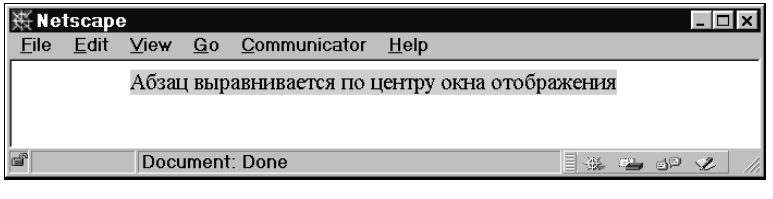
```
H1 EM {color: green;}
```

Оба браузера отображают HTML-элементы с присоединенными стилями в целом одинаково, но существуют некоторые различия в трактовках некоторых свойств.

В частности, наблюдается отличие при закрашивании фона абзаца, раздела и всех заголовков. В IE закрашивается все видимое в окне браузера пространство, относящееся к этим элементам, а в NN — только часть, в которой расположен текст. Это расхождение иллюстрируется на рис. 10.29.

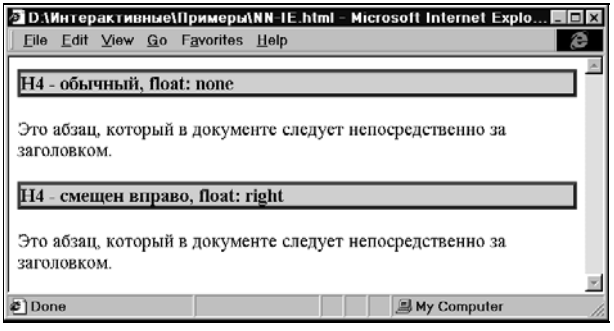


а

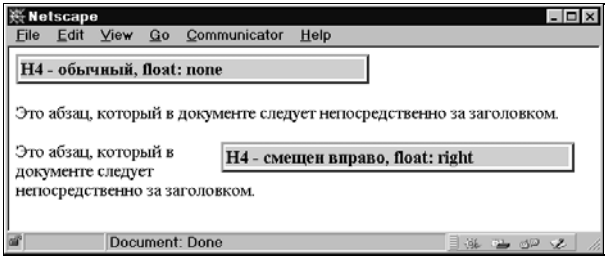


б

Рис. 10.29. Отображение абзаца в браузерах IE (*а*) и NN (*б*)



а



б

Рис. 10.30. Отображение "плавающих" блоковых элементов в браузерах IE (*а*) и NN (*б*)

По-разному применяется свойство `float`, что связано с различной интерпретацией перевода элементов в разряд "плавающих". В IE плавающими элементами можно сделать только встроенные элементы, т. е. элементы, которые появляются внутри блоковых. Например, элемент `<SPAN>` в блоковом элементе `<P>`. В NN плавающим можно сделать любой блоковый элемент, в том числе и абзац, и заголовок. В этом случае, например, заголовок со значением `right` свойства `float` будет выделен из нормального последовательного потока отображения элементов страницы и прижат вправо, а последующий абзац будет отображаться перед ним, обтекая его справа. На рис. 10.30 показано отображение двумя браузерами следующей страницы с установленным свойством `float` второго заголовка:

```
<HEAD>
<STYLE TYPE="text/css">
   H4 { width:70%;
        border-style:outset;
        border-width:2pt;
        border-color:green;
        background-color:rgb(70%, 90%, 80%);
        font-weight:bold;
      }
   H4.FLOATRIGHT {float:right;}
</STYLE>
</HEAD>
```

```
<H4>H4 — обычный, float: none</H4>
<P>Это абзац, который в документе следует непосредственно
   за заголовком.</P>
<H4 CLASS = FLOATRIGHT>H4 — смещен вправо, float: right</H4>
<P>Это абзац, который в документе следует непосредственно
   за заголовком.</P>
```

Обратите внимание на то, что установка значения свойства `width` заголовка совершенно не влияет на его отображение в окне IE, но учитывается при отображении браузером NN. Это связано с различной реализацией позиционирования элементов в этих браузерах.

Позиционирование и объектная модель сценария

В каскадных таблицах стилей свойство `position` элемента определяет его положение в плоскости страницы относительно других элементов страницы. Если элемент позиционирован (значение его свойства `position` равно `relative` или `absolute`), то значение свойства `z-index` определяет, какие позиционированные элементы он перекрывает при отображении и какие элементы перекрывают его. Значение свойства `visibility` определяет, виден или не виден этот элемент в окне браузера.

В Internet Explorer, в соответствии с рекомендациями Консорциума W3 по объектной модели документа, все перечисленные свойства являются обычными свойствами объектов, соответствующих элементам HTML-документа.

В Netscape Navigator, в отличие от Internet Explorer, позиционированный элемент реализуется в виде объекта `layer` со своими свойствами и методами. В этом заключается различие между IE и NN. Они по-разному отражают в объектных моделях позиционированные элементы, что приводит к функциональной несовместимости браузеров при обработке страниц, содержащих сценарии динамического изменения содержимого документа с использованием позиционирования.

В NN каждый позиционированный элемент считается слоем и отображается в объектной модели сценария в виде объекта `layer`, порождаемого корневым объектом `document`. В документации по динамическому HTML в Netscape Navigator на сервере фирмы Netscape термины "позиционированный элемент", "позиционированный блок" и "слой" являются синонимами и представляют позиционированный элемент вместе со всем его содержимым как одно целое.

Слои могут содержать другие слои. Если в каком-либо слое определен еще один слой, то этот слой является вложенным для первого слоя, который, в свою очередь, является содержащим слоем для вложенного. Эти понятия важны для определения положения абсолютно и относительно позиционированных слоев, задаваемых свойствами `top` и `left`.

Положение абсолютно позиционированного слоя (значение `absolute` свойства `position`) отсчитывается от левого верхнего угла содержащего его слоя, тогда как относительно позиционированный слой (значение `relative` свойства `position`) располагается относительно текущего положения в документе.

В NN для задания слоев можно использовать тэги, являющиеся расширением HTML. Тэг `<LAYER>` определяет слой, а тэг `<ILAYER>` определяет вложенный слой. Однако следует учитывать, что только браузер NN поддерживает эти расширения. Поэтому при разработке HTML-документов лучше ими не пользоваться. Здесь они упомянуты исключительно для полноты изложения позиционирования в NN.

Все свойства, применяемые в каскадных таблицах стилей для позиционированных элементов, применимы и для слоев в NN. Размеры слоя определяются значениями свойств `width` и `height`, свойство `visibility` позволяет отображать или не отображать слой и т. д.

Таким образом, статическое использование слоев дает одинаковый эффект в обоих браузерах, если только относительно позиционированные слои вложены в абсолютно позиционированные. Если это не так, то могут обнаружиться отличия при отображении документа в IE и в NN. В каждом случае следует просто проверять отображение документа в обоих браузерах.

Существенные отличия, как отмечалось выше, появляются при реализации динамических эффектов, что связано с разными объектными моделями, используемыми этими браузерами.

Объект `document` имеет свойство-набор `layers`, в котором содержатся ссылки на объекты `layer`, соответствующие созданным в документе слоям. Если известен идентификатор слоя `idName`, определенный в параметре `ID` слоя, то получить доступ к объекту, соответствующему этому слою, можно одним из следующих способов:

```
document.idName
document.layers[idName]
document.layers[index]
```

В последнем случае используется индекс слоя в наборе `layers`. Все слои документа располагаются в этом наборе в порядке, определяемом значениями их свойства `z-index`: сначала идут слои, располагающиеся при отображении дальше всего от пользователя, затем те, которые перекрывают предыдущие, и т. д.

Каждый объект, соответствующий слою в документе, имеет свойство `document`, которое, в свою очередь, имеет свойство-набор `layers`, в котором содержатся ссылки на все вложенные в данный слой слои. Определить число вложенных слоев можно с помощью свойства `length` набора `layers`:

```
document.layers[idNameLayer].document.layers.length
```

Каждый объект-слой имеет свойство, соответствующее примененному к нему свойству каскадных таблиц стилей. Например, определить в сценарии значение свойства `top` слоя с именем `layer1` можно следующим оператором:

```
layer1Top=document.layer1.top
```

Все допустимые в объектной модели сценария браузера NN свойства слоев перечислены в табл. 10.14.

Таблица 10.14. *Свойства объектов-слоев*

Свойство	Описание
`document`	Представляет объект `document` слоя, содержащий все вложенные в слой объекты: изображения, апплеты, слои и т. д.
`name`	Идентификатор слоя, заданный в параметрах `ID` или `NAME`
`left`	Определяет значение свойства `left`
`top`	Определяет значение свойства `top`
`pageX`	Смещение слоя по горизонтали относительно страницы
`pageY`	Смещение слоя по вертикали относительно страницы
`zIndex`	Определяет значение свойства `z-index`
`visibility`	Определяет значение свойства `visibility`

`clip.top`
`clip.left`
`clip.bottom`
`clip.right`
`clip.width`
`clip.height`
	Определяют параметры прямоугольника, представляющего видимое отображение слоя. Все, что не входит в заданный прямоугольник, остается невидимым при отображении слоя в окне браузера. Соответствуют свойству `clip` каскадных таблиц стилей

`background.src`	Задает графическое изображение, используемое в качестве фона слоя
`bgColor`	Определяет цвет фона слоя
`siblingAbove`	Определяет вложенный в тот же слой-родитель объект-слой, перекрывающий заданный. Если такого слоя нет, то значение равно `null`
`siblingBelow`	Определяет вложенный в тот же слой-родитель объект-слой, перекрываемый заданным слоем. Если такого слоя нет, то значение равно `null`
`above`	Определяет объект-слой из всего множества слоев документа, перекрывающий заданный. Если такого слоя нет, то значение равно объекту `window`

Таблица 10.14 *(окончание)*

Свойство	Описание
`below`	Определяет объект-слой из всего множества слоев документа, перекрываемый заданным слоем. Если такого слоя нет, то значение равно `null`
`parentLayer`	Определяет объект-слой, содержащий заданный слой. Если такого слоя нет, то значение равно объекту `window`
`src`	Относительный или абсолютный URL-адрес файла содержимого слоя

В объектной модели сценария определены методы объекта-слоя, которые позволяют модифицировать его из сценария JavaScript. Набор допустимых методов един и применяется к слоям, созданным как с помощью тэгов `<LAYER>` и `<ILAYER>`, так и с помощью CSS-стиля. В табл. 10.15 перечислены все методы объекта `layer`.

Таблица 10.15. *Методы объектов-слоев*

Метод	Описание
`moveBy(dx,dy)`	Перемещает слой на `dx` пикселов вправо и `dy` пикселов влево относительно его текущего положения
`moveTo(x,y)`	Действие аналогично действию, осуществляемому при установке новых значений свойств `top` и `left`
`moveToAbsolute(x,y)`	Действие аналогично действию, осуществляемому при установке новых значений свойств `pageX` и `pageY`
`resizeBy(dwidth,dheight)`	Изменяет размеры видимого отображения слоя на заданную ширину и высоту. Содержимое слоя не прорисовывается. Действие аналогично действию, осуществляемому при установке новых значений свойств `clip.width` и `clip.height`
`resizeTo(width,height)`	Изменяет размеры видимого отображения слоя до заданной ширины и высоты. Содержимое слоя не прорисовывается. Действие аналогично действию, осуществляемому при установке новых значений свойств `clip.width` и `clip.height`
`moveAbove(layer)`	Изменяет отображение слоя таким образом, что он располагается поверх заданного параметром метода слоя

Таблица 10.15 *(окончание)*

Метод	Описание
`moveBelow(layer)`	Изменяет отображение слоя таким образом, что заданный параметром метода слой располагается поверх слоя, к которому применяется метод
`load(sourcestring, width)`	Изменяет содержимое слоя на содержимое файла, заданного параметром `sourcestring`, и устанавливает ширину слоя в соответствии с параметром `width`

В сценарии можно динамически создать новый слой в документе. Для этого следует определить новый объект `layer` с помощью конструктора `Layer()`:

```
bluelayer = document.bluelayer;
newbluelayer = new Layer(300, bluelayer);
```

Первый параметр конструктора — ширина слоя, а необязательный второй — родительский слой для создаваемого слоя.

После того как новый объект-слой создан, определить его содержимое можно установкой значения свойства `src` или выполнением метода `load()`, а можно и непосредственно записать содержимое в объект `document` слоя.

Примечание

Создать новый слой можно только после полной загрузки документа в браузер. Объект `document` только одного слоя может быть открыт для записи. Чтобы записать содержимое следующего слоя, следует закрыть объект `document` предыдущего.

Динамическое позиционирование

В данном разделе рассматривается пример создания HTML-страницы с динамическим позиционированием графического изображения, который одинаково отображается в браузерах IE и NN.

Прежде всего, разработаем две функции, определяющие, в каком браузере происходит отображение страницы. Функция `isNN()` возвращает значение `true`, если браузером является Netscape Navigator версии 4.0 и выше; функция `isIE()` возвращает значение `true`, если используется Internet Explorer. Тексты этих функций приведены ниже:

```
function isNN()
{
   // Получаем информацию о браузере
   appName= navigator.appName;
```

```
   appLongVer = navigator.appVersion;
   appVer = appLongVer.substring(0, 1);

   // Проверяем Netscape Navigator версии 4+.
   if ((appName == "Netscape") && (appVer >= 4)) return true;
   return false;
}

function isIE()
{
   // Получаем информацию о браузере
   appName= navigator.appName;
   appLongVer = navigator.appVersion;
   appVer = appLongVer.substring(0, 1);

   // Проверяем Internet Explorer версии 4+.
   if ((appName == "Microsoft Internet Explorer") && (appVer >= 4))
   return true;
   return false;
}
```

На страницу поместим графическое изображение с именем `imgMove`, и напишем функцию `moveDHTML()`, перемещающую это изображение:

```
function moveDHTML() {
// Задание объектов перемещения для разных браузеров
if (isNN()) targetObj = document.imgMove;
if (isIE()) targetObj = imgMove.style;

// Перемещаем изображение на 10 пикселов вправо.
targetObj.left = parseInt(targetObj.left) + 10;
}
```

Обратите внимание, как для каждого браузера задается объект перемещения.

Теперь остается привязать перемещение изображения к нажатию кнопки. Исходный текст HTML-страницы представлен ниже:

```
<SCRIPT LANGUAGE="JavaScript">
 Здесь размещаются тексты трех разработанных функций
</SCRIPT>
</HEAD>
<SPAN ID="imgMove" STYLE="position:relative; top:0; left:0;">
   <IMG SRC="Ж.gif" ALT="Перемести меня">
</SPAN>
<FORM >
  <P><INPUT TYPE="button" VALUE="Перемещение" onclick="moveDHTML()></P>
</FORM>
```

При просмотре страницы графическое изображение буквы Ж появляется в левой части окна браузера. Ниже расположена кнопка с надписью **Перемещение**. При нажатии на нее буква Ж смещается вправо. Это будет происходить при просмотре страницы как в браузере IE, так и в браузере NN. На рис. 10.31 показано смещенное изображение в окне браузера Netscape Navigator.

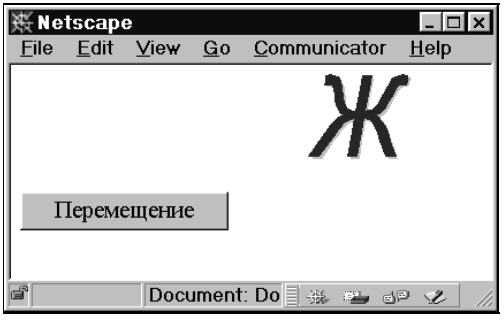


Рис. 10.31. Динамическое позиционирование графического элемента

Примечание

Приведенный пример достаточно прост. Более сложные приемы динамического позиционирования элементов в Netscape Navigator можно найти в документации "Dynamic HTML in Communicator" на сервере разработчика фирмы Netscape по адресу **http://developer.netscape.com/**.

Загружаемые шрифты

Для повышения привлекательности HTML-документов автор может использовать многочисленные шрифты, находящиеся в его распоряжении. Но что произойдет, если этих шрифтов не окажется на компьютере читателя его документов? Они будут заменены другими в соответствии с установленными на компьютере пользователя правилами подстановки шрифтов. В этом случае, возможно, вся привлекательность HTML-страницы исчезнет.

Во избежание подобных ситуаций в браузере NN реализована возможность загрузки необходимых для просмотра страницы шрифтов. Файл описания шрифтов автоматически загружается с хост-сервера, на котором находится просматриваемая страница, во временную кэш-область браузера пользователя и находится там, пока связанная с ним HTML-страница в ней хранится. Загрузить эти шрифты с хост-машины для постоянного использования нельзя.

Файл описания шрифтов создается в специальных программах (например HexMac Typograph) или с помощью Font Composer Plugin для Netscape Navigator. Работа в подобных программах достаточно проста: загружается документ, для которого необходимо создать файл описания шрифтов, поме-

чаются все шрифты, которые будут загружаться, и специальной командой создается файл описания шрифтов (расширение PFR), который сохраняется на сервере разработчика.

К документу файл описания шрифтов присоединяется либо посредством тэга `<LINK>`, либо через задание свойства `@fontdef` каскадных таблиц стилей в тэге `<STYLE>`. Ниже показаны оба способа встраивания ссылки на файл описания шрифтов:

```
<STYLE TYPE="text/css"><!--
   @fontdef url(http://home.netscape.com/fonts/sample.pfr);
--></STYLE>
<LINK REL=FONTDEF SRC="http://home.netscape.com/fonts/sample.pfr">
```

Использование загружаемых шрифтов ничем не отличается от использования других шрифтов. Их имена и параметры можно использовать в качестве значений параметра `FACE` тэга `<FONT>` или свойств каскадных таблиц стилей из категории шрифтов:

```
<STYLE type="text/css">><!--
   H1 {font-family:"Impress BT", "Helvetica", sans-serif;}
--></STYLE>
<H1> <FONT FACE="Impress BT">В H1 используется шрифт
                        Impress BT</FONT></H1>
```

При задании загружаемого шрифта в тэге `<FONT>` можно использовать дополнительные параметры: `FONT-SIZE` для определения размера шрифта и `WEIGHT` для определения степени жирности шрифта.

ГЛАВА 11

Встраиваемые компоненты

Современные тенденции в области разработки программных продуктов характеризуются широким использованием самостоятельных программных компонентов как строительных единиц создаваемых приложений.

Не так давно разработанный (1995), но получивший достаточно быстрое распространение язык Java позволяет создавать специальные программные компоненты — апплеты, которые предназначены для встраивания в HTML-страницы и передаче их на компьютер пользователя, где они и выполняются в режиме интерпретации специальным модулем браузера.

Элементы управления ActiveX (ранее называвшиеся *элементы управления OLE*) давно применяются в разработках программных продуктов. Они создаются на основе технологии COM и представляют собой самостоятельные программные единицы, выполняющиеся в определенной среде — среде программы-контейнера. Их возможность встраивания в HTML-страницы обеспечивается либо самим браузером (Internet Explorer), либо специальными дополнительными модулями к нему (Netscape Navigator).

Данная глава посвящена вопросам встраивания апплетов и элементов управления ActiveX в HTML-страницы и взаимодействия с ними из сценария JavaScript или VBScript.

Элементы управления ActiveX

Элементы управления ActiveX — это самостоятельные программные компоненты, которые можно использовать в разрабатываемых приложениях для реализации необходимых функциональных возможностей. Обратим внимание читателя на слова "программные компоненты" в приведенном определении. Дело в том, что элементы управления ActiveX создаются с использованием технологии Component Object Model (COM — Компонентная объектная модель), разработанной фирмой Microsoft, и не являются самостоятельным приложением — они выполняются только в приложении, которое позволяет

встраивать элементы управления ActiveX, являясь, тем самым, для них неким контейнером. Вне приложения-контейнера ни один элемент управления ActiveX невозможно запустить на выполнение и требовать от него каких-нибудь результатов.

Не всякое приложение может быть контейнером для элементов управления ActiveX. Оно должно поддерживать технологию COM и предоставлять возможность манипулирования встраиваемыми компонентами. Многие приложения фирмы Microsoft позволяют широко использовать элементы управления ActiveX либо для расширения функциональных возможностей самого приложения, либо для быстрого создания новых приложений, использующих элементы управления ActiveX в качестве строительных блоков. К подобным приложениям можно отнести Microsoft Visual Basic, Microsoft Access, Microsoft Internet Explorer и некоторые другие.

Технология COM — это дальнейшее развитие идеи объектно-ориентированного программирования. Она позволяет использовать объекты со своими свойствами, методами и событиями, которые создаются не во время выполнения программы, написанной на каком-либо объектно-ориентированном языке программирования, а существуют в виде отдельных программных единиц, называемых компонентами. При включении подобного объекта-компонента в тело разрабатываемой программы он раскрывает свои свойства и методы, которые можно использовать в присущей объектно-ориентированным технологиям манере: получать или устанавливать значения его свойств, а также выполнять доступные методы.

Для реализации какой-нибудь функциональной возможности с помощью элемента управления ActiveX, например отображения дерева каталогов, программист может разработать свой элемент управления ActiveX, а может воспользоваться одним из многочисленных элементов управления ActiveX, разработанных и распространяемых другими фирмами. Следует знать и помнить, что не все они распространяются бесплатно. Поэтому, если вы решили использовать на своей HTML-странице какой-либо элемент управления ActiveX, следует позаботиться о приобретении лицензии на его использование у фирмы-изготовителя.

Преимущества использования компонентной объектной модели и элементов управления ActiveX для разработки новых приложений очевидны: не надо тратить времени и усилий для программирования функциональных возможностей приложения. А какие преимущества дает использование элементов управления ActiveX в Web-документах? Прежде всего, они расширяют возможности Web-документов, приближая их к полноценным приложениям. Помните, как легко была реализована возможность работы с содержимым базы данных прямо с HTML-страницы простым включением в ее состав элемента управления TDC *(см. главу 2 раздел "Связывание данных с документом")*, который, естественно, является элементом управления ActiveX! Далее, они могут добавить привлекательности к разрабатываемым страницам,

если реализуют мультимедийные эффекты, позволить отобразить данные способом, не возможным в рамках языка HTML или даже динамического HTML, например, вывести текст, расположенный под углом, и многое другое. И все это практически без программирования! Простое использование тэга `<OBJECT>` совместно с тэгом `<PARAM>`, в котором задаются значения свойств элемента управления ActiveX, — и вы читаете данные из базы данных.

Встраивание в HTML-страницу

Для встраивания в HTML-страницу внешних объектов предназначен тэг `<OBJECT>`, который является контейнером для тэгов `<PARAM>`, определяющих значения свойств включаемого объекта. Тэг `<OBJECT>` имеет несколько параметров, два из которых непосредственно связаны с элементами управления ActiveX.

Параметр `CLSID` задает уникальный идентификационный номер встраиваемого на страницу элемента управления ActiveX, а значение параметра `CODEBASE` определяет URL-адрес расположения исходных файлов этого элемента.

Но прежде, чем начинать описание тэга `<OBJECT>` и его параметров, остановимся немного на вопросах установки элементов управления ActiveX на компьютере пользователя, так как они тесно переплетаются с использованием элементов ActiveX на HTML-страницах и в Web-приложениях, реализованных с помощью встраиваемых сценариев.

Любой устанавливаемый на компьютере элемент управления ActiveX регистрируется в системном реестре, куда заносится и там же хранится соответствующая информация. В дальнейшем эта информация используется браузером Internet Explorer для обработки HTML-страниц, содержащих встроенные элементы управления ActiveX.

В разделе HKEY_CLASSES_ROOT/CLSID/ реестра хранятся *уникальные идентификационные номера* всех установленных на компьютере элементов управления ActiveX. Запись в этом разделе для элемента управления RDS, знакомого читателю по главе 10, можно увидеть на рис. 11.1.

Для рассматриваемого элемента управления ActiveX создан раздел, названием для которого и служит уникальный идентификационный номер элемента управления `{BD96C556-65A3-11D0-983A-00C04FC29E33}`, присваиваемый ему при создании. Для целей применения элементов управления ActiveX на HTML-страницах достаточно знать, что именно COM ответственна за присваивание уникального идентификационного номера. Интересующийся читатель может обратиться к документации по разработке элементов управления ActiveX для более подробного ознакомления с механизмом назначения идентификационных номеров.

Каждый раздел — это параметр реестра. Если выделить интересующий раздел в левой панели окна программы работы с реестром Regedit.exe, то на

правой панели отобразится значение этого параметра. На рис. 11.1 значением выделенного раздела является RDS.DataControl.

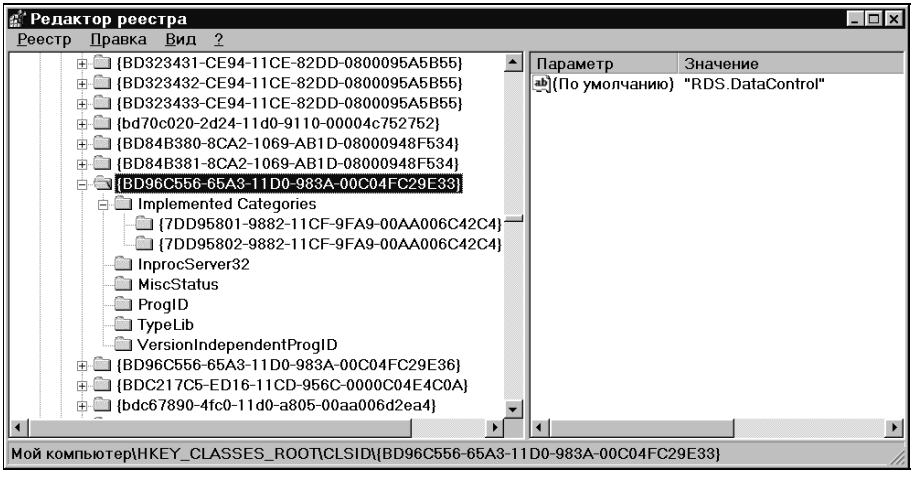


Рис. 11.1. Запись в системном реестре Windows для элемента управления RDS

Кроме записи в разделе HKEY_CLASSES_ROOT/CLSID/ для элемента управления создается собственный раздел в каталоге HKEY_CLASSES_ROOT, в котором параметр `Clsid` также имеет значение, равное идентификационному номеру элемента управления. На рис. 11.2 показан раздел элемента управления RDS.DataControl.

Значением параметра `Clsid` является уникальный идентификационный номер элемента управления RDS.DataControl.

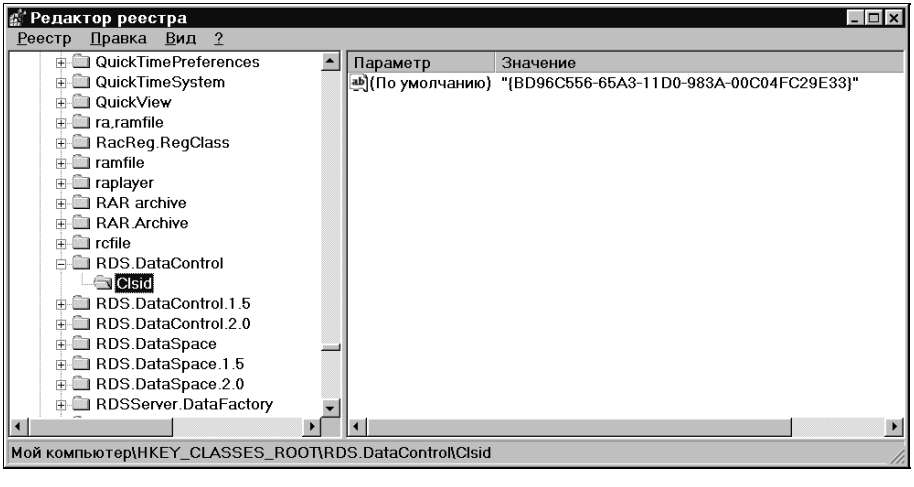


Рис. 11.2. Запись в системном реестре Windows для элемента управления RDS

Как отмечалось выше, элементы управления ActiveX встраиваются в страницу HTML тэгом-контейнером `<OBJECT>`. Он имеет большое число параметров, и его полный синтаксис представлен ниже:

```
<OBJECT
    ACCESSKEY=клавиша
    ALIGN=ABSBOTTOM | ABSMI
DDLE | BASELINE | BOTTOM | LEFT |
        MIDDLE | RIGHT | TEXTTOP | TOP
    CLASS=имя_класса
    CLASSID=идентификатор_объекта
    CODE=имя_файла
    CODEBASE=url-адрес[#version=a,b,c,d]
    CODETYPE=media-тип
    DATA=url-адрес
    DATAFLD=имя_столбца
    DATASRC=#идентификатор_источника
    HEIGHT=целое_число
    ID=идентификатор
    LANG=язык
    LANGUAGE=JAVASCRIPT | JSCRIPT | VBSCRIPT  | VBS
    NAME=имя
    STYLE=правила_CSS
    TABINDEX=целое_число
    TITLE=текст
    TYPE=MIME-тип
    WIDTH=целое_число
>
```

Элемент HTML `<OBJECT>` является блоковым, поэтому для его правильной интерпретации необходимо предусмотреть на странице закрывающий тэг `</OBJECT>`. Задание всех параметров, естественно, не обязательно, но некоторые необходимы для правильной работы внедряемого объекта и включения его самого на HTML-страницу.

Для элементов управления ActiveX *обязательно* задание параметра `CLASSID`. Значением его является уникальный идентификационный номер встраиваемого элемента управления ActiveX. При загрузке страницы браузер проверяет, установлен ли элемент управления на компьютере пользователя, осуществляя его поиск в системном реестре по заданному идентификационному номеру. В случае отсутствия записи в реестре браузер автоматически начинает процедуру загрузки элемента управления ActiveX с сервера, URL-адрес которого указан в параметре `CODEBASE` тэга `<OBJECT>`.

Значение параметра `CLASSID` представляет строку, определяющую встраиваемый объект, и для зарегистрированных элементов управления ActiveX задается в форме:

```
"clsid:XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
```

Первая часть `clsid:` сообщает анализатору браузера, что вставляется элемент управления ActiveX. Вторая часть представляет собой уникальный идентификационный номер этого элемента. Например, кнопка управления из набора элементов управления, используемых при создании форм в приложениях Office 97 фирмы Microsoft, встраивается на HTML-страницу при помощи следующего тэга:

```
<OBJECT ID=cmd1 CLASSID="CLSID:D7053240-CE69-11CD-A777-00DD01143C57">
</OBJECT>
```

В параметре `CODEBASE` задается адрес компонента, по которому браузер может загрузить его на компьютере пользователя в случае отсутствия:

```
<OBJECT ID="myActX" WIDTH=32 HEIGHT=32
    CLASSID="CLSID:12D3959D-5048-11D3-A272-8C0305C10000"
    CODEBASE="http://bhv.spb.ru/ActiveX/advert32.cab
    #Version=1,0,0,0">
</OBJECT>
```

В этом примере определен полный адрес элемента управления, причем обратим внимание читателя на расширение CAB загружаемого файла. Очень часто для ускорения загрузки разнообразных компонентов они пересылаются по сети в упакованном виде. В данном случае элемент управления упакован программой Cabarc.exe, используемой фирмой Microsoft для упаковки и распространения своих продуктов.

После адреса элемента управления задается его версия в виде: `#Version=a,b,c,d`, где `a` и `b` представляют, соответственно, старшее и младшее слово максимально доступной на сервере версии элемента управления, а `c` и `d` — соответственно старшее и младшее слово минимально доступной на сервере версии элемента управления. Эти значения используются браузером в процессе принятия решения о загрузке элемента управления с сервера. Если на компьютере пользователя установлена более новая версия элемента управления, то загрузка не производится. В приведенном примере, если на компьютере пользователя будет установлена версия элемента управления выше, чем 1.0, то загрузки не будет.

Значения, определяющие интервал доступных версий компонента могут быть все установлены равными "`-1`". В этом случае элемент управления загружается, если дата выпуска его версии на сервере позже даты его установки на компьютере пользователя.

Параметры `WIDTH` и `HEIGHT` задают в пикселах размеры визуального интерфейса элемента управления ActiveX, если он у него существует. Например, кнопка управления или метка, реализованные в виде элементов управления ActiveX, имеют визуальные интерфейсы, а элементы управления RDS или TDC, позволяющие работать с данными, не имеют.

Значение параметра ACCESSKEY определяет клавишу быстрого доступа к элементу управления. Это означает, что при одновременном нажатии комбинации клавиш <Alt> и заданной, элемент управления получает фокус.

Примечание

При получении фокуса некоторыми элементами управления выполняются предписанные им по умолчанию действия. Например, если фокус получает кнопка управления, то генерируется событие Click этой кнопки и выполняются предусмотренные в процедуре обработки этого события действия. Если фокус получает элемент управления **OptionButton** (Переключатель), то также генерируется событие Click, и свойство checked (выбран) переключателя принимает противоположное значение, отображая новое состояние переключателя: если он был выбран, то устанавливается в положение "не выбран", и наоборот.

Каждый элемент управления ActiveX, являясь компонентом, раскрывает свои свойства и методы программе-контейнеру. При включении элемента управления на HTML-страницу можно задать значения его свойств в тэгах <PARAM>, содержащихся в теле тэга <OBJECT>. Тэг <PARAM> не является тэгом-контейнером и ему не требуется закрывающий тэг </PARAM>. Название свойства и его значение определяются параметрами NAME и VALUE тэга <PARAM>:

```
<OBJECT ID="myActX" WIDTH=32 HEIGHT=32
    CLASSID="CLSID:12D3959D-5048-11D3-A272-8C0305C10000"
    CODEBASE="http://bhv.spb.ru/ActiveX/advert32.cab
    #Version=1,0,0,0">
    <PARAM NAME="Value"     VALUE="Текст">
    <PARAM NAME="TextColor" VALUE="red">
</OBJECT>
```

В этом примере задаются значения двух свойств элемента управления. Если значения свойств не заданы при внедрении элемента управления, то он инициализируется со значениями свойств по умолчанию.

Примечание

Все свойства и методы элемента управления ActiveX можно найти в документации, поставляемой продавцом элемента управления.

Элементы управления ActiveX и сценарии

Каждый встроенный на HTML-страницу элемент управления ActiveX в объектной модели документа браузера Internet Explorer представляется в виде объекта Object. Свойства, методы и события этого объекта соответствуют свойствам, методам и событиям соответствующего элемента управления ActiveX.

В объектной модели документа реализованы свойства, соответствующие параметрам тэга `<OBJECT>`. Например, свойство `className` соответствует параметру `CLASS` тэга. Может оказаться, что реализация внедренного объекта поддерживает какое-либо свойство или метод с таким же названием, определенным в реализации объектной модели. Во избежание конфликтов в таких случаях следует использовать свойство `object` объекта для доступа к соответствующему свойству или методу, реализованному внедренным объектом. Например, если в объектной модели и в самом элементе управления ActiveX реализован метод `item()`, то следующие операторы обеспечивают, соответственно, доступ к методу объектной модели и самого объекта:

```
document.all.nameOfObject.item() // метод объектной модели
document.all.nameOfObject.object.item() // метод элемента управления
```

В этих операторах `nameOfObject` — имя объекта, определенное в параметре `ID` тэга `<OBJECT>`.

События, поддерживаемые элементом управления ActiveX, посылаются непосредственно объекту `Object`, представляющему в объектной модели элемент управления. В сценарии можно определить обработчик событий — процедуру, выполняющуюся при генерировании определенного события элемента управления.

Определение *обработчика событий* для элементов управления ActiveX в Internet Explorer 4.0 несколько не укладывается в общепринятые нормы назначения процедуры обработки события какого-либо HTML-элемента. Задание обработчика события `onСобытие` в качестве параметра тэга `<OBJECT>` не приводит к желаемым результатам. Чтобы определить процедуру обработки события элемента управления ActiveX, следует воспользоваться тэгом `<SCRIPT>` с параметрами `FOR` и `EVENT` *(см. главу 9 "Выполняемые сценарии")*:

```
<SCRIPT FOR=имяЭлемента EVENT=Событие LANGUAGE=[JavaScript|VBScript]>
 исходный текст процедуры обработки события
</SCRIPT>
```

С помощью такого синтаксиса тэга `<SCRIPT>` можно определять обработчики событий как в языке JavaScript, так и в языке VBScript, не забывая указывать последний явно в параметре `LANGUAGE`. По умолчанию любой браузер использует JavaScript в качестве языка сценария.

В VBScript обработчик события для элемента управления ActiveX можно определить, также используя специальные правила именования процедур, вызываемых в ответ на возникновение определенного события для определенного объекта. Имя такой процедуры составляется из имени объекта и названия события, соединенных знаком подчеркивания "_". Например, если элемент управления определен с именем `TreeView1` и необходимо создать процедуру обработки события `Click`, то достаточно определить процедуру

с именем `TreeView1_Click`. Внимательный читатель тотчас же заметит, что это несколько отличается от подобного же правила для процедур обработки событий обычных HTML-элементов, для которых во второй части имени процедуры задается не имя самого события, а обработчик события с префиксом `on` перед именем события. Если бы объект `TreeView1` был обычным объектом, соответствующим HTML-элементу, то имя процедуры обработки его события `Click` было бы `TreeView1_onClick`. Такое задание имени процедуры обработки событий элементов управления ACttiveX является исключением и его следует помнить, так как если для элемента управления ActiveX задать процедуру по правилам VBScript, то интерпретатор не сгенерирует никакой ошибки, но и никакого действия не будет выполняться при возникновении соответствующего события.

В примере 11.1 показано, как правильно обрабатывать щелчок кнопкой мыши на элементе управления **Label** (Метка), поставляемого вместе с Internet Explorer 4.0.

Пример 11.1. Обработка событий элементов управления ActiveX

```
<HTML>
<SCRIPT LANGUAGE="VBScript">
<!--
Sub lblActive1_Click
  lblActive1.Angle = (lblActive1.Angle + 45) Mod 360
End Sub
-->
</SCRIPT>
</HEAD>
<BODY>
<H2 STYLE="color: blue; background-color: lightgrey">
Пример 11.1 Обработка щелчка на элементе управления ActiveX
</H2>
<P>Щелчок кнопкой мыши на слове "Текст" поворачивает его
   на 45 градусов!<P>
<DIV>
<OBJECT CLASSID="clsid:99B42120-6EC7-11CF-A6C7-00AA00A47DD2"
       ID=lblActive1
       WIDTH=150
       HEIGHT=150 >
  <PARAM NAME="Angle" VALUE="0">
  <PARAM NAME="Alignment" VALUE="4">
  <PARAM NAME="BackStyle" VALUE="0">
  <PARAM NAME="Caption" VALUE="ТЕКСТ">
  <PARAM NAME="FontName" VALUE="Verdana, Arial, Helvetica">
  <PARAM NAME="FontSize" VALUE="20">
```

```
  <PARAM NAME="FontBold" VALUE="1">
  <PARAM NAME="FrColor" VALUE="0">
</OBJECT>
</DIV>
</BODY>
```

Используемый в этом примере элемент управления имеет свойство `Angle`, определяющее наклон текста, задаваемого в свойстве `Caption`, относительно горизонтали. Первоначальное значение этого свойства равно 0°. Когда пользователь щелкает кнопкой мыши на содержимом этого элемента управления, генерируется событие `Click`, которое перехватывается и обрабатывается процедурой `lblActive1_Click`, имя которой определено в соответствии с изложенными выше правилами. Эта процедура добавляет 45° к значению свойства `Angle` элемента управления и приводит его к диапазону от 0° до 360°, если новое значение угла поворота выходит за указанный диапазон. Изменение значения свойства приводит к тому, что надпись "ТЕКСТ" поворачивается на 45° относительно своего предыдущего положения.

Следует заметить, что событие, генерируемое для элемента управления ActiveX, как и любое событие в объектной модели документа, "всплывает" вверх по иерархии объектов документа и обрабатывается соответствующими обработчиками событий объектов-контейнеров верхних уровней.

Обратим внимание на то, что в примере 11.1 элемент управления **Label** содержится в тэге раздела `<DIV>`. Если задать обработчик события `onclick` для этого раздела, например, в виде вызова окна предупреждения

```
<DIV onclick="alert('Я всплыл из тэга DIV!')>
```

то при щелчке кнопкой мыши на метке кроме поворота текста будет отображаться диалоговое окно с надписью "Я всплыл из тэга DIV".

Вставка на HTML-страницу элементов управления ActiveX связано со знанием его уникального идентификационного номера, а также всех доступных свойств, методов и событий элемента управления. Существуют средства разработки Web-документов, автоматизирующие процесс встраивания элементов управления ActiveX на HTML-страницы.

Одним из первых подобных средств была программа Microsoft ActiveX Control Pad, которая послужила основой для редактора FrontPage, входящего в целый комплекс программного обеспечения FrontPage 98 фирмы Microsoft, предназначенный для разработки и поддержки простого "домашнего" Web-сервера.

Примечание

Программа Microsoft ActiveX Control Pad свободно распространяется в Интернете и доступна на сервере разработчика фирмы Microsoft по адресу **http://msdn.microsoft.com/workshop/**.

Программа FrontPage 98 получила дальнейшее развитие и в настоящее время ее новая модификация под названием FrontPage 2000 интегрирована в Microsoft Office 2000.

В следующем разделе будет описана технология включения элементов управления ActiveX на HTML-страницу в редакторе FrontPage 98, так как FrontPage 2000 еще не получил большого распространения, а основные приемы работы остаются такими же, как и в редакторе FrontPage 98.

Редактор FrontPage 98

Редактор FrontPage предназначен для создания, разработки и редактирования Web-документов. При добавлении текста, изображений, таблиц и других элементов на страницу редактор FrontPage отображает их в том виде, какой они будут иметь в окне браузера. Знание языка HTML не обязательно, так как FrontPage в конечном итоге сам создает HTML-страницу, содержащую все включенные разработчиком элементы. Этот редактор может генерировать все известные тэги HTML, включая свойства каскадных таблиц стилей, фреймы и *элементы управления ActiveX*.

Его рабочая область состоит из трех вкладок, переход между которыми осуществляется с помощью расположенных в нижней части рабочей области ярлычков (рис. 11.3). На каждой вкладке разрабатываемый или редактируемый HTML-документ имеет разное представление.

Примечание

Описание работы с элементами управления ActiveX в редакторе FrontPage будет производиться на примере английской версии редактора, поэтому указанные в скобках русские названия меню, команд, полей, списков и т. д. могут не соответствовать русской версии редактора.

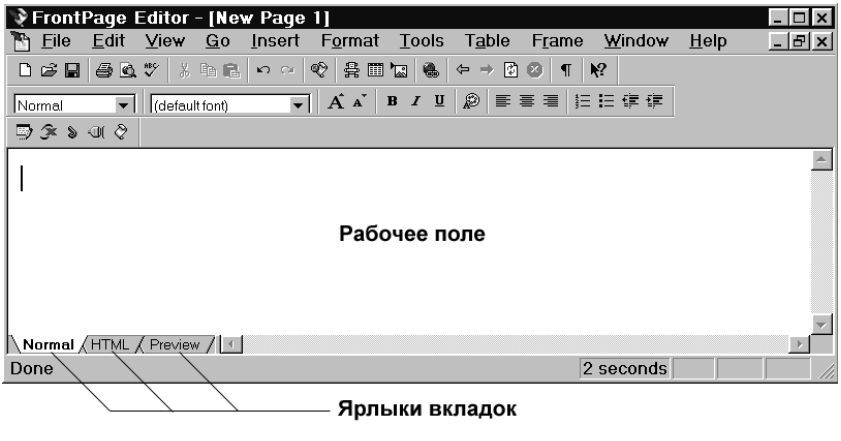


Рис. 11.3. Общий вид интерфейса редактора FrontPage 98

Вкладка **Normal** (Нормальная) является рабочей. Именно здесь осуществляется создание и разработка HTML-страницы. Большинство вставляемых элементов имеют такой же внешний вид, что и в окне браузера, но их расположение не отражает окончательную компоновку страницы.

Переход на вкладку **HTML** позволяет увидеть исходный текст HTML разрабатываемой или редактируемой страницы, причем пользователь может его редактировать, что достаточно удобно при внесении небольших корректировок в исходный текст.

На вкладке **Preview** (Просмотр) можно посмотреть Web-документ так, как он будет отображаться в окне браузера. Здесь нельзя вносить никакие изменения в разрабатываемую страницу.

На рис. 11.4 показан вид HTML-страницы примера 11.1 на всех трех вкладках редактора FrontPage 98.

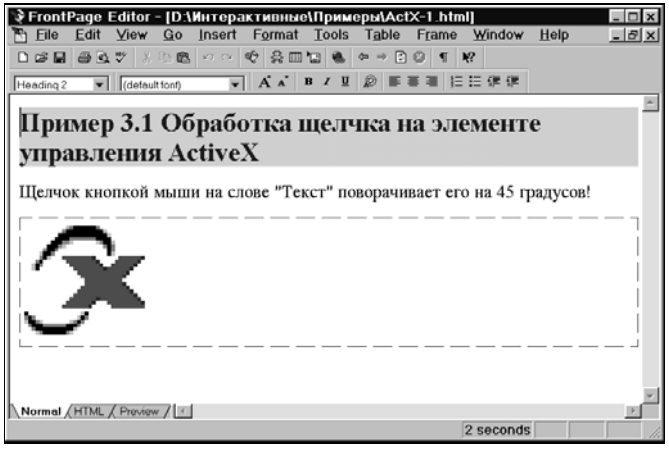
На рис. 11.4, *а* видно, что встроенный на страницу элемент управления ActiveX (а читатель помнит, что в примере 11.1 использовался элемент управления **Label**) на основной, рабочей вкладке представляется специальным графическим изображением — пиктограммой ActiveX. Причем ширина и высота этой пиктограммы соответствуют ширине и высоте элемента управления, определенными, соответственно, в параметрах `WIDTH` и `HEIGHT` тэга `<OBJECT>`, задающего элемент управления ActiveX.

Двойной щелчок на пиктограмме отображает диалоговое окно, в котором можно произвести корректировку значений свойств элемента управления, добавить или изменить установки свойств каскадных таблиц стилей, а также откорректировать старые или добавить новые процедуры обработки событий элемента управления.

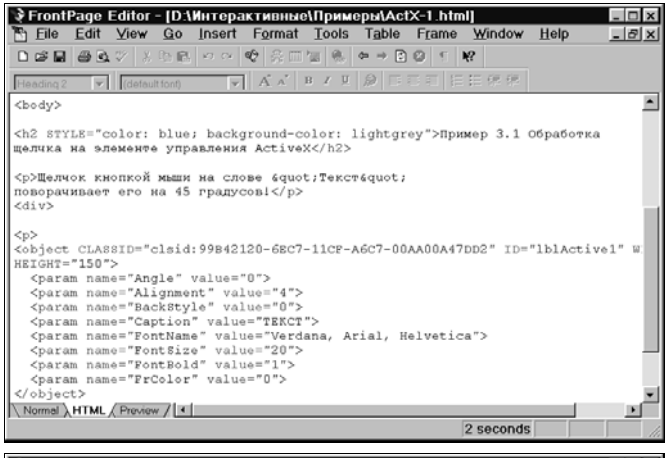
В редакторе FrontPage любой элемент добавляется на страницу посредством команд меню **Insert** (Вставить). Вставка элемента управления ActiveX осуществляется вызовом команды **ActiveX Control** (Элемент управления ActiveX) подменю **Advanced** (Дополнительно) этого меню. Дополнительная информация относительно встраиваемого элемента управления ActiveX задается в отображаемом этой командой диалоговом окне **ActiveX Control Properties** (Свойства элемента управления ActiveX), показанном на рис. 11.5.

В раскрывающемся списке **Pick a Control** (Выберите элемент управления) следует выбрать элемент управления ActiveX. Этот список содержит все, установленные на компьютере элементы управления ActiveX.

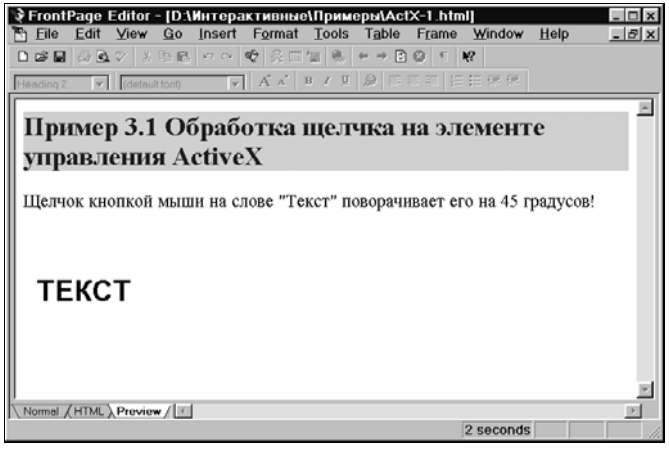
В полях этого диалогового окна вводятся значения параметров тэга `<OBJECT>`, который встраивает выбранный элемент управления ActiveX на HTML-страницу. Каждое поле соответствует определенному параметру тэга, название которого ясно из названия самого поля. Например, поле **Code Source** соответствует параметру `CODEBASE`, поле **Data Source** — параметру `DATA` и т. д.



а



б



в

Рис. 11.4. Одна и та же HTML-страница на разных вкладках рабочего окна редактора FrontPage 98

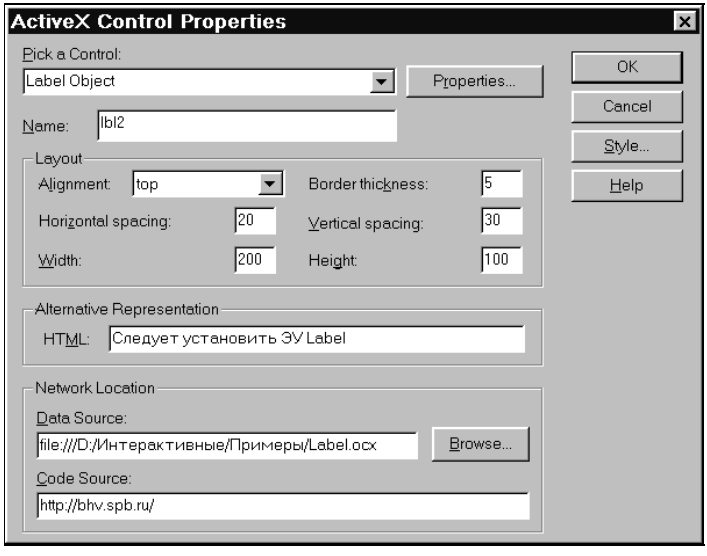


Рис. 11.5. Диалоговое окно **ActiveX Control Properties**

Для заполненного диалогового окна на рис. 11.5 редактор FrontPage вставит в исходный текст HTML-страницы следующий тэг `<OBJECT>` с заданными значениями параметров:

```
<object align="top" codebase="http://bhv.spb.ru/"
        id="lbl2" border="5" hspace="20"
        vspace="30" width="200" height="100"
        data="file:///D:/Интерактивные/Примеры/Label.ocx"
        classid="clsid:99B42120-6EC7-11CF-A6C7-00AA00A47DD2">
Следует установить ЭУ Label</object>
```

Совет

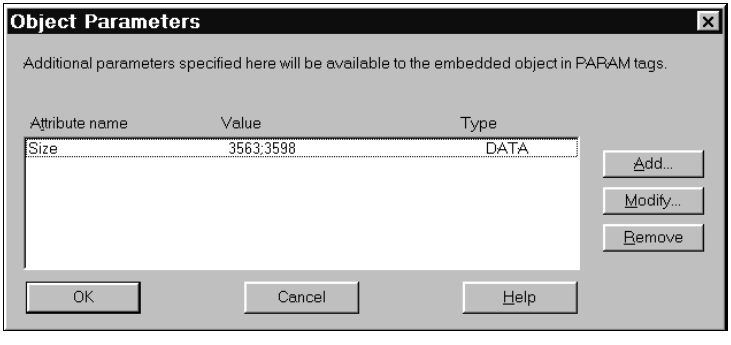
Сопоставление параметров и их значений в приведенном примере со значениями полей диалогового окна на рис. 11.5 поможет найти соответствие между полями формы и параметрами тэга `<OBJECT>`, если оно не достаточно ясно из названий полей.

Обратим внимание только на параметр `CLASSID`, в котором задается уникальный идентификационный номер элемента управления. Редактор автоматически определяет его в реестре системы и вставляет в качестве значения параметра `CLASSID`, избавляя разработчика от самостоятельного поиска идентификационного номера в реестре.

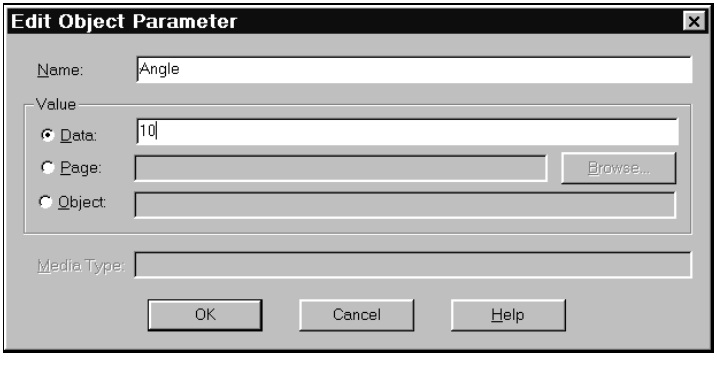
Однако пока остаются не определенными значения свойств элемента управления, задаваемые в тэгах `<PARAM>`, содержащихся в теле тэга `<OBJECT>`. Установка значений доступных свойств элемента управления осуществляется

в диалоговом окне редактора свойств, если элемент управления установлен и поддерживает локальное редактирование свойств, или в диалоговом окне **Object Parameters** (Параметры объекта) в противном случае. Окно **Object Parameters** отображается нажатием кнопки **Properties** (Свойства) диалогового окна **ActiveX Control Properties**.

Элемент **Label Control**, используемый в нашем примере, не поддерживает локальное редактирование свойств, поэтому установку его свойств приходится осуществлять в диалоговом окне **Object Parameters**, которое требует знания всех свойств элемента управления и их допустимых значений (рис. 11.6, *а*). В нередактируемом списке этого диалогового окна отображаются названия и значения установленных свойств. В поле **Attribute name** (Имя параметра) отображается имя свойства, которое задается в параметре `NAME` тэга `<PARAM>`, а поле **Value** (Значение) содержит его значение, задаваемое параметром `VALUE`. Добавление или изменение значения свойства осуществляется нажатием кнопок **Add** (Добавить) или **Modify** (Изменить). При этом отображается диалоговое окно **Edit Object Properties** (Редактирование свойств объекта), в полях которого задается название свойства и его значение (рис. 11.6, *б*).



а



б

Рис. 11.6. Установка значений свойств элемента управления ActiveX в окне **Object Parameters**

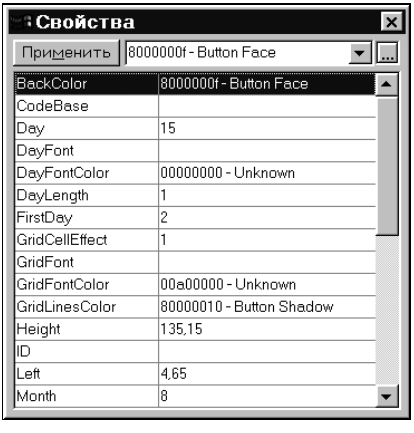
Совет

Для элемента управления **Label Control** программа MS ActiveX Control Pad отобразит все устанавливаемые свойства элемента в редакторе свойств. Если FrontPage по какой-либо причине не отображает для элемента управления редактор свойств, а вы забыли названия некоторых свойств и нет под рукой документации, можно попробовать использовать упомянутую программу, которая свободно распространяется в Интернете.

Редактор свойств для элемента управления ActiveX, поддерживающего локальное редактирование свойств, представляет два окна: в одном отображается визуальный интерфейс элемента управления или его пиктограмма, если у элемента нет визуального интерфейса, другое окно представляет окно редактирования свойств, хорошо знакомое всем, работающим с визуальными системами программирования. Два окна редактора свойств для элемента управления **Calendar Control 8.0** показаны на рис. 11.7 (*а* — визуальный интерфейс, *б* — окно свойств).

Выбирая в окне свойств необходимый параметр, можно изменять его значение. Это изменение немедленно отображается в окне визуального интерфейса, если свойство влияет на его параметры. Некоторые свойства элемента можно менять в окне визуального интерфейса, например ширину и высоту элемента. Их изменение также будет отображаться в окне свойств.

а



б

Рис. 11.7. Окна редактора свойств элемента управления ActiveX

По завершении редактирования свойств элемента управления ActiveX и закрытия диалогового окна **ActiveX Control Properties**, в исходном тексте HTML-страницы в тэг `<OBJECT>` вставляются все необходимые тэги `<PARAM>`, определяющие значения свойств элемента управления. Для нашего примера после добавления значения свойства `Angle` тэг `<OBJECT>` будет выглядеть следующим образом:

```
<object align="top" codebase="http://bhv.spb.ru/"
        id="lbl2" border="5" hspace="20"
```

```
        vspace="30" width="200" height="100"
        data="file:///D:/Интерактивные/Примеры/Label.ocx"
        classid="clsid:99B42120-6EC7-11CF-A6C7-00AA00A47DD2">
  <param name="Size" value="3563;3598">
  <param name="Angle" value="10">
Следует установить ЭУ Label</object>
```

Добавление обработчиков событий элементов управления ActiveX на страницу HTML в программе FrontPage 98 так же просто, как и встраивание и изменение свойств элементов управления ActiveX. Для этого предназначена команда **Script Wizard** (Мастер сценария) контекстного меню элемента управления ActiveX, которое отображается щелчком правой кнопки мыши на элементе управления при работе на вкладке **Normal**. Эта команда отображает диалоговое окно Мастера сценария (рис. 11.8.), в котором задаются все необходимые параметры и действия для процедуры обработки события.

В заголовке диалогового окна в скобках после имени файла отображается используемый язык сценария: VBScript или JavaScript. Установка языка сценария осуществляется командой **Insert**, **Advanced**, **Script** (Вставка, Дополнительно, Сценарий), в диалоговом окне которой в группе переключателей **Language** (Язык) следует выбрать используемый язык сценария.

Примечание

При разработке HTML-страницы в программе FrontPage можно использовать только один язык сценария для процедур обработки событий: VBScript или JavaScript. Если редактируется страница со смешанными сценариями, то программа выдаст предупреждение о том, что смешанные сценарии не поддерживаются полностью и могут возникнуть проблемы при отображении страницы.

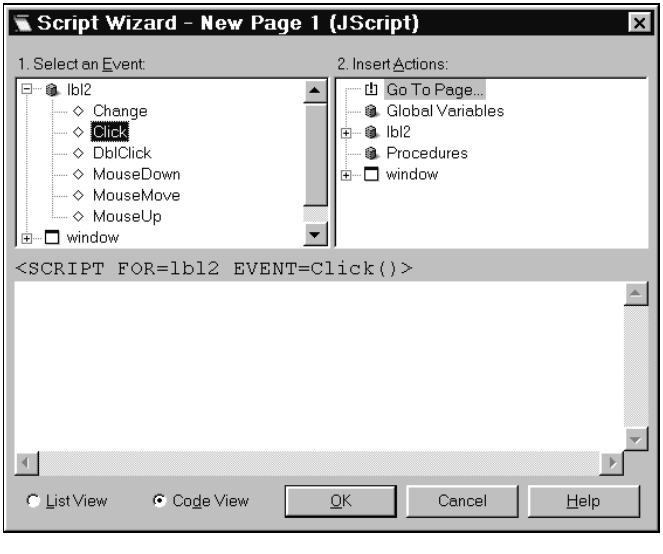


Рис. 11.8. Диалоговое окно Мастера сценария

Диалоговое окно Мастера сценария состоит из трех панелей. В панели **Select an Event** (Выбрать событие) отображаются в алфавитном порядке все элементы редактируемой HTML-страницы, которые могут генерировать события. Двойной щелчок на любом из элементов раскрывает список доступных для программирования событий этого элемента. При выборе какого-либо события элемента, для которого необходимо написать процедуру его обработки, над нижней панелью появляется заголовок, представляющий собой тэг `<SCRIPT>` с соответствующими параметрами `FOR` и `EVENT`. Теперь в рабочем поле этой панели можно вводить операторы процедуры.

Сделаем небольшое замечание, что подобное поведение редактора сценария происходит только тогда, когда в нижней части его диалогового окна выбран переключатель **Code View** (Просмотр кода). Выбор этого режима работы редактора сценария позволяет пользователю в рабочей области нижней панели набирать исходный текст сценария.

При выборе переключателя **List View** (Просмотр списка) включается режим создания сценария простым указанием необходимых действий из соответствующего списка. Более подробно об этом режиме можно прочитать в любой книге по FrontPage 98.

В панели **Insert Actions** (Ввод действий) можно найти значки, соответствующие доступным в сценарии объектам, например, на рис. 11.8 встроенный нами на страницу HTML под именем `lbl2` элемент управления **Label**. Двойной щелчок на элементе раскрывает список его доступных свойств, который можно использовать при ссылке на свойства элемента управления в процессе написания сценария: двойной щелчок на свойстве в этом списке помещает ссылку на свойство элемента управления в рабочее поле панели написания сценария, как показано на рис. 11.9.

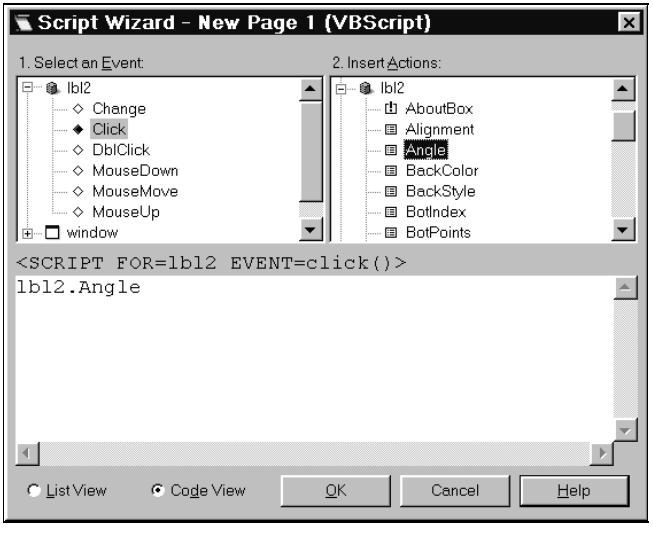


Рис. 11.9. Создание процедуры обработки события в окне Мастера сценария

Теперь можно просто продолжить оператор, присвоив этому свойству, например, значение 10. Там, где в сценарии необходимо сослаться на значение какого-либо свойства элемента, следует применять подобную технику во избежание ошибок в синтаксисе свойств.

Примечание

В списке событий элемента управления те события, для которых созданы процедуры обработки событий, помечены закрашенным ромбиком.

Редактор сценария позволяет создавать отдельные процедуры, которые можно использовать в других процедурах обработки событий, определять глобальные переменные, доступные всем процедурам страницы. Их можно создать, установив курсор на элементе **Global Variables** (Глобальные переменные) или **Procedures** (Процедуры) панели **Insert Actions**, и, щелкнув правой кнопкой мыши, в контекстном меню выбрать команду **New Global Variable** (Новая глобальная переменная) для создания новой переменной или **New Procedure** (Новая процедура) для создания отдельной процедуры.

Мы кратко остановились на возможностях FrontPage 98, имея в виду его использование для внедрения элементов управления ActiveX на HTML-страницы, а также написания процедур обработки их событий, хотя возможности этого редактора достаточно велики и позволяют быстро и эффективно создавать великолепные Web-документы.

Безопасность и элементы управления ActiveX

Вопрос о *безопасности компьютера* пользователя, подключенного к любой сети, является одним из важнейших в ряду других вопросов, связанных с получением внешних данных. Здесь возникает достаточно много проблем, знание о существовании которых уже может оградить пользователя от возможных неприятностей. Несанкционированный доступ к информации, находящейся на компьютере, запуск потенциально опасных программ, проникновение "вирусов" и многое другое — все это относится к безопасности компьютера.

Пользователь "путешествует" по сети Интернет и получает информацию с помощью специальных программ — браузеров (от английского *browse* — читать, просматривать). Каждый браузер поддерживает определенную стратегию безопасности, предупреждая пользователя о потенциально опасных загружаемых компонентах.

Internet Explorer 4.0 позволяет пользователю установить три уровня безопасности: высокий, средний и низкий. Каждый уровень характеризуется своими ограничениями на загрузку из сети программ, файлов, изображений и

другого ее содержимого. Это может быть полный отказ от загрузки потенциально опасных программ (как в случае высокого уровня безопасности) или разрешение, даже без дополнительного информирования пользователя, загружать на компьютер все, что встретится на HTML-странице (как в случае низкого уровня безопасности).

Широко используемые на HTML-страницах объекты — Java-апплеты и элементы управления ActiveX — также разрабатываются с учетом их безопасного применения. Стратегии безопасности для этих двух типов объектов различны. Если средства разработки Java-апплетов вообще не позволяют им обращаться к информации, расположенной на компьютере, то с элементами управления ActiveX дело обстоит немного сложнее. Последние используются не только в Web-приложениях, но и как строительные единицы при разработке обычных приложений. Поэтому, естественно, ни одна среда разработки элементов управления ActiveX не накладывает ограничений на доступ к информации, хранящейся на компьютере, где используется этот элемент. Каким же образом можно ограничить доступ к "закрытой" информации элементу управления ActiveX? Это достигается совместным использованием системы безопасности браузера и мероприятиями, связанными с регистрацией и установкой элементов управления ActiveX на компьютере пользователя при загрузке HTML-страницы.

Цифровая подпись

Если используемый на странице элемент управления ActiveX не установлен на компьютере пользователя, то браузер начнет процедуру загрузки и установки элемента управления на этом компьютере. Но, прежде чем загружать необходимые файлы, браузер проверит, снабжен ли элемент управления *цифровой подписью*, и если да, то в диалоговом окне отобразит содержимое цифровой подписи, если нет — предупредит пользователя о том, что будет загружаться элемент управления без цифровой подписи. Пользователь сам решает, стоит ли ему загружать такой элемент управления, который создан не известно кем и может быть потенциально опасен. На рис. 11.10 показано диалоговое окно безопасности браузера Internet Explorer 4.0 при загрузке элемента управления ActiveX, снабженного цифровой подписью.

В диалоговом окне на рис. 11.10 сообщается, что будет установлен элемент управления **Microsoft Windows Media Player**, подписанный **07.07.98** и распространяемый фирмой Microsoft Corporation. Подлинность распространителя установлена организацией **VeriSign**.

Выделенная полужирным шрифтом информация в предыдущем абзаце взята из цифровой подписи, которой снабжен элемент управления **Microsoft Windows Media Player**. Пользователь по отображенной информации решает, можно ли разрешить загрузку элемента управления или нет.

Рис. 11.10. Диалоговое окно безопасности браузера Internet Explorer 4.0

Примечание

Если элемент управления ActiveX был однажды установлен на компьютере пользователя, то при загрузке новых версий этого же элемента управления браузер не проверяет наличие цифровой подписи, а без всякого предупреждающего сообщения устанавливает новую версию элемента управления.

Предупреждение

Подумайте, стоит ли загружать элемент управления ActiveX от непроверенного поставщика и, в особенности, без цифровой подписи, так как следующие версии устанавливаются без предупреждающего сообщения и потенциально могут нанести вред компьютеру.

Каким образом разработчик элемента управления ActiveX снабжает свой элемент цифровой подписью? Для этого он, прежде всего, получает соответствующий сертификат подлинности от организации, уполномоченной на это. Одной из таких организаций является VeriSign, сервер которой можно найти по адресу **http://digital.verisign.com/**.

Основная функция организации типа VeriSign — проверить подлинность и правильность информации о фирме или индивидуальном программисте, желающих подписывать свои продукты. Сертификат, по существу, является цифровой идентификационной карточкой производителя программного обеспечения и аналогичен товарному знаку на рынке обычных продуктов. Приходя в магазин и покупая товар, мы обращаем внимание на фирму-производителя, которую идентифицируем по зарегистрированной торговой марке. Точно так же мы можем теперь делать и в сети Интернет с загружаемым программным обеспечением.

Существует два типа сертификатов, выдаваемых для создания цифровой подписи: сертификат класса 2 и сертификат класса 3.

Сертификат класса 2 выдается индивидуальному программисту, публикующему в Интернете свои программы. Для его получения он должен представить в организацию, выдающую сертификаты, информацию о себе: имя, адрес, адрес электронной почты, дату рождения и номер карточки социального страхования (Social Security Number). После проверки предоставленной информации ему выдается сертификат сроком на один год. Заметим, что при получении сертификата он обязан оплатить его стоимость в размере 20 долларов США.

Сертификат класса 3 выдается фирмам-производителям программного обеспечения, и его стоимость составляет 400 долларов США в год. Для его получения необходимо представить название фирмы, место регистрации, контактные адреса и телефоны, а также рейтинг фирмы в соответствии с методикой Dun-and-Bradstreet.

Как обратил внимание читатель, сертификат имеет ограниченный срок действия, и по истечении этого срока требует своего возобновления.

После получения сертификата с помощью специального программного обеспечения, поставляемого разными фирмами, производитель программного продукта вставляет цифровую подпись в двоичный код распространяемого им программного обеспечения, например элемента управления ActiveX.

Таким образом, действующая цифровая подпись гарантирует, что загружаемый элемент управления ActiveX произведен конкретной фирмой или лицом, и подтверждает, что он никем не был изменен со дня его подписания фирмой-производителем. Однако она еще не гарантирует безопасности элемента управления ActiveX.

Безопасное использование элементов управления ActiveX

Безопасность — понятие достаточно субъективное. То, что для одного пользователя может казаться потенциально опасным, другому пользователю покажется совершенно безобидным. Но существует ряд общепризнанных действий, которые не должен выполнять безопасный элемент управления ActiveX. К ним относятся:

- Доступ к информации, расположенной на локальном компьютере, включая информацию о пользователе.
- Предоставление закрытой, частной информации о локальном компьютере или сети.
- Модификация или разрушение информации, находящейся на локальном компьютере или в сети.

- Ошибки в работе элемента управления, которые потенциально могут полностью вывести из строя браузер.
- Полный захват процессора или памяти компьютера во время своей работы.
- Выполнение потенциально опасных системных команд, включая выполнение загрузочных модулей.

Чтобы пользователь элемента управления ActiveX был уверен в том, что все перечисленные действия не выполняются приобретенным элементом управления, разработчик обычно включает в процедуру установки элемента его регистрацию в реестре как безопасного.

Существует два типа безопасности, применимых не только к элементам управления ActiveX, но и любым выполняемым программным объектам: *безопасность при инициализации* и *безопасность использования во встроенных сценариях.*

Для инициализации элементу управления ActiveX могут потребоваться данные, задаваемые пользователем или поступающие из внешних источников. В этом действии скрывается потенциальная возможность пробить брешь в безопасности компьютера пользователя. Внешние источники данных могут оказаться ненадежными, а заданные пользователем данные могут привести элемент управления в неуправляемое состояние, в котором он начнет, например, изменять значения ключей системного реестра. Элемент управления, который гарантирует надежность источника данных для инициализации, а также гарантирует, что он правильно обрабатывает любые пользовательские данные, не допуская разрушения системы локального компьютера, считается элементом управления, *безопасным при инициализации.*

Наибольший эффект от элемента управления ActiveX достигается его совместным использованием со встроенным на HTML-страницу сценарием. Может оказаться, что специально написанный сценарий для безопасного и полученного из надежного источника элемента управления ActiveX вызовет его методы и назначит такие значения его свойствам, что элемент управления опять начнет "ломать" систему безопасности компьютера или разрушать системные данные. Элемент управления, который гарантирует надежность применения его методов при любой последовательности их вызовов и при любых значениях параметров, а также при любых значениях его свойств, считается *надежным при использовании во встроенных сценариях.*

При установке на компьютере пользователя элемента управления ActiveX, помеченного разработчиком безопасным при инициализации и использовании в сценариях, в разделе системного реестра этого элемента управления создается ключ **Implemented Categories** (Реализованные категории), содержащий два подключа, отмечающие соответствующие категории безопасности элемента управления. Подключ `{7DD95801-9882-11CF-9FA9-00AA006C42C4}` соответствует безопасности при инициализации, а подключ `{7DD95802-9882-11CF-9FA9-00AA006C42C4}` — безопасности при использовании в сценарии.

На рис. 11.11 показано окно программы работы с реестром regedit.exe, в котором отображен раздел для элемента управления TDC. Видно, что этот элемент управления является безопасным во всех отношениях: ключ **Implemented Categories** содержит два подключа безопасности: `{7DD95801-9882-11CF-9FA9-00AA006C42C4}` и `{7DD95802-9882-11CF-9FA9-00AA006C42C4}`.

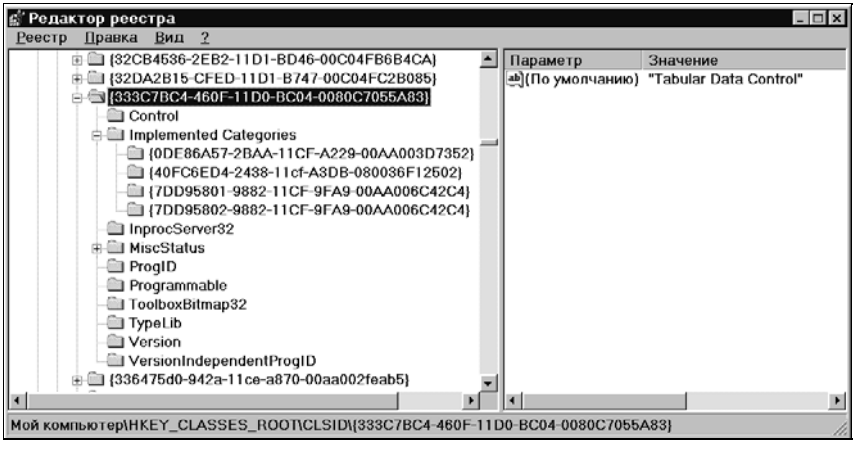


Рис. 11.11. Раздел реестра для элемента управления TDC

В разделе **Component Categories** реестра перечислены ключи для каждого вида выполняемых функций, которые реализованы или требуются компонентам и приложениям, установленным на компьютере. Этот раздел можно использовать для определения значения подключей, перечисленных в ключе **Implemented Categories** приложения или компонента. На рис. 11.12 показан раздел **Component Categories** с подключами в окне программы regedit.exe.

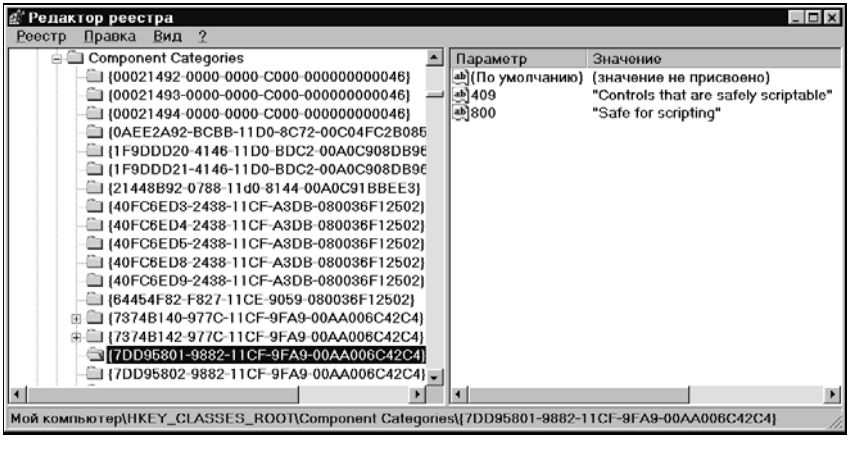


Рис. 11.12. Раздел **Component Categories** реестра

При использовании элемента управления на странице HTML браузер проверяет, отмечен ли он в системном реестре как безопасный для инициализации и использования в сценариях. Если да, то элемент управления используется для выполнения предусмотренных на странице действий, если нет — браузер отображает диалоговое окно (рис. 11.13) с предупреждением пользователю о небезопасности элемента управления ActiveX, и пользователь принимает решение о его использовании.

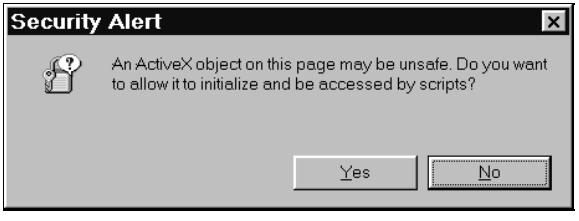


Рис. 11.13. Окно предупреждений браузера Internet Explorer 4.0 при загрузке небезопасного элемента управления ActiveX

Примечание

Предупреждение о небезопасности элемента управления ActiveX будет отображаться, даже если элемент управления разработан самим пользователем, но не помечен в реестре как безопасный.

Примечание

Если в процедуре установки элемента управления предусмотрена его регистрация как безопасного элемента управления, то он будет отмечаться таким во всех операционных системах. Автору элемента управления следует *всегда* проверять его безопасность во всех возможных операционных системах, используемых на компьютерах пользователей сети Интернет.

Лицензирование элементов управления ActiveX

Лицензирование программных продуктов, в том числе и элементов управления ActiveX, связано с их легальным, законным использованием. Применение "пиратских" версий элементов управления ActiveX на HTML-страницах может привести к осложнениям с фирмами-разработчиками, вплоть до судебного разбирательства. Каждая фирма, распространяющая свои программные продукты, разрабатывает и поддерживает собственные стратегии лицензирования. В этом разделе мы кратко остановимся на стратегии фирмы Microsoft, как одной из первых зачинательниц легального использования программных продуктов.

Большинство предлагаемых этой фирмой элементов управления ActiveX, если только они не распространятся бесплатно, должны поддерживать лицензирование во *время разработки* и во *время выполнения*. Лицензирование

во время разработки гарантирует, что автор Web-документа или программного обеспечения использует легально приобретенные элементы управления. Лицензирование во время выполнения гарантирует, что пользователь просматривает Web-документ или применяет программное обеспечение с легально приобретенными элементами управления.

Лицензия элемента управления во время разработки проверяется программой-контейнером, которая встраивает элемент управления в разрабатываемый продукт. Визуальные средства разработки фирмы Microsoft, например Visual Basic и Visual InterDev, прежде чем поместить элемент управления в программу или Web-документ, проверяют наличие ключа лицензии. Если такового не обнаружится, то соответствующий элемент управления не может быть использован в разработке, при этом отображается диалоговое окно с описанием причины отказа в использовании этого элемента управления. На рис. 11.14 показано диалоговое окно, отображаемое Visual Basic 6.0 (русская редакция) при попытке использовать нелицензированный элемент управления.

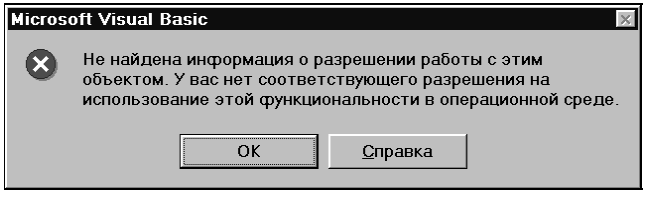


Рис. 11.14. Окно предупреждений Visual Basic при встраивании нелицензированного элемента управления ActiveX

За проверку лицензии во время выполнения также отвечают упомянутые выше средства разработки. Они встраивают лицензию на использование элемента управления во время разработки в программу или Web-документ, создаваемый этими средствами.

MS Internet Explorer — это другой тип программы-контейнера, использующего элементы управления ActiveX. В отличие от программных средств разработки, которые встраивают двоичный код элемента управления в выполняемый файл разрабатываемого приложения, Internet Explorer сначала загружает элемент управления на компьютер пользователя, а потом проверяет его лицензию на использование перед отображением на странице. Чтобы предупредить пиратское использование элемента управления, так как пользователь в любой момент может увидеть исходный текст HTML-страницы, механизм проверки легальности прибегает к "сокрытию" от пользователя лицензии на использование элемента управления ActiveX. Это достигается использованием файла упаковки лицензии (license package file) с расширением LPK, который встраивается в HTML-страницу с помощью тэга `<OBJECT>`. Параметр `CLSID` идентифицирует этот объект как файл упаковки лицензии, а тэг `PARAM` определяет его адрес относительно адреса HTML-страницы:

```
<OBJECT CLASSID="clsid:5220CB21-C88D-11CF-B347-00AA00A28331">
    <PARAM NAME="LPKPath" VALUE="time.lpk">
</OBJECT>
```

Уникальный идентификационный номер, заданный в этом примере, определяет объект как файл упаковки лицензии, а значение параметра `LPKPath` в тэге `<PARAM>` указывает, что этот файл имеет имя `time.lpk` и расположен в том же каталоге и на том же сервере, что и просматриваемая страница.

Файл упаковки лицензии создается специальной программой Lpk_Tool.exe из средств разработки приложений клиента Интернета и содержит пары `CLSID` элемента управления — лицензионный ключ. Браузер Internet Explorer специальной программой ActiveX License Manager выделяет для элемента управления лицензионный ключ и сравнивает его с ключом загруженного элемента управления. Если они совпадают, то элемент управления ActiveX отображается на странице, если не совпадают, то элемент управления не отображается и страница теряет свою функциональность, связанную с использованием этого элемента управления.

Совет

Разработчикам HTML-страниц рекомендуется использовать файл упаковки лицензии для элементов управления, приобретенных ими у третьих лиц.

Элементы управления на HTML-страницах

В этом разделе мы приведем примеры использования элементов управления ActiveX для расширения функциональных возможностей HTML-страниц. Вероятно, читатель заметит, что некоторые "трюки" можно было бы сделать и с помощью средств динамического HTML, но следует заметить, что элементы управления ActiveX продолжают оставаться мощным средством реализации функциональности, и практика их использования расширит кругозор разработчиков Web-приложений. Если читатель знаком с Visual Basic, то он уже работал с элементами управления ActiveX, и этот опыт он может использовать, привлекая известные ему элементы управления для разработки Web-приложений на языке VBScript.

Сразу оговоримся, что современные элементы управления ActiveX содержат большой спектр функциональности, который реализуется раскрытием свойств и методов элемента управления программе-контейнеру, причем сложная функциональность требует наличия у элемента управления свойств, значениями которых являются некоторые объекты. Это приводит к тому, что значения многих свойств наиболее популярных элементов управления ActiveX уже нельзя определить в тэге `<PARAM>`. Отсюда возникает необходимость инициализации параметров элементов управления и организации дальнейшей работы с ними осуществлять из специально создаваемых на страницах

сценариях, привязываемых к допустимым событиям элементов управления ActiveX. В этом разделе в качестве языка сценария используется язык VBScript, хотя все то же самое можно осуществить и с помощью языка JavaScript.

Элемент управления *TabStrip*

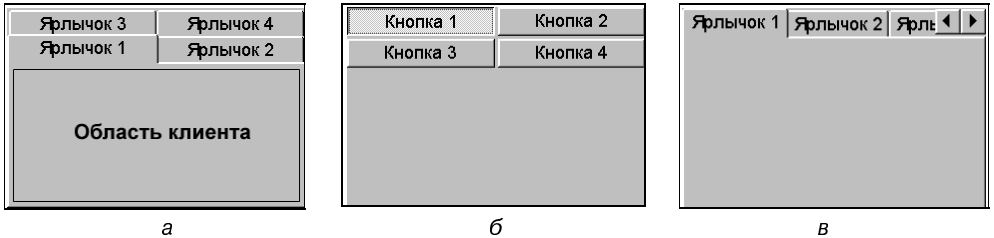
Элемент управления **TabStrip** напоминает закладки в записной книжке или разделители в библиотечной картотеке. Он состоит из нескольких ярлычков и области клиента, в которой отображается связанная с каждым ярлычком информация, размещаемая в расположенных там элементах управления ActiveX (отсюда и название этой области). Стандартный вид элемента управления **TabStrip** показан на рис. 11.15, *а*, где также черными линиями обозначена граница области клиента. При щелчке на каком-либо ярлычке он как бы располагается перед оставшимися ярлычками, и в области клиента отображается связанная с этим ярлычком информация.

Примечание

Элемент управления **TabStrip** используется для создания диалоговых окон со вкладками. Каждая вкладка — это ярлычок с соответствующим ему содержимым области клиента.

Использование элемента управления **TabStrip** экономит место на странице Web-документа. Множественная информация отображается поочередно в одном месте на странице — области клиента элемента управления.

Свойство `Style` этого элемента управления позволяет ярлычки-закладки (значение свойства равно 0) заменить кнопками управления (значение свойства равно 1). Внешний вид элемента управления **TabStrip** с кнопками вместо ярлычков показан на рис. 11.15, *б*.



а *б* *в*

Рис. 11.15. Элемент управления **TabStrip** (Microsoft 6.0)

У этого элемента управления есть очень полезное свойство: `MultiRow`. Если оно равно `True`, то отображаются *все* ярлычки или кнопки, в случае необходимости, в несколько рядов (рис. 11.15 *а*, *б*). Дополнительные ряды создаются за счет области клиента, уменьшая ее высоту. Установка значения

этого свойства равным `False` отображает все ярлычки или кнопки в один ряд. Если их общая ширина превышает ширину самого элемента управления, то в правом верхнем углу добавляются две кнопки, позволяющие перемещаться по ярлычкам или кнопкам (рис. 11.15, *в*).

Элемент управления **TabStrip** в таком виде удобен для использования в визуальных системах программирования, в которых необходимые элементы управления ActiveX перетаскиваются мышью в область клиента.

В Web-приложениях программирование этого элемента управления требует определенных усилий, однако можно использовать только ярлычки этого элемента управления, установив высоту всего элемента управления равной высоте ярлычка, и отображая их в один ряд (`Style=1`), для отображения в одной области страницы разнообразной информации. Например, таким способом можно создать галерею картин какого-либо художника, образцов товаров и т. п.

В качестве примера такого использования элемента управления **TabStrip** создадим страницу интерактивного учебного пособия, содержащую галерею элементов управления ActiveX.

На этой странице ярлычки элемента управления **TabStrip** будут использоваться для отображения в плавающем фрейме информации об элементе управления ActiveX, имя которого будет написано на выбранном пользователем ярлычке.

Прежде всего, встроим на страницу сам элемент и создадим плавающий фрейм. Для лучшего восприятия "связанности" этих объектов разместим их в одной таблице в двух соседних строках. Исходный текст HTML, размещаемый в тэге `<BODY>`, показан ниже:

```
<body bgcolor="lightgrey">
<H2 STYLE="color:red; backgrounr-color:lightgrey">
ГАЛЕРЕЯ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ ActiveX</H2>
<table border="1">
  <tr><td>
       <object ID="TabStrip1"
               WIDTH="720" HEIGHT="33"
               CLASSID="clsid:1EFB6596-857C-11D1-B16A-00C0F0283628">
       </object>
      </td>
  </tr>
  <tr><td>
       <IFRAME ID="FloatingFrame" HEIGHT="400" WIDTH="720">
       </IFRAME>
      </td>
  </tr>
</table>
</body>
```

Обратите внимание на уникальный идентификационный номер элемента управления ActiveX в параметре `CLASSID` тэга `<OBJECT>`. Это идентификационный номер элемента управления **TabStrip** (Microsoft 6.0), под которым он зарегистрирован в системном реестре.

Если отобразить эту страницу в окне браузера, то мы увидим картинку, представленную на рис. 11.16, *а*.

а *б*

Рис. 11.16. Отображение страницы примера и элемента **TabStrip** без параметров

Виден без надписи один ярлычок элемента управления **TabStrip** (его высота определена в параметре `HEIGHT`), ниже которого расположен плавающий фрейм. На рис. 11.16, *б* показан элемент управления **TabStrip**, если бы его высота была определена равной 400 пикселов, и он точно так же, как и в нашем примере, был бы встроен в страницу без задания каких-либо своих параметров.

Определим вид элемента управления **TabStrip**, задавая значения его свойств в процедуре обработки события `OnLoad` загрузки содержимого окна браузера.

Каждый элемент управления **TabStrip** состоит из одного или нескольких объектов `Tab` (Вкладка), которые хранятся в наборе `Tabs` элемента управления. Объект `Tab` представляет собой ярлычок и соответствующую ему область клиента. Его свойство `Caption` определяет надпись на ярлычке этого объекта. Например, если в программе необходимо задать надпись "Ярлычок 1" на первом ярлычке элемента управления **TabStrip**, то это можно сделать следующим оператором:

```
TabStrip1.Tabs(1).Caption="Ярлычок 1"
```

Здесь `TabStrip1` — имя объекта, соответствующего элементу управления **TabStrip**.

Но прежде чем работать с объектами `Tab`, их нужно создать. Из рис. 11.16 ясно, что при инициализации элемента управления **TabStrip** по умолчанию

создается только одна вкладка. Метод `Add()` набора `Tabs` служит для создания новой вкладки (объект `Tab`) элемента управления **TabStrip**. Следующий оператор создает еще одну вкладку элемента управления **TabStrip1**:

```
TabStrip1.Tabs.Add()
```

Теперь, если предположить, что в массиве `Name` размерности `N` хранятся имена ярлычков, то создание `N` вкладок элемента управления **TabStrip** с именем `TabStrip1` в процедуре обработки события `OnLoad` окна браузера будет выглядеть следующим образом:

```
Sub Window_OnLoad()
' Создание вкладок элемента управления TabStrip1
  For I=1 To N
    TabStrip1.Tabs.Add()
    TabStrip1.Tabs(I).Caption=Name(I)
  Next

  TabStrip1.object.Style=0 ' Стиль отображения ярлычков
  TabStrip1.MultiRow=False ' Не отображать ярлычки в несколько рядов
  TabStrip1.Font.Size="10"    ' Размер шрифта текста ярлычков
  TabStrip1.Font.Name="Arial" ' Тип шрифта текста ярлычков
End Sub
```

В этой же процедуре определяется стиль и параметры шрифта элемента управления **TabStrip1**. Свойство `Font` элемента управления хранит ссылку на объект `Font` (Шрифт), который определяет параметры шрифта надписей ярлычков.

Примечание

Так как объект `TabStrip1`, как элемент объектной модели документа, имеет свойство `Style`, определяющее свойства каскадных таблиц стилей этого элемента, то ссылка на свойство `Style` элемента управления **TabStrip**, представленного объектом `TabStrip1`, осуществляется через свойство `object` *(см. выше раздел "Элементы управления ActiveX и сценарии")*.

Если эту процедуру добавить в разрабатываемую нами HTML-страницу, а также определить значение переменной `N`, представляющей число элементов управления ActiveX в нашей галерее, и названия этих элементов управления в элементах массива `Name`, то при загрузке страницы будут отображаться ярлычки с названиями элементов управления ActiveX и плавающий фрейм.

Теперь следует связать выбор каждого ярлычка с загрузкой в плавающий фрейм HTML-страницы, описывающей соответствующий названию ярлычка элемент управления ActiveX. Это осуществляется в процедуре обработки события `Click` элемента управления **TabStrip**, которое генерируется при щелчке кнопкой мыши на каком-нибудь ярлычке элемента управления:

```
Function TabStrip1_Click()
 Window.FloatingFrame.Location.Href = "actx-" & _
               TabStrip1.SelectedItem.Index & ".html"
End Function
```

Чтобы загрузить HTML-страницу в плавающий фрейм, необходимо присвоить адрес этой страницы свойству `Href` объекта `Location` этого фрейма. Если страница расположена на том же сервере и в том же каталоге, что и исходная HTML-страница, то достаточно задать только имя загружаемой страницы. В нашем примере при щелчке на первом ярлычке будет загружаться файл `actx-1.html`, при щелчке на втором — `actx-2.html` и т. д. Свойство `SelectedItem` элемента управления **TabStrip** содержит объект `Tab`, соответствующий выбранной вкладке, а свойство `Index` хранит его индекс в наборе `Tabs`.

Предупреждение

При именовании процедуры обработки событий элемента управления ActiveX имя события задается без префикса `on` (в соответствии с правилами именования VBScript).

Суммируя все вместе на одной HTML-странице, получаем окончательный исходный текст нашего приложения.

Пример 11.2. Галерея элементов управления ActiveX

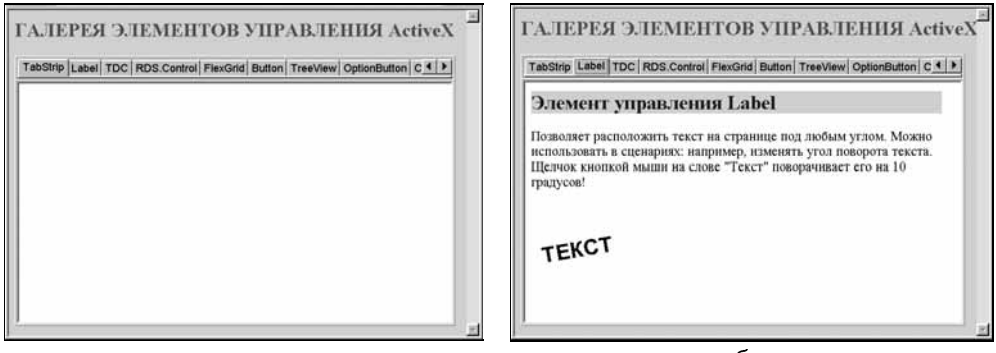
```
<head>
<script LANGUAGE="VBScript" TYPE="text/vbscript">
<!--
Dim Name()
N=12
ReDim Name(N)
Name(1)="TabStrip"
Name(2)="Label"
Name(3)="TDC"
Name(4)="RDS.Control"
Name(5)="FlexGrid"
Name(6)="Button"
Name(7)="TreeView"
Name(8)="OptionButton"
Name(9)="CheckList"
Name(10)="Slider"
Name(11)="Image"
Name(12)="Picture"

Sub Window_OnLoad()
      For I=1 To N
```

```
        TabStrip1.Tabs.Add()
        TabStrip1.Tabs(I).Caption=Name(I)
      Next

        TabStrip1.object.Style=0
        TabStrip1.MultiRow=False
        TabStrip1.Font.Size="10"
        TabStrip1.Font.Name="Arial"
End Sub

Function TabStrip1_Click()
 Window.FloatingFrame.Location.Href = "actx-" &
TabStrip1.SelectedItem.Index & ".html"
End Function
'-->
</script>
</head>
<body bgcolor="lightgrey">
<H2 STYLE="color:red; backgrounr-color:lightgrey">
ГАЛЕРЕЯ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ ActiveX</H2>
<table border="1">
  <tr>
    <td>
<object ID="TabStrip1"
  WIDTH="720" HEIGHT="33"
  CLASSID="clsid:1EFB6596-857C-11D1-B16A-00C0F0283628">
</object>
    </td>
  </tr>
  <tr>
    <td>
<IFRAME ID="FloatingFrame" HEIGHT="400" WIDTH="720">
</IFRAME>
    </td>
  </tr>
</table>
</body>
```

Обратим внимание, что массив `Name` задается как динамический. Это сделано для того, чтобы легче можно было добавлять описание дополнительных элементов управления ActiveX.

Результаты загрузки в браузер разработанного Web-приложения можно видеть на рис. 11.17.

На рис. 11.17, *а* показано окно браузера при первоначальной загрузке страницы примера 11.2, а на рис. 11.17, *б* — при выборе элемента управления **Label**.



а *б*

Рис. 11.17. Приложение с элементом управления **TabStrip**

Аналогично можно сделать галерею картин любимого художника, образцов производимой продукции и т. д.

Элемент управления *TreeView*

Элемент управления **TreeView**, общий вид которого показан на рис. 11.18, предназначен для отображения в компактном виде данных древовидной структуры. Этот элемент управления часто используется для представления оглавления книг, иерархической структуры базы данных, каталогов данных на жестком диске компьютера и т. д. На рис. 11.18 он отражает оглавление книги "Справочник Web-мастера" издательства BHV.

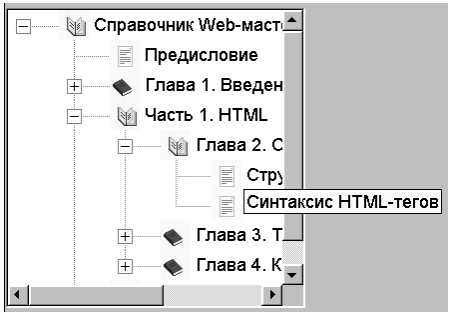


Рис. 11.18. Общий вид элемента управления **TreeView**

Основной единицей этого элемента управления ActiveX является *узел*, который в самом общем случае состоит из изображения и метки. Изображения для узлов поставляются элементом управления **ImageList**, ассоциированным с элементом управления **TreeView**. Например, самый первый узел на рис. 11.18 состоит из изображения раскрытой книги и метки, задающей отображаемый текст этого узла ("Справочник Web-мастера").

Узлы в дереве могут содержать подчиненные узлы. В этом случае их можно раскрыть, показав все подчиненные узлы, или свернуть. Такие узлы отобра-

жаются с расположенным слева от изображения небольшим квадратиком (его еще называют флажком), в котором располагается знак + (плюс), если узел свернут, и знак — (минус), если узел раскрыт. Для свернутого или раскрытого узла можно использовать разные изображения, как показано на рис. 11.18. Двойной щелчок на узле или одинарный щелчок на квадратике раскрывает или сворачивает узел с подчиненными узлами.

Если структура дерева при раскрытии узлов не помещается в области отображения элемента управления, то автоматически добавляются вертикальная и горизонтальная полосы прокрутки. Если надпись в узле полностью не помещается в пространстве отображения элемента управления, то при расположении курсора мыши над таким узлом появляется всплывающая подсказка, в которой отображается весь текст, содержащийся в метке узла, как видно на рис. 11.18.

Каждый узел реализован в виде программируемого объекта `Node` (Узел). Эти объекты хранятся в наборе `Nodes` элемента управления **TreeView**. Свойство `Index` узла позволяет определить его индекс в наборе `Nodes`. Узлы помещаются в этот набор в соответствии с порядком их задания в программе.

Каждый узел может быть либо только родителем других узлов (тогда он является началом дерева и называется корневым узлом), либо только порожденным узлом (тогда он завершает ветвь дерева), либо и родителем, и порожденным узлом одновременно (тогда он расположен в точке ветвления дерева). Корневых узлов может быть несколько. Они не связаны между собой никакими "родственными" связями, и порождают, каждый в отдельности, самостоятельные деревья иерархической структуры. Любое дерево может иметь только один корневой элемент.

Для определения "родства" узлов объект `Node` имеет набор свойств, позволяющих определить родителя (свойство `Parent`), первый порожденный узел (свойство `Child`), количество порожденных узлов (свойство `Children`) и корневой узел (свойство `Root`).

Узлы, непосредственно порождаемые каким-либо узлом, являются узлами-"братьями", или узлами одного уровня. Ряд свойств позволяет обрабатывать подобные "родственные" связи: `FirstSibling` возвращает ссылку на первый узел из множества узлов одного уровня, `LastSibling` — на последний узел из этого множества, `Next` определяет следующего брата, а `Previous` — предыдущего.

Перечисленные свойства объекта `Node` позволяют полностью определить структуру данных, отображаемую элементом управления **TreeView**, и организовать перемещение по ее узлам и ветвям.

Но как создать узлы, как определить, какой узел порождает другой, как задать корневой узел дерева? Для этого предназначен метод `Add` набора `Nodes`, имеющий следующий синтаксис:

```
объектTreeView.Nodes.Add узел, отнош, ключ, текст, изобр, выбр_изобр
```

Первый параметр `узел` этого метода определяет объект `Node`, относительно которого создается новый узел элемента управления **TreeView** с заданным параметром `отнош` "родственным" отношением. Параметр `узел` задает либо индекс, либо уникальный ключ существующего объекта `Node` (*см. ниже*). Параметр `отнош` может быть целым числом от 0 до 4. Описание всех значений этого параметра и определяемых ими отношений представлено в табл. 11.1.

Таблица 11.1. *Родственные отношения узлов в методе* `Add`

Значение параметра `отнош`	**Описание**
0	Новый узел создается перед всеми узлами того же уровня, что и узел, определяемый параметром `узел`
1	Новый узел создается после всех узлов того же уровня, что и узел, определяемый параметром `узел`
2	Новый узел создается непосредственно после узла, определяемого параметром `узел`
3	Новый узел создается непосредственно перед узлом, определяемым параметром `узел`
4	Новый узел создается как узел, порождаемый узлом, который определен параметром `узел`

Примечание

Если отсутствует параметр `узел`, то метод `Add` создает корневой узел дерева.

Параметр `ключ` задает уникальный идентификатор, или уникальный ключ узла, по которому можно ссылаться на создаваемый узел. Его значением является символьная строка.

Параметр `текст` определяет строку текста, появляющуюся в метке узла при отображении в браузере элемента управления.

Последние два параметра `изобр` и `выбр_изобр` определяют, соответственно, графическое изображение в узле и графическое изображение, когда узел выбран щелчком кнопки или с помощью клавиш со стрелками. Их значениями являются ссылки на графические изображения, хранящиеся в элементе управления **ImageList**, присоединенном к элементу управления **TreeView** *(см. ниже)*.

Из всех шести параметров этого метода только параметр `текст` является обязательным. Запятые, определяющие положение отсутствующих параметров, обязательны, если эти параметры расположены раньше параметра `текст` в списке параметров.

Структура содержания книги, отображаемая в элементе управления TreeView на рис. 11.18, создана с помощью следующих операторов:

```
imgList1.ListImages.Add,"open", LoadPicture("D:/Picture/open.jpg")
imgList1.ListImages.Add,"close", LoadPicture("D:/Picture/close.jpg")
imgList1.ListImages.Add,"leaf", LoadPicture("D:/Picture/leaf.jpg")
TreeView1.ImageList=imgList1

TreeView1.Nodes.Add ,,"R","Справочник Web-мастера" ","close"

TreeView1.Nodes.Add "R", 4,"Intr","Предисловие" ","leaf"
TreeView1.Nodes.Add "R", 4,"Ch1","Глава 1. Введение" ","close"
TreeView1.Nodes.Add "Ch1", 4,"Par1","О Web в двух словах" ","leaf"
TreeView1.Nodes.Add "Ch1", 4,"Par2","Кто такие Web-мастера?" ","leaf"
TreeView1.Nodes.Add "Ch1", 4,"Par3","Рекомендуемая литература" ","leaf"

TreeView1.Nodes.Add "R", 4,"Part1","Часть 1. HTML","close"
TreeView1.Nodes.Add "Part1", 4,"Ch2","Глава 2. Обзор HTML","close"
TreeView1.Nodes.Add "Ch2", 4,"Par1-2","Структура HTML-документа","leaf"
TreeView1.Nodes.Add "Ch2", 4,"Par2-2","Синтаксис HTML-тэгов","leaf"

TreeView1.Nodes.Add "Part1", 4,"Ch3","Глава 11. Тэги HTML","close"
TreeView1.Nodes.Add "Ch3", 4,"Par1-3", _
                    "Описание тэгов HTML и их параметров","leaf"

TreeView1.Nodes.Add "Part1", 4,"Ch4","Глава 4. Фреймы","close"
TreeView1.Nodes.Add "Ch4", 4,"Par1-4","Компоновка фреймов","leaf"
TreeView1.Nodes.Add "Ch4", 4,"Par2-4","Вложенные наборы фреймов","leaf"
TreeView1.Nodes.Add "Ch4", 4,"Par3-4","Тэг <FRAME>","leaf"
TreeView1.Nodes.Add "Ch4", 4,"Par4-4","Целевые фреймы","leaf"
TreeView1.Nodes.Add "Ch4", 4,"Par5-4","Параметры обрамления
фреймов","leaf"

TreeView1.Nodes.Add "R", 4,"Part2","Часть 2. CGI","close"
```

Первые три оператора этого фрагмента определяют список графических изображений, хранящихся в элементе управления **ImageList**, который определен переменной `ImageList1`. Для добавления ссылок на графические изображения предназначен метод `Add` набора `ListImages` элемента управления. Именно этот набор содержит ссылки на все графические изображения, хранящиеся в элементе управления **ImageList**. Для работы метода `Add` необходимо задание трех параметров. Первый параметр, не обязательный, определяет индекс в наборе `ListImages` включаемого изображения. Второй параметр задает уникальный ключ изображения, который используется в качестве значений параметров `изобр` и `выбр_изобр` метода `Add` набора `Nodes` элемента управления **TreeView**. Последний параметр определяет файл включаемого изображения, причем он определяется с помощью функции `LoadPicture(имя_файла)`.

Четвертым оператором осуществляется привязка элемента управления `ImageList1` к элементу управления **TreeView**, определяемому переменной `TreeView1`. Это делается простым присваиванием свойству `ImageList` элемента управления `TreeView1` ссылки на элемент управления `ImageList1`.

Остальные операторы приведенного фрагмента текста сценария задают узлы отображаемой структуры содержания книги и не требуют пояснений.

Простое отображение содержания книги в компактной форме достаточно полезная вещь, но если к нему добавить возможность отображения содержимого книги, то мы получим электронный вариант книги. Далее в этом разделе мы покажем, как обрабатывать события элемента управления **TreeView**, чтобы при раскрытии узлов менялось его изображение, как связать выбор завершающего узла дерева с отображением ассоциированного с ним файла. Решение этих задач покажем на примере создания электронного варианта книги простой структуры — книги, состоящей из трех простых глав без разделов.

Создадим HTML-страницу, на которой разместим элемент управления **TreeView**, содержащий оглавление книги, и плавающий фрейм, в котором будет отображаться содержимое глав при щелчке кнопкой мыши на соответствующем узле элемента управления **TreeView**:

```
<body bgcolor="lightgrey">

<p>
<!—Элемент управления TreeView-->
<object id="treeView1"
        classid="clsid:C74190B6-8589-11D1-B16A-00C0F0283628"
        width="250" height="300">
</object>
<!—Элемент управления ImageList-->
<object id="imgList1" width=0 height=0
        classid="clsid:2C247F23-8591-11D1-B16A-00C0F0283628">
</object>
     
<IFRAME ID="iFrame1" WIDTH=680 HEIGHT=300>
</IFRAME>
</p>
</body>
```

Обратите внимание, что кроме элемента управления **TreeView** на страницу помещен и элемент управления **ImageView**, который необходим для привязки графических файлов изображений в узлах иерархической структуры, отображаемой элементом управления **TreeView**. Так как этот элемент управления не имеет видимого интерфейса, то значения параметров `WIDTH` и `HEIGHT`, определяющих размеры, не влияют на его отображение и установлены равными нулю.

Инициализацию элементов управления, как обычно, произведем в процедуре обработки события `Load` (Загрузка) объекта `window`. Но прежде создадим графические изображения, которые будут использоваться в раскрываемых и завершающих ветвях дерева оглавления узлах. Их можно создать в любом графическом редакторе, причем размеры не имеют значения, так как при отображении их в узлах они автоматически масштабируются. Для свернутого узла будем использовать изображение из файла "open.jpg", для раскрытого узла — из файла "close.jpg" и для завершающего ветвь узла — из файла "leaf.jpg". Изображения из этих файлов показаны на рис. 11.19.

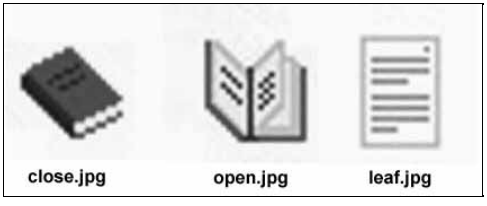


Рис. 11.19. Графические изображения, используемые в узлах дерева оглавления

Теперь, создав графические изображения, необходимо поместить их в элемент управления **imgList1** и связать его с элементом управления **TreeView1** до того, как начинать создавать структуру оглавления. Ниже представлен код процедуры обработки события загрузки окна:

```
Sub Window_onLoad()
   imgList1.ListImages.Add,"open", LoadPicture("D:/Picture/open.jpg")
   imgList1.ListImages.Add,"close", LoadPicture("D:/Picture/close.jpg")
   imgList1.ListImages.Add,"leaf", LoadPicture("D:/Picture/leaf.jpg")
   TreeView1.ImageList=imgList1

   TreeView1.Nodes.Add ,,"R","Интерактивные Web-документы","close"
   TreeView1.Nodes.Add "R", 4,"Ch1","Глава 1","leaf"
   TreeView1.Nodes.Add "R", 4,"Ch2","Глава 2","leaf"
   TreeView1.Nodes.Add "R", 4,"Ch3","Глава 3","leaf"
End Sub
```

Пока наше приложение, если его поместить в окно браузера, не будет отображать никаких глав в плавающем фрейме, и более того, не будет меняться изображение в корневом узле при его раскрытии, как показано на рис. 11.20.

Для того чтобы изображение узла менялось при его раскрытии и сворачивании, следует написать процедуры обработки событий `DblClick`, `Expand` и `Collapse` элемента управления **TreeView1**. При двойном щелчке кнопкой мыши на узле, имеющем подчиненные узлы, он раскрывается или сворачивается. События `Expand` (Раскрыть) и `Collapse` (Свернуть) генерируются при щелчке на квадратике, отображаемом слева от раскрываемого узла: первое, когда узел свернут, а второе, когда узел был раскрыт. В процедурах обработки этих событий необходимо просто заменить изображение в узле, соответ-

ственно, для события `Expand` на файл "open.jpg", а для события `Collapse` на файл "close.jpg", тогда как в процедуре обработки двойного щелчка на узле следует проверить, раскрываемый ли это узел и в каком состоянии он находился: был раскрыт или свернут. После такой проверки заменить изображение на противоположное. Ниже представлены исходные тексты процедур обработки событий:

```
Sub TreeView1_DblClick()
  If TreeView1.SelectedItem.Children <> 0 Then
    If TreeView1.SelectedItem.Expanded = True Then
      TreeView1.SelectedItem.Image = "open"
    Else
      TreeView1.SelectedItem.Image = "close"
    End If
  End If
End Sub

Sub TreeView1_Expand(ByVal Node)
  Node.Image = "open"
End Sub

Sub TreeView1_Collapse(ByVal Node)
  Node.Image = "close"
End Sub
```

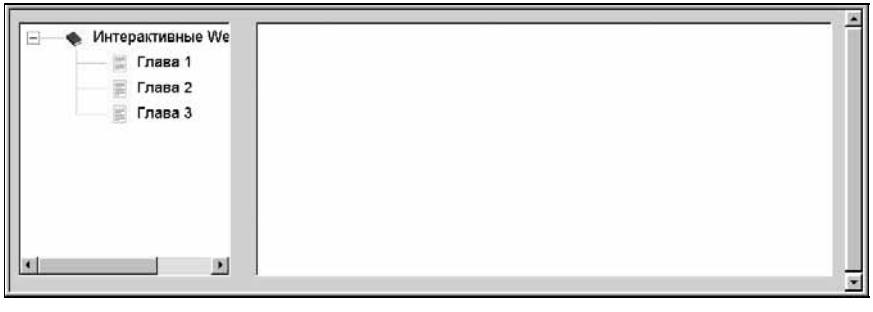


Рис. 11.20. Электронный вариант книги без обработки событий элемента управления **TreeView**

Процедурам обработки событий `Expand` и `Collapse` в качестве параметра передается объект `Node` (узел) элемента управления **TreeView1**, на флажке которого был произведен щелчок кнопкой мыши. Он используется в этих процедурах для установки нового значения своего свойства `Image`, в котором хранится ссылка на изображение.

В процедуре обработки события `DblClick` проверяется, имеет ли узел, на котором произведен щелчок (`TreeView1.SelectedItem`), порождаемые им узлы (свойство `Children` не равно нулю). Если имеет, то проверяется, раскрыт ли

он или нет. В зависимости от результатов проверки узлу назначается соответствующее изображение.

Теперь остается только при щелчке на завершающем узле ветви дерева оглавления загрузить в плавающий фрейм файл соответствующих глав книги. Это действие осуществляется в процедуре обработки события `Click` элемента управления `treeView1`:

```
Sub TreeView1_Click()
  If TreeView1.SelectedItem.Children = 0 Then
    Window.iFrame1.Location.Href = TreeView1.SelectedItem.Text & ".html"
  End If
End Sub
```

Содержимое глав книги хранится в файлах "`Глава 1.html`", "`Глава 2.html`" и "`Глава 3.html`", расположенных в том же каталоге, что и файл нашего приложения, окончательный, полный текст которого приведен в примере 11.3.

Пример 11.3. Электронный вариант книги

```
<html>
<head>
<SCRIPT Language="VBScript">
Sub Window_onLoad()
   imgList1.ListImages.Add,"open", LoadPicture("D:/Picture/open.jpg")
   imgList1.ListImages.Add,"close", LoadPicture("D:/Picture/close.jpg")
   imgList1.ListImages.Add,"leaf", LoadPicture("D:/Picture/leaf.jpg")
   TreeView1.ImageList=imgList1

   TreeView1.Nodes.Add ,,"R","Интерактивные Web-документы","close"
   TreeView1.Nodes.Add "R", 4,"Ch1","Глава 1","leaf"
   TreeView1.Nodes.Add "R", 4,"Ch2","Глава 2","leaf"
   TreeView1.Nodes.Add "R", 4,"Ch3","Глава 3","leaf"
   TreeView1.object.Style=7
   TreeView1.LineStyle=1
   TreeView1.Font.Size=10
   TreeView1.Font.Name="Arial"
End Sub

Sub TreeView1_Click()
  If TreeView1.SelectedItem.Children = 0 Then
    Window.iFrame1.Location.Href = TreeView1.SelectedItem.Text & ".html"
  End If
End Sub

Sub TreeView1_DblClick()
  If TreeView1.SelectedItem.Children <> 0 Then
```

```
    If TreeView1.SelectedItem.Expanded = True Then
      TreeView1.SelectedItem.Image = "open"
    Else
      TreeView1.SelectedItem.Image = "close"
    End If
  End If
End Sub

Sub TreeView1_Expand(ByVal Node)
  Node.Image="open"
End Sub

Sub TreeView1_Collapse(ByVal Node)
  Node.Image = "close"
End Sub
</SCRIPT>
</head>
<body bgcolor=lightgrey>

<p>
<object id="TreeView1"
        classid="clsid:C74190B6-8589-11D1-B16A-00C0F0283628"
        width="250" height="300">
</object>
<object id="imgList1" width=100 height=10
        classid="clsid:2C247F23-8591-11D1-B16A-00C0F0283628">
</object>
     
<IFRAME ID=iFrame1 WIDTH=680 HEIGHT=300>
</IFRAME>
</p>
</body>
</html>
```

Результаты отображения первой главы показаны на рис. 11.21.

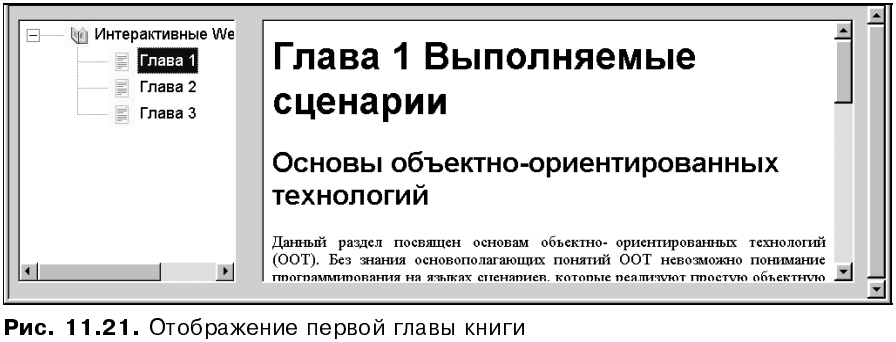


Рис. 11.21. Отображение первой главы книги

В процедуру обработки события загрузки окна введены дополнительные установки для элемента управления **TreeView**. Свойство `Font` устанавливает параметры шрифта, которым отображается текст в метках узлов, а свойство `Style` элемента управления **TreeView** определяет вид отображения узлов и всей иерархической структуры, представляемой этим элементом. Его значением может быть любое целое число из диапазона 0—7, определяющее, что отображается в узлах: 0 — только текст меток, 1 — изображение и текст, 2 — флажок и текст, 3 — флажок, изображение и текст, 4 — соединительные линии древовидной структуры и текст, 5 — линии, изображение и текст, 6 — линии, флажок и текст, 7 — линии, флажок, изображение и текст.

Предупреждение

Так как объект `TreeView`, как элемент объектной модели документа, имеет свойство `Style`, определяющее свойства каскадных таблиц стилей этого элемента, то ссылка на свойство `Style` элемента управления **TreeView**, представленного объектом `treeView1`, осуществляется через свойство `object` *(см. выше раздел "Элементы управления ActiveX и сценарии")*.

Java-апплеты

Java-апплеты — это еще один метод создания динамических HTML-документов. Технология Java была разработана относительно недавно (1995 г.) программистами фирмы Sun и в настоящее время продолжает быстро развиваться, занимая все более прочное положение на рынке программных средств для создания приложений Интернета. Она включает в себя *язык программирования* и *платформу*, для обозначения которых используется одно название — Java.

В одном из своих документов фирма Sun характеризует язык Java как простой, объектно-ориентированный, распределенный, интерпретируемый, надежный, защищенный, архитектурно-независимый, переносимый, высокопроизводительный, многопоточный, динамический язык программирования высокого уровня и поясняет, какой смысл имеют все эти определения.

Программа на языке Java транслируется компилятором в специальный байтовый код — Java *bytecode*, называемый также *J-кодом*, для выполнения которого требуется интерпретатор Java. Таким образом, язык Java является одновременно компилируемым и интерпретируемым. Интерпретатор Java — это приложение, предназначенное для конкретной аппаратно-программной платформы: PC-Windows, PC-Linux, Mac, UNIX-машины. J-код не зависит от платформы. Он может быть подготовлен, например, на рабочей станции Sun, работающей под управлением операционной системы Solaris, а выполнен на персональном компьютере под управлением Windows NT. Это обеспечивает архитектурную независимость и переносимость программ на языке Java. Байтовый код Java можно рассматривать как набор машинных команд

для некоторой виртуальной машины, реализуемой интерпретатором. Он может исполняться в любой среде, в которой выполняется виртуальная машина Java.

Основными единицами, из которых строится программа Java, являются *классы*. Классы образуют иерархическую древовидную структуру, лежащую в основе объектно-ориентированной модели языка. Класс представляет собой совокупность данных и *методов* для их обработки. Слово *метод* в терминологии объектно-ориентированной модели является синонимом слов *процедура* или *функция*. На базе уже существующего класса может быть определен новый класс, являющийся подклассом своего предшественника. Исходный класс является родительским классом или надклассом для своего подкласса. Подкласс наследует у родительского класса коллекцию данных и методов. Кроме них он может содержать собственные данные и методы.

Класс определяет целый тип данных. Конкретный экземпляр этого типа называется *объектом*. Написание Java-программы заключается в создании нового класса или совокупности связанных между собой классов, то есть в определении набора данных, описывающих решаемую программистом задачу, и в конструировании методов, реализующих обработку этих данных. При этом для порождения новых классов может использоваться существующая базовая коллекция классов. Исходный текст на языке Java, составляющий описание класса, сохраняется в файле, имя которого имеет расширение JAVA. Этот файл компилируется в J-код и сохраняется в файле с тем же именем и расширением CLASS. Скомпилированные классы Java в виде J-кода могут храниться локально на компьютере или распределенно в сети и динамически загружаться выполняющимися приложениями по мере необходимости.

Платформа Java состоит из виртуальной машины и интерфейса прикладного программирования Java API. Виртуальная машина предназначена для выполнения J-кода на различных платформах. Интерфейс прикладного программирования Java представляет собой большую коллекцию классов в формате J-кода, сгруппированных по своему назначению в отдельные пакеты. Список имеющихся пакетов занимает несколько страниц текста, и каждый из пакетов содержит коллекцию классов. Для работы с такой сложной структурой используется специальная схема именования пакетов и классов, напоминающая доменную систему имен Интернета (DNS). Например, имя `java.lang` определяет пакет, содержащий базовые классы самого языка Java. Имя `java.lang.Object` определяет класс `Object` внутри этого пакета. Пакеты, имена которых начинаются со слова `java`, составляют основу интерфейса прикладного программирования и входят в состав любой платформы Java. Кроме основной части API существуют стандартные расширения для работы с трехмерной графикой, анимацией и т. д.

Для создания Java-программ необходим набор средств разработчика Java Development Kit (JDK), включающий в себя базовую коллекцию классов,

компилятор `javac`, отладчик `jdb`, виртуальную машину `java` для выполнения готовых приложений и программу просмотра апплетов `appletviewer`. Существуют варианты набора JDK от фирм Sun и Microsoft и, кроме того, визуальные средства разработки фирм Microsoft, Borland, Symantec. Основным разработчиком, определяющим направление развития технологии Java, является фирма Sun, поэтому лучше использовать инструментальные средства именно этого производителя. К тому же визуальные средства не позволяют проследить все стадии разработки программы. Сведения о текущей версии и приобретении JDK можно получить на сервере Sun **http://www.javasoft.com**.

Приложения и апплеты

В среде Java существуют два основных типа программ: *приложения* (application) и *апплеты* (applets). Приложение — это самостоятельная программа, для выполнения которой требуется только наличие виртуальной машины Java. Апплет — это программа, которая предназначена для выполнения в составе Web-браузера или специальной программы просмотра апплетов.

Схема применения апплетов выглядит следующим образом. Апплет создается, компилируется и сохраняется на Web-сервере. В HTML-документ, выставляемый на сервере, при помощи специального тэга помещается ссылка на местоположение апплета. При получении документа с сервера браузер загружает апплет и начинает его выполнять. Такой подход к созданию динамических документов прямо противоположен подходу, который используется в программировании интерфейсов CGI: апплет выполняется на стороне клиента, а CGI-сценарий — на стороне сервера. Два основных браузера, Netscape и Internet Explorer, имеют в своем составе средства для выполнения апплетов, включающие виртуальную машину и набор классов.

Слово applet представляет собой уменьшительную форму слова application и может быть переведено как "приложеньице" или "программка". Это название подчеркивает, что апплет должен быть небольшой по размеру программой, выполняющей достаточно простые действия. Большие апплеты увеличивают нагрузку на сеть и требуют заметного времени для передачи браузеру, в результате чего утрачивается то, для чего они предназначены — динамичное, соответствующее скорости реакции пользователя, изменение отображаемого документа.

Существует потенциальная опасность того, что полученная из сети неизвестная программа, которой позволено выполняться на локальном компьютере, может оказать разрушающее воздействие на локальную систему или прочитать конфиденциальную информацию. Разрушающие воздействия могут выражаться, например, в удалении или искажении файлов, в запуске программ, потребляющих ограниченные ресурсы системы, что приводит к ее деградации и вынужденной перезагрузке, в уничтожении других приложе-

ний, выполняющихся параллельно. По этой причине апплеты должны в обязательном порядке удовлетворять некоторым ограничениям безопасности, предъявляемым как самим языком Java, так и браузерами, в составе которых выполняются апплеты.

Апплет не может:

- осуществлять операции чтения-записи в локальной файловой системе, то есть читать и изменять файлы, просматривать содержимое каталогов, создавать, удалять, переименовывать файлы и каталоги;
- устанавливать сетевые соединения с другими компьютерами, кроме компьютера, с которого он был загружен;
- запускать на выполнение другие программы на компьютере, на котором он сам выполняется;
- использовать библиотеки других языков программирования, например, загружать библиотеки C++, хотя сам язык Java такую возможность предоставляет;
- изменять системные параметры (а может читать только некоторые из них, например, имя и версию операционной системы).

Для проверки соблюдения правил безопасности существует специальный класс `java.lang.SecurityManager`. Экземпляр этого класса выполняется в составе виртуальной Java-машины браузера. Когда объект `java.lang.SecurityManager` обнаруживает нарушение правил безопасности, возникает *исключение* (понятие близкое понятию прерывание, означающее возникновение ситуации, требующей специальной обработки), которое апплет перехватывает и реагирует заданным образом.

Структура приложения

Мы установили, что апплет это особый тип Java-приложения. Прежде чем перейти к рассмотрению апплетов, рассмотрим простейший пример приложения, не являющегося апплетом. В соответствии с международной традицией изучение языка программирования начинается с написания программы "Hello, world!". Не будем ее нарушать. Создадим файл `HelloWorld.java`, содержащий следующий текст:

Пример 11.4. Программа "Hello, world!" на языке Java

```
class HelloWorldApplication {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

и скомпилируем его, используя компилятор `javac`:

```
javac HelloWorldApplication.java
```

Результатом компиляции является файл `HelloWorldApplication.class`, содержащий J-код, выполнение которого на виртуальной Java-машине завершается выводом строки

```
Hello, World!
```

Как мы уже знаем, программа на языке Java строится из классов. Простая программа состоит из одного класса. Определение класса должно содержать две обязательные компоненты — ключевое слово `class` и следующее за ним имя класса. В примере это имя HelloWorldApplication. Оно должно совпадать с именем файла, содержащего текст программы. Следует иметь в виду, что в языке Java различаются большие и малые буквы. Компилятор следит за правильностью употребления имен: HelloWorld и Helloworld обозначают разные имена. Кроме обязательных компонентов определение может содержать модификаторы, характеризующие некоторые свойства класса. Вот список всех модификаторов в порядке их появления в определении.

public	Класс может быть использован классами из другого пакета; по умолчанию класс может использоваться только классами из одного с ним пакета
abstract	Абстрактный класс, для которого нельзя создать экземпляры; используется для последующего создания подклассов, дополняющих его структуру и поведение
final	Объявляет о том, что для данного класса нельзя создавать подклассы
class *class_name*	Обязательная часть определения класса
extends *Superclass*	Определяет, что для данного класса надклассом является *Superclass*
implements *Interface_list*	Определяет, что данный класс реализует список интерфейсов *Inteface_list*

Внутри фигурных скобок содержится описание переменных и методов данного класса. В классе `HelloWorldApplication` отсутствуют переменные и имеется всего один метод с именем `main()`. Каждое приложение Java обязательно должно содержать метод `main()`. При запуске приложения ему можно передать параметры через строковый массив `String[] args`. В рассмотренном примере метод `main()` состоит из одного обращения к методу `System.out.println()`, направляющему список своих аргументов в стандартный выходной поток.

Описания переменных и методов содержат обязательные компоненты и модификаторы. Обязательными являются имя, а также тип переменной или

тип значения, возвращаемого методом. Если метод не возвращает никакого значения, то в качестве описателя типа используется ключевое слово `void`.

Язык Java определяет несколько базовых типов для представления чисел, символов и булевских значений, и ссылочные типы, значениями которых являются ссылки на массив, класс или интерфейс. Имя класса само может использоваться в качестве идентификатора типа. Модификаторы представляют собой ключевые слова, задающие дополнительную информацию или накладывающие дополнительные ограничения на переменные и методы. Выше мы познакомились с модификаторами, которые используются в описании классов.

Интерфейсы

Интерфейс по своей структуре очень похож на абстрактный класс, в определении которого вместо ключевого слова `class` используется ключевое слово `interface`:

```
interface interface_name [extends interface_name] {
    [описание переменной;]
    [описание метода;]
}
```

Интерфейс может содержать описание переменных и заголовки описаний методов. Сами методы являются пустыми, их реализация в интерфейсе не определена. Интерфейс может наследовать свойства любого числа других интерфейсов, задаваемых при помощи модификатора `extends` *`interface_name`*. Если класс реализует некоторый интерфейс, то он должен явно определить все методы, перечисленные в описании интерфейса. Применению интерфейсов можно дать следующее объяснение. Класс может наследовать свойства (переменные и методы) только одного класса. Если родительский класс не содержит необходимых методов, то наследующий класс должен определить и реализовать их самостоятельно. Но лучше делать это, следуя определенной схеме. Эта схема, содержащая необходимые переменные и названия методов, предлагается интерфейсами. Класс же может реализовать любое число интерфейсов.

Например, интерфейс `java.awt.event.MouseListener` имеет следующее описание

```
public abstract interface MouseListener extends EventListener {
void      mouseClicked(MouseEvent e)
void      mouseEntered(MouseEvent e)
void      mouseExited(MouseEvent e)
void      mousePressed(MouseEvent e)
void      mouseReleased(MouseEvent e)
}
```

Класс, используемый в системе графического интерфейса пользователя и предполагающий обрабатывать события, связанные с манипуляциями мышью: нажатие и отпускание кнопки, щелчок, перемещение курсора мыши в пределы и за пределы объекта, должен реализовать интерфейс `java.awt.event.MouseListener`. Внутри класса для слежения за этими событиями при помощи метода `addMouseListener` создается специальный объект. Когда этот объект регистрирует одно из перечисленных событий, вызывается метод обработки этого события, указанный в интерфейсе `java.awt.event.MouseListener` и реализованный в самом классе.

Апплеты

После того как мы познакомились со структурой и рассмотрели простой пример обычного приложения, сделаем то же самое применительно к апплетам. С точки зрения структуры апплет отличается от приложения, прежде всего, отсутствием метода `main()`. Любой апплет представляет собой подкласс класса `Applet`. В иерархии пакетов и классов вновь создаваемый апплет `newApplet` занимает место согласно схеме, показанной на рис. 11.22.

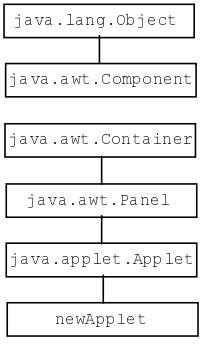


Рис. 11.22. Место апплета newApplet в иерархии классов

Каждый апплет, являясь подклассом класса `java.applet.Applet`, наследует все переменные и методы своих предшественников в данной иерархической структуре. В частности, класс `Applet` определяет следующие собственные методы.

Таблица 11.2. *Методы класса* `Applet`

Тип возвращаемого значения	Метод	Описание
`void`	`Destroy()`	Вызывается браузером, чтобы сообщить апплету о предстоящем завершении его работы с тем, чтобы он мог освободить используемые им ресурсы

Таблица 11.2 *(продолжение)*

Тип возвращаемого значения	Метод	Описание
`AppletContext`	`getAppletContext()`	Предоставляет возможность получать информацию о среде, в которой выполняется апплет и воздействовать на нее. Средой апплета является HTML-документ, содержащий этот апплет
`String`	`getAppletInfo()`	Возвращает информацию о самом апплете
`AudioClip`	`getAudioClip(URL url)`	Возвращает объект типа `AudioClip`, находящийся по адресу `url`
`AudioClip`	`getAudioClip(URL url, String name)`	Возвращает объект типа `AudioClip`, задаваемый параметрами `url` и `name`: `url` — базовый URL, задающий местоположение каталога `name` — местоположение аудиоклипа относительно каталога url
`URL`	`getCodeBase()`	Возвращает базовый URL апплета
`URL`	`GetDoc_umentBase()`	Возвращает базовый URL документа, в который встроен апплет
`Image`	`getImage(URL url)`	Возвращает графический объект типа `Image`, который затем может быть отображен на экране. Объект задается своим адресом URL
`Image`	`getImage(URL url, String name)`	Возвращает графический объект типа `Image`, который затем может быть отображен на экране. Объект задается своим базовым адресом `url` и относительным `name`
`Locale`	`getLocale()`	Возвращает объект типа `Locale`, содержащий информацию о национальных стандартах представления даты, времени, чисел и т. д., если таковые были установлены

Таблица 11.2 *(продолжение)*

Тип возвращаемого значения	Метод	Описание
String	getParameter(String name)	Возвращает значение именованного параметра name, определенного в HTML-тэге <param>
String[][]	getParameterInfo()	Возвращает строковый массив, содержащий информацию о параметрах. Обычно переопределяется в описании апплета
void	init()	Вызывается браузером для инициализации апплета: просмотра списка параметров, построения интерфейса пользователя, загрузки ресурсов
boolean	isActive()	Сообщает, является ли апплет активным
void	play(URL url)	Проигрывает аудиоклип, расположенный по указанному адресу url
void	play(URL url, String name)	Проигрывает аудиоклип, расположенный по указанному адресу, определяемому базовым адресом url и относительным name
void	resize(Dimension d)	Запрашивает новые размеры для апплета, задаваемые при помощи аргумента типа Dimension
void	resize(int width, int height)	Запрашивает новые размеры для апплета, задаваемые при помощи аргументов, определяющих новую ширину width и высоту height
void	setStub(AppletStub stub)	Вызывается браузером перед инициализацией апплета для создания интерфейса между апплетом и средой браузера
void	showStatus(String msg)	Запрашивает браузер отобразить в окне состояния строку msg
void	start()	Вызывается каждый раз, когда апплет попадает в область видимости

Таблица 11.2 *(окончание)*

Тип возвращаемого значения	Метод	Описание
void	stop()	Вызывается каждый раз, когда апплет исчезает из области видимости для приостановки его выполнения

Обсудим применение некоторых методов, оформив в виде апплета программу "Hello, World!". Создадим для этого подкласс класса `Applet` и сохраним его в файле `HelloWorld.java`. Простейший вариант может выглядеть следующим образом:

Пример 11.5. Текст апплета `HelloWorld.java`

```
import java.applet.Applet;
import java.awt.Graphics;
public class HelloWorld extends Applet {
    public void paint(Graphics g) {
       g.drawString("Hello world!", 5, 15);
    }
}
```

Первые две строки содержат оператор `import`. Он позволяет использовать внутри текущего класса сокращенные имена классов. Если бы его не было, то для ссылки на класс необходимо было указывать полное имя, определяющее положение класса в иерархии классов Java, например, `java.applet.Applet`.

Выводимая строка "Hello world" должна появиться в окне браузера, поэтому вместо записи ее в стандартный выходной поток используется метод `drawString()` класса `java.awt.Graphics`, используемый для отображения заданной строки в окне браузера. Пакет `java.awt` содержит полный набор классов для создания графического интерфейса пользователя и отображения графики и изображений.

Метод `paint()` является методом класса `java.awt.Container` и используется для изображения объекта, которому он принадлежит. В данном примере он переопределяется так, чтобы выводить изображение строки.

Для того чтобы просмотреть апплет в окне браузера, необходим HTML-документ, в который при помощи специального тэга вставлена ссылка на апплет. Создадим файл `HelloWorld.html`:

```
<HTML>
<HEAD><TITLE>Программа "Hello, world"</TITLE></HEAD>
```

```
<BODY>
<H3>Программа выводит строку:<H3>
<APPLET CODE="HelloWorld.class" WIDTH=150 HEIGHT=25>
</APPLET>
</BODY>
</HTML>
```

Откроем этот файл в окне браузера. Мы должны увидеть изображение, показанное на рис. 11.23.

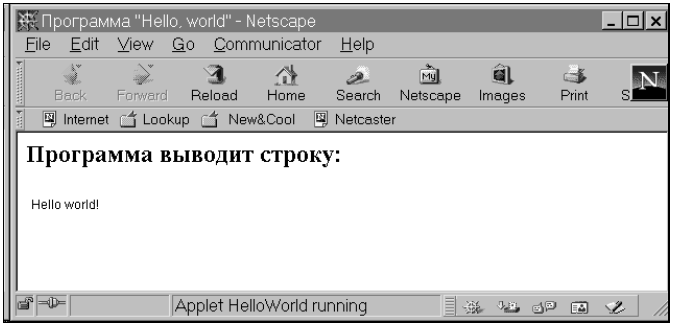


Рис. 11.23. HTML-документ апплета HelloWorld

Встраивание апплета в HTML-документ

Для встраивания апплетов в HTML-документ можно использовать два тэга: `<APPLET>` или `<OBJECT>`. Для каждого из них существует обязательный закрывающий тэг, соответственно `</APPLET>` и `</OBJECT>`. В спецификации языка HTML 4.0 тэг `<APPLET>` называется отмененным и рекомендуется использовать для встраивания апплетов более общий тэг `<OBJECT>`, который был описан ранее в этой главе. Вместе с тем, тэг `<APPLET>` продолжает широко использоваться. В версии языка HTML 4.0 он имеет следующий набор собственных параметров:

- `codebase=`*`url`*

  Задает базовый `URL` апплета. Если значение этого параметра не задано, то по умолчанию используется базовый `URL` текущего документа. Значением параметра может быть только URL-адрес каталога текущего документа или его подкаталогов.

- `archive=`*`url_list`*

  Задает список разделенных запятыми URL-адресов архивов. Архивы содержат классы и другие ресурсы, которые могут быть использованы апплетом. Они предварительно загружаются браузером для повышения производительности апплета. Адреса архивов задаются относительно базового URL апплета.

- `code=file_name`

  Определяет имя и местоположение файла, содержащего класс апплета, относительно базового URL апплета. Один из параметров `code` или `object` должен обязательно присутствовать.

- `object=file_name`

  Определяет имя и местоположение относительно базового URL апплета файла, содержащего описание апплета в специальном формате, содержащем только имена классов без их реализации. По этим именам в случае необходимости сами классы могут быть получены из архива.

- `alt=string`

  Указывается альтернативный текст для замены области `<APPLET>` в браузерах, которые не могут отображать апплеты.

- `name=string`

  Задает имя конкретного экземпляра апплета.

- `width=n`

  Ширина области апплета в пикселах.

- `height=n`

  Высота области апплета в пикселах.

- `align=(top|middle|bottom|left|right)`

  Задает выравнивание апплета в соответствии с заданным значением параметра.

- `hspace=n`

  Задает размер дополнительного пространства слева и справа от области апплета. Может задаваться в пикселах или процентах доступного свободного пространства.

- `vspace=n`

  Задает размер дополнительного пространства сверху и снизу от области апплета. Может задаваться в пикселах или процентах доступного свободного пространства.

Жизненный цикл апплета

Каждый апплет имеет свой цикл существования. То, что происходит с апплетом при открытии документа, просмотре, переходе к другому документу и возвращении, зависит от методов `init()`, `start()`, `stop()`, `destroy()`. Вызывая эти методы, браузер или программа просмотра апплетов appletviewer управляют поведением апплета.

Метод `init()` вызывается один раз сразу после загрузки документа и создания апплета. Он используется для того, чтобы подготовить условия для вы-

полнения апплета, например, создать интерфейс пользователя: кнопки, области ввода текста и другие элементы управления. Апплету могут понадобиться различные ресурсы, которые также удобно загрузить во время инициализации: шрифты, изображения и т. д.

Метод `start()` вызывается каждый раз, когда апплет становится видимым, метод `stop()`, напротив, когда апплет становится невидимым. Такая смена состояний происходит, например, при просмотре многостраничного документа или переходе по ссылке к другому документу и возвращении обратно. Метод `stop()` приостанавливает выполнение апплета, а метод `start()` возобновляет.

Метод `destroy()` также вызывается всего один раз и сообщает апплету о необходимости перед завершением работы освободить используемые ресурсы, кроме процессора и памяти.

Указанные методы могут переопределяться конкретным классом апплета. Рассмотренный пример `HelloWorld.java` все методы наследует, переопределяя только метод `paint()` для вывода изображения строки. Метод `paint()`, а также метод `update()`, наследуемые у класса `java.awt.Container`, используются для изображения на экране того объекта, к которому они относятся. Метод `paint()` полностью перерисовывает объект, а метод `update()` рисует только изменения в изображении объекта, экономя время, необходимое для восстановления картинки. Управление обновлением изображения организовано таким образом, что метод `update()` никогда не вызывается объектом непосредственно, а только через метод `repaint()`, как будет показано ниже. Методы `paint()` и `update()`, как правило, переопределяются внутри апплета.

От апплетов, подобных `HelloWord.java`, мало пользы. Отображаемая ими статическая информация может быть размещена в HTML-документе без дополнительных затрат на выполнение апплета. Следующий вариант апплета, также статический, добавляет к выводимому изображению некоторые элементы украшения.

Пример 11.6. Текст апплета `HelloWorld2.java`

```
import java.applet.*;
import java.awt.*;
public class HelloWorld2 extends Applet {
  private Font font;
  public void init() {
    font = new Font("Arial", Font.ITALIC, 48);
  }
  public void paint(Graphics g) {
    // The pink oval
    g.setColor(Color.blue);
```

```
    g.fillOval(10, 10, 330, 100);
    g.setColor(Color.yellow);
    g.setFont(font);
    g.drawString("Hello, World!", 40, 75);
  }
}
```

В этом примере уже переопределяется метод `init()`, осуществляющий загрузку необходимого ресурса — шрифта "Arial" для отображения выводимой строки.

Изображение апплета в окне браузера показано на рис. 11.24.

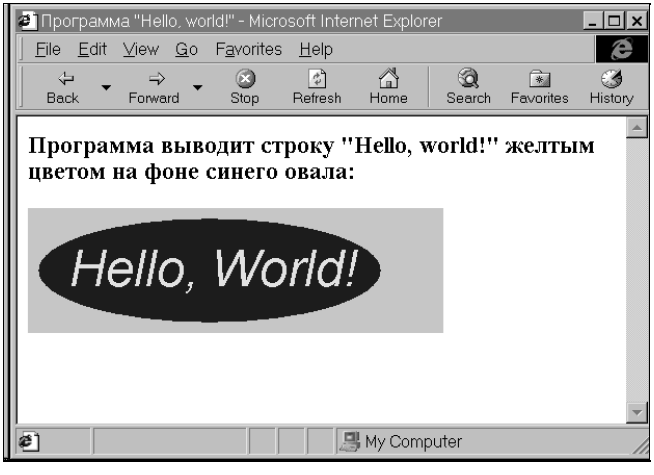


Рис. 11.24. HTML-документ апплета HelloWorld2

В следующем примере мы попытаемся переопределить методы, управляющие жизненным циклом апплета: `init()`, `start()`, `stop()`, `destroy()`, таким образом, чтобы каждое обращение к ним сопровождалось сообщением, выводимым на экран.

Пример 11.7. Текст апплета `HelloWorld3.java`

```
import java.applet.Applet;
import java.awt.Graphics;
public class HelloWorld3 extends Applet {
    StringBuffer buffer;
    int start=0;
    int stop=0;
    public void init() {
        buffer = new StringBuffer();
        addItem("Инициализация; ");
    }
```

```
    public void start() {
        addItem("Запуск апплета "+(++start)+"-й раз; ");
    }
    public void stop() {
        addItem("Приостановка апплета "+(++stop)+"-й раз; ");
    }
    public void destroy() {
        addItem("Заканчиваем работу");
    }
    void addItem(String newWord) {
        System.out.println(newWord);
        buffer.append(newWord);
        repaint();
    }
    public void paint(Graphics g) {
        g.drawRect(0, 0, getSize().width-1, getSize().height - 1);
        g.drawString("Hello, world!", 5, 15);
        g.drawString(buffer.toString(), 5, 30);
    }
}
```

В данном примере каждый из методов `init()`, `start()`, `stop()`, `destroy()` использует определенный внутри апплета метод `addItem()` для добавления к переменной `buffer` своего сообщения. Затем изображение строки `buffer`, помещенное после приветствия "Hello, world!" в прямоугольной рамке, при помощи метода `paint()` отображается на экране. Обратите внимание на тип переменной `buffer`. В языке Java для описания строкового типа используются классы `String` и `StringBuffer`. Класс `String` служит для представления строковых констант, а класс `StringBuffer` — строковых переменных.

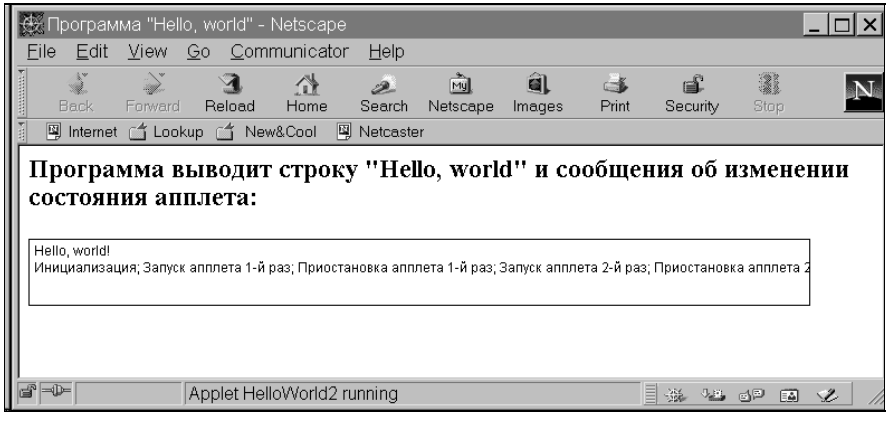


Рис. 11.25. HTML-документ апплета HelloWorld3

Выводимое на экран изображение будет динамически меняться с каждым изменением в жизненном цикле апплета и иметь вид, изображенный на рис. 11.25.

Создание графического интерфейса пользователя

Одним из недостатков последнего примера является невозможность просмотреть выводимую строку целиком. Видимая часть ограничена размером рамки. Для того чтобы иметь возможность просмотреть всю строку, необходимо применить соответствующий элемент графического интерфейса пользователя (GUI).

Для работы с окнами, графикой и создания пользовательского интерфейса в состав средств разработчика JDK входит набор AWT (Abstract Windowing Toolkit). Ему соответствует пакет java.awt, содержащий все необходимые классы. Набор AWT предоставляет для создания интерфейса пользователя такие компоненты:

Кнопка	(класс `java.awt.Button`).
Флажок	(класс `java.awt.Checkbox`).
Строка текста	(класс `java.awt.TextField`).
Многострочное текстовое поле	(класс `java.awt.TextArea`).
Метка	(класс `java.awt.Label`).
Список	(класс `java.awt.List`).
Всплывающее меню	(класс `java.awt.Choice`).
Полоса прокрутки	(класс `java.awt.Scrollbar`).
Графическая область экрана	(класс `java.awt.Canvas`).
Меню	(класс `java.awt.Menu`).

Начиная с версии 1.1, набор JDK содержит новый комплект средств разработки графического интерфейса пользователя — `Swing`. Комплект `Swing` содержит многие элементы, отсутствующие в комплекте `AWT`, но не является его полной заменой. `Swing` позволяет создавать элементы графического интерфейса, которые выглядят одинаково на различных платформах. Как известно, в графической оконной системе UNIX внешний вид элементов интерфейса зависит от выбора соответствующей библиотеки, содержащей набор этих элементов. Используя `Swing`, можно создавать графический интерфейс, внешне одинаковый для разных платформ. В Java 1.2 пользовательский интерфейс рекомендуется создавать, используя `Swing` вместо `AWT`.

В следующем примере для иллюстрации используется всего один элемент GUI — текстовое поле, которое может содержать несколько одновременно

отображаемых строк текста с возможностью просмотра в горизонтальном и вертикальном направлении.

Пример 11.8. Текст апплета `HelloWorld4.java`

```
import java.applet.Applet;
import java.awt.TextArea;
public class HelloWorld4 extends Applet {
    TextArea text_area;
    int start=0;
    int stop=0;
    public void init() {
       text_area = new TextArea("Hello, World!");
       text_area.setEditable(false);
       setLayout(new java.awt.GridLayout(1,0));
       add(text_area);
       addItem("Инициализация;");
    }
    public void start() {
       addItem("Запуск апплета "+(++start)+"-й раз; ");
    }
    public void stop() {
       addItem("Приостановка апплета "+(++stop)+"-й раз; ");
    }
    public void destroy() {
       addItem("Заканчиваем работу");
    }
    void addItem(String newWord) {
       text_area.append(newWord);
    }
}
```

Изображение апплета представлено на рис. 11.26.

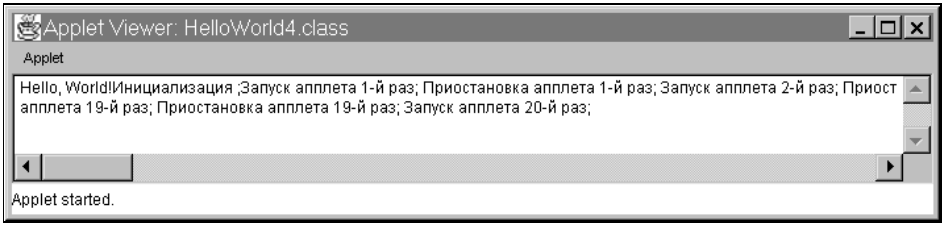


Рис. 11.26. HTML-документ апплета HelloWorld4

Последний пример отличается от предыдущего только способом вывода на экран текста. В примере 11.8 для вывода используется элемент графического

интерфейса — текстовая область. Это изменение находит отражение в тексте апплета. В последнем примере отсутствует переопределение метода `paint()`. В нем нет необходимости, так выводом на экран теперь управляет не метод апплета, а сам элемент интерфейса. Посмотрим, в каком месте программы это происходит.

Основные изменения произошли в методе `init()`. Сам апплет содержит новую переменную `text_area`, которая может принимать значение ссылки на объект типа `TextArea`. Этот тип представляет область, способную отображать несколько строк текста. Метод `init()` в качестве первого шага инициализации апплета создает при помощи оператора `new` новый объект типа `TextArea` и выводит в него начальную строку `"Hello, World!"`.

Вызов метода `setEditable()` с аргументом `false` запрещает редактирование текста внутри области.

Следующие две строки содержат обращения к методам `setLayout()` и `add()`. Эти методы класс `Applet` наследует у класса `java.awt.Container`.

Метод `add()` служит для того, чтобы вставить в апплет элемент, задаваемый аргументом, в данном случае — текстовую область, на которую указывает переменная `text_area`.

Метод `setLayLayout()` создает новый объект типа `GridLayout` для управления размещением элементов внутри апплета.

Последнее замечание нуждается в пояснении. Существуют разные формы метода `add()`, отличающиеся количеством и типом аргументов. Форма, использованная в примере, не имеет дополнительных аргументов, при помощи которых можно было бы указать, каким образом следует расположить добавляемый элемент внутри апплета. А что произойдет, если кроме текстовой области добавить в апплет другие элементы: кнопки, метки, меню? Для управления размещением элементов внутри объектов-контейнеров (к ним относятся и апплеты) в языке Java существует специальный тип объектов `LayoutManager`. Если говорить точнее, то `java.awt.LayoutManager` — это интерфейс, содержащий список методов, без их реализации, для управления компоновкой элементов внутри контейнера. Реализация методов интерфейса, как мы знаем, осуществляется внутри классов. В состав AWT входят несколько классов, реализующих некоторые общие схемы размещения. Это классы `BorderLayout`, `CardLayout`, `FlowLayout`, `GridLayout`, `GridBagLayout`. Внутри апплета создается объект одного из перечисленных типов, называемый менеджером размещения, который и управляет размещением элементов. В примере 11.8 используется менеджер типа `GridLayout`.

Менеджер `BorderLayout` старается разместить элемент в одной из пяти именованных областей контейнера: `NORTH` (СЕВЕР), `SOUTH` (ЮГ), `EAST` (ВОСТОК), `WEST` (ЗАПАД) и `CENTER` (ЦЕНТР), в соответствии со значением аргумента в методе `add()`, например:

```
add(new Button("OK"), BorderLayout.SOUTH);
```

Менеджер `CardLayout` размещает элементы подобно карточкам в картотеке, когда только одна карточка является видимой в каждый момент.

Менеджер `FlowLayout` размещает элементы слева направо и сверху вниз подобно тому, как размещается текст на странице.

Менеджер `GridLayout` делит область контейнера на равные прямоугольники и размещает элементы по одному в каждом прямоугольнике. Например, обращение к методу

```
setLayout(new GridLayout(3,2));
```

устанавливает для размещения элементов сетку, состоящую из трех строк и двух столбцов.

Менеджер `GridBagLayout` использует наиболее гибкую схему, выравнивая по горизонтали и по вертикали элементы, которые могут быть разного размера.

Обработка событий

Среди методов, которые класс `Applet` наследует от своих предшественников в иерархии классов, есть методы, предназначенные для обработки различных событий, связанных с мышью и клавиатурой. В предыдущих примерах апплеты не реагировали на такие события. Внесем добавления в текст последнего примера так, чтобы апплет воспринимал нажатие клавиши и щелчок мышью на области, занимаемой апплетом, и реагировал на эти события выводом сообщения.

Пример 11.9. Текст апплета `HelloWorld5.java`

```
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.awt.event.KeyListener;
import java.awt.event.KeyEvent;
import java.applet.Applet;
import java.awt.TextArea;

public class HelloWorld5 extends Applet implements MouseListener,
KeyListener {
    TextArea text_area;
    int start = 0;
    int stop  = 0;
    int click = 0;
    int keytyped = 0;

       public void init() {
              addMouseListener(this);
              addKeyListener(this);
```

```
            text_area = new TextArea("Hello, World!",4,80);
            text_area.setEditable(false);
            setLayout(new java.awt.FlowLayout());
            add(text_area);
            addItem("Инициализация;");
        }
        public void start() {
            addItem("Запуск апплета "+(++start)+"-й раз; ");
        }
        public void stop() {
            addItem("Приостановка апплета "+(++stop)+"-й раз; ");
        }
        public void destroy() {
            addItem("Заканчиваем работу");
        }
        void addItem(String newWord) {
            text_area.append(newWord);
        }
        public void mouseEntered(MouseEvent event) {
        }
        public void mouseExited(MouseEvent event) {
        }
        public void mousePressed(MouseEvent event) {
        }
        public void mouseReleased(MouseEvent event) {
        }
        public void mouseClicked(MouseEvent event) {
            addItem((++click)+"-й "+"щелчок;");
        }
        public void keyPressed(KeyEvent event) {
        }
        public void keyReleased(KeyEvent event) {
        }
        public void keyTyped(KeyEvent event) {
            addItem((++keytyped)+"-ое "+" нажатие клавиши; ");
        }
}
```

Прежде всего, мы замечаем, что изменилась первая строка описания класса. Она содержит предложение `implements MouseListener, KeyListener`. Это означает, как мы помним, что класс `HelloWorld5` реализует два интерфейса: `java.awt.event.MouseListener` и `java.awt.event.KeyListener`, и, значит, обязан реализовать методы этих интерфейсов. Выше мы перечислили методы интерфейса `java.awt.event.MouseListener`. В классе `HelloWorld5` они реализованы очень просто. Метод `mouseClicked()`, отвечающий за обработку щелч-

ков мышью, вызывает метод `addItem()` для добавления в текстовую область сообщения об этом событии. Остальные методы свои события игнорируют. Интерфейс `java.awt.event.KeyListener` объявляет три метода:

`void keyPressed(KeyEvent e)`	Вызывается при нажатии клавиши
`void keyReleased(KeyEvent e)`	Вызывается при отпускании клавиши
`void keyTyped(KeyEvent e)`	Вызывается при отпускании клавиши после нажатия

В классе `HelloWorld5` все они также реализованы, но обрабатывается только отпускание клавиши после нажатия.

Если запустить апплет, то он будет реагировать на нажатие клавиш и щелчки мыши в своей области выводом сообщений, как показано на рис. 11.27.

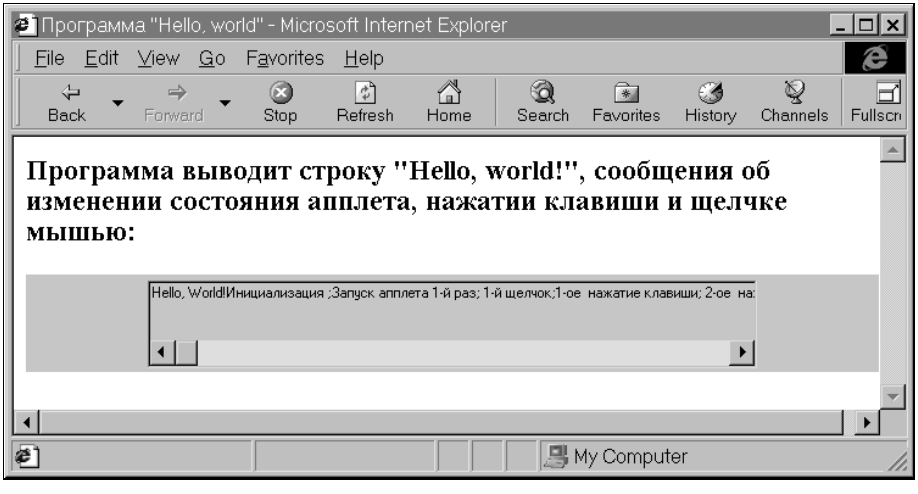


Рис. 11.27. HTML-документ апплета HelloWorld5

В языке Java каждому типу событий соответствует класс, например, событиям, связанным с мышью, — класс `MouseEvent`, событиям, связанным с клавиатурой, — класс `KeyEvent`, событиям, которые генерируются при манипуляциях с компонентами пользовательского интерфейса (нажатие кнопки), — класс `ActionEvent`. Каждое событие имеет источник, представленный объектом, и идентификатор типа, позволяющий различать разные события одного класса, например, нажатие и отпускание клавиши. Модель обработки событий основана на применении специальных объектов — регистраторов событий. Объект-источник поддерживает список всех объектов-регистраторов. При возникновении события он вызывает метод объекта-регистратора и передает ему в качестве аргумента объект, соответствующий типу события. Чтобы источник события мог вызвать метод объекта-регистратора, необходимо, чтобы этот метод был регистратором реализован. С этой целью все

объекты-регистраторы одного типа должны реализовывать соответствующий интерфейс. Например, если объект регистрирует события клавиатуры `KeyEvent`, то он должен реализовать интерфейс `KeyListener`.

В нашем примере в качестве регистратора событий клавиатуры и мыши используется сам апплет HelloWord5. Он добавляет себя в списки регистраторов к объектам-источникам событий при помощи методов `addKeyListener()` и `addMouseListener()`, соответственно. Идентификатор `this` обозначает текущий апплет. При щелчке мышью в области апплета объект-источник события вызовет метод `mouseClicked(MouseEvent event)`, реализованный в апплете. Аналогично будут обрабатываться и другие события.

В примере 11.9 апплет обрабатывает не все события клавиатуры и мыши, но должен определять все методы соответствующего интерфейса, даже если он их не использует. В результате текст апплета содержит шесть пустых методов или "заглушек": `mouseEntered`, `mouseExited`, `mousePressed`, `mouseReleased`, `keyPressed` и `keyReleased`. Если апплет реализует несколько интерфейсов, и из каждого интерфейса использует только по одному методу, то он все равно должен реализовать оставшиеся методы как "заглушки". Избежать этого можно, только отказавшись от реализации интерфейсов. Но как тогда обрабатывать события? В этом случае можно использовать возможности, добавленные в версию Java 1.1.

В Java 1.0 не разрешалось определять новые классы внутри других классов. Новый класс мог быть определен только на верхнем уровне — как член пакета. Расширения языка, добавленные в Java 1.1, позволяют определять новые классы внутри других классов и даже внутри блоков, заключенных в фигурные скобки {}. Такие классы называют внутренними классами. Используя внутренние классы, в предыдущем примере можно отказаться от интерфейсов `KeyListener`, `MouseListener` и записать текст более компактно:

Пример 11.10. Текст апплета `HelloWord5a.java`

```
import java.awt.event.*;
import java.applet.*;
import java.awt.*;

public class HelloWorld5a extends Applet {
       TextArea text_area;
       int start = 0;
       int stop  = 0;
       int click = 0;
int keytyped = 0;

       public void init() {
              this.addMouseListener(new MouseAdapter() {
                  public void mouseClicked(MouseEvent event) {
```

```
                    addItem((++click)+"-й "+"щелчок;");
                    }
                });
            this.addKeyListener(new KeyAdapter() {
                public void keyTyped(KeyEvent event) {
                    addItem((++keytyped)+"-ое "+" нажатие клавиши; ");
                    }
                });
            text_area = new TextArea("Hello, World!",4,80);
            text_area.setEditable(false);
            setLayout(new java.awt.FlowLayout());
            add(text_area);
            addItem("Инициализация;");
        }
        public void start() {
            addItem("Запуск апплета "+(++start)+"-й раз; ");
        }
        public void stop() {
            addItem("Приостановка апплета "+(++stop)+"-й раз; ");
        }
        public void destroy() {
            addItem("Заканчиваем работу");
        }
        void addItem(String newWord) {
            text_area.append(newWord);
        }
}
```

Разберем, как обрабатываются события в этом апплете. В методе `init()` апплет передает своему методу `addMouseListener()` аргумент, чтобы добавить его в список объектов-регистраторов событий от мыши. Аргументом является объект, создаваемый оператором `new`. Рассмотрим его подробно.

`MouseAdapter` — это абстрактный класс, имеющий следующее описание:

```
public abstract class MouseAdapter extends Object implements MouseListener {
void   mouseClicked(MouseEvent event);
void   mouseEntered(MouseEvent event);
void   mouseExited(MouseEvent event);
void   mousePressed(MouseEvent event);
void   mouseReleased(MouseEvent event);
}
```

Все методы класса `MouseAdapter` — это методы интерфейса `MouseListener`. Они также являются пустыми. Этот абстрактный класс является шаблоном, используя который можно создать новый класс, в котором переопределить

пустые методы так, чтобы они обрабатывали события нужным образом. Если затем создать объект этого нового класса, то его можно использовать в качестве регистратора событий мыши, передав в виде аргумента методу `addMouseListener()`. Оператор

```
new MouseAdapter() {
        public void mouseClicked(MouseEvent event) {
                addItem((++click)+"-й "+"щелчок;");
        }
}
```

одновременно определяет новый внутренний безымянный класс и создает объект этого класса. Данный внутренний класс наследует классу `MouseAdapter`, переопределяя только один, необходимый апплету, метод `mouseClicked()`. Вновь созданный объект безымянного класса в качестве аргумента `addMouseListener()` добавляется в список регистраторов событий мыши.

Обработка событий клавиатуры организована аналогично.

Рисунки в апплетах

Наличие хорошо подобранных рисунков оживляет изображение HTML-страницы, делает ее более выразительной. Рисунок можно вставить непосредственно в HTML-документ при помощи тэга `<img>` или отобразить в составе апплета. Второй способ является более гибким и мощным, так как апплет кроме простого статического отображения рисунка обладает и многими другими возможностями, например, способностью динамически отображать и удалять рисунки, используя кнопки и другие элементы графического интерфейса, или создавать эффект анимации. Апплет располагает также методами для проигрывания звуковых файлов. Добавление в апплет изображений и звуков осуществляется однотипно. Рассмотрим простой пример апплета, отображающего статический рисунок.

Пример 11.11. Текст апплета `HelloWorld6.java`

```
import java.awt.*;
import java.applet.*;
import java.net.*;
public class HelloWorld6 extends Applet {
        Image globe;
        public void init() {
                URL codeBase = getCodeBase();
                globe = getImage(codeBase, "globe.gif");
        }
        public void paint(Graphics g){
                int width = globe.getWidth(this);
```

```
                int height = globe.getHeight(this);
                g.drawRect(52, 52, width+10, height+10);
                g.drawImage(globe, 57, 57, width, height, this);
        }
}
```

В примере 11.11 предполагается, что файл `globe.gif`, содержащий изображение, находится в том же каталоге, что и файл, содержащий код апплета. При помощи метода `getCodeBase()` определяется URL-адрес этого каталога, после чего метод `getImage()` загружает из каталога указанный файл. Для отображения файла в окне браузера переопределяется метод `paint()`. Изображение апплета приведено на рис. 11.28.

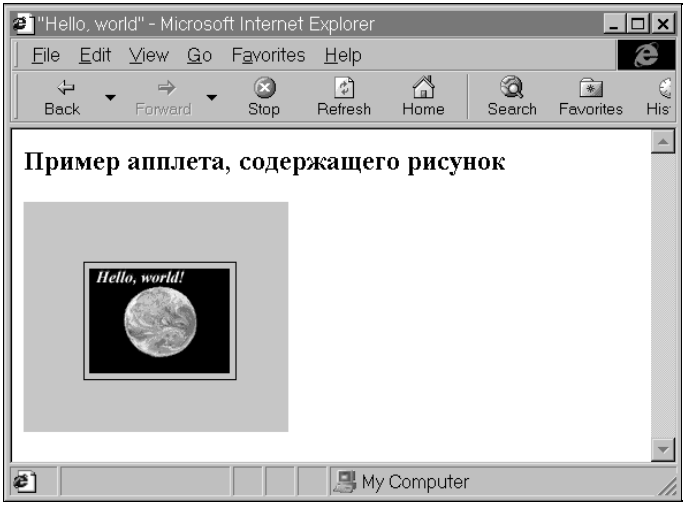


Рис. 11.28. Пример апплета, содержащего рисунок

Архивы

В предыдущем примере для нормальной работы апплета требуются два файла: файл апплета `HelloWorld6.class` и файл рисунка `globe.gif`. Апплет может содержать несколько рисунков, несколько звуковых файлов и использовать внутренние классы, о которых говорилось выше. Для внутренних классов компилятор создает отдельные файлы. Таким образом, для работы апплета может потребоваться достаточно большое количество файлов. Браузер при загрузке апплета должен загрузить также и все необходимые файлы, и для каждого из них он открывает отдельное HTML-соединение с сервером. В результате увеличивается сетевой трафик и загруженность Web-сервера, который вместо одного запроса вынужден обрабатывать несколько. Выход заключается в том, чтобы поместить все необходимые файлы в один архивный файл. Передача этого файла браузеру произойдет в рамках одного

HTML-соединения. Впервые передача апплетов в составе архивных файлов была поддержана браузером Netscape, который использовал для этой цели формат ZIP без компрессии. Для того чтобы сообщить браузеру о местонахождении архива, в состав тэга `<APPLET>`был добавлен новый параметр `ARCHIVE`:

```
<APPLET CODE = "myApplet.class" ARCHIVE = "myApplet.zip"> </APPLET>
```

Местоположение архива указывается относительно базового URL-адреса апплета, то есть относительно каталога, содержащего его код. Версия HTML 4.0 официально поддерживает параметр `ARCHIVE`.

Впоследствии фирмы Netscape и Sun совместно разработали на базе ZIP-формата новый формат JAR (Java ARchive), позволяющий передавать апплеты вместе с сопутствующими файлами в составе одного архивного файла в сжатом виде. Формат JAR поддерживается также последними версиями Internet Explorer (4.0+). Для создания JAR-архива в состав JDK входит утилита командной строки `jar`. Типичное обращение к ней имеет вид:

```
jar   -cf   myarchive.jar   *.class   image.gif   sound.au
```

Значение опций:

`-c` — создать новый архив;

`-f` — указать имя архивного файла.

Java и JavaScript

Применения апплетов в Web-программировании многочисленны и разнообразны. Их подробное изучение не является целью данной книги. Мы рассмотрели основы апплетов и посвятим остаток раздела тому, как организовать совместное использование двух разных технологий, затронутых в этой книге: сценариев JavaScript и Java-апплетов.

Для создания динамических HTML-страниц используются различные подходы. С некоторыми из них мы уже познакомились. Язык JavaScript создан в фирме Netscape и встроен разработчиками в браузер Netscape для создания интерактивных сценариев. Технология Java развивается фирмой Sun. Каждая из этих технологий имеет свои преимущества и недостатки, но обе располагают большим арсеналом средств для решения своих задач. Некоторые задачи удобнее решать при помощи JavaScript, другие — используя Java. Браузер Netscape поддерживает обе технологии, но первоначально они существовали внутри браузера изолированно друг от друга, не имея возможности взаимодействовать. Возможность такого взаимодействия полезна, так как позволяет некоторые задачи решать проще, комбинируя подходящие приемы из обеих технологий. Начиная с версии 3.0 в браузеры Netscape добавлена новая технология LiveConnect, поддерживающая взаимодействие JavaScript и Java, а именно:

- сценарии JavaScript могут взаимодействовать со стандартными классами Java, встроенными в браузер;
- сценарии JavaScript могут читать и изменять данные апплетов, в описании которых имеется ключевое слово `public`;
- сценарии JavaScript могут вызывать методы апплетов, в описании которых имеется ключевое слово `public`;
- апплеты могут читать и изменять свойства объектов, элементы массивов и вызывать функции JavaScript.

Для взаимодействия с Java технология LiveConnect добавляет в язык JavaScript новые типы данных.

- `JavaPackage` — объект, являющийся представлением пакета Java в JavaScript
- `JavaClass` — объект, являющийся представлением класса Java в JavaScript
- `JavaObject` — объект, являющийся представлением объекта Java в JavaScript
- `JavaArray` — объект, являющийся представлением массива Java в JavaScript

Объект `JavaClass` не имеет собственных свойств. Все его свойства представляют поля и методы соответствующего класса Java, имеющие ключевое слово `public` в своем описании, и имеют те же самые имена. Например, объект `java.lang.System` представляет Java-класс `java.lang.System`.

Объект `JavaObject` представляет объект Java, то есть экземпляр Java-класса. Его свойства представляют открытые поля и методы экземпляра класса.

Объект `JavaPackage` может содержать классы, представленные объектами `JavaClass`. Все объекты `JavaPackage` содержатся в родительском объекте. На этот объект самого верхнего уровня иерархии ссылается свойство `Packages` объекта `Window`. Схема именования свойств для JavaPackage отражает схему именования пакетов Java. Например, свойство `java.lang.System` представляет класс `java.lang.System`.

Браузер Netscape, поддерживая технологию Java, содержит собственную коллекцию Java-классов. В эту коллекцию для взаимодействия с JavaScript добавлены новые классы, которые сгруппированы в самостоятельную иерархию пакетов, схема именования которой содержит имена, начинающиеся словом `netscape`. Классы, обеспечивающие взаимодействие Java с JavaScript, содержатся в пакете `netscape.javascript`. Основным среди них является класс `netscape.javascript.JSObject`, представляющий объект JavaScript.

Примечание

Браузер MS Internet Explorer также имеет свою коллекцию классов Java. Начиная с версии Internet Explorer 4.0 в нее включен пакет `netscape.javascript`.

Класс `JSObject` имеет следующий синтаксис:

```
public final class JSObject extends Object {
       public static JSObject getWindow(java.applet.Applet applet);
       public Object getMember(String name);
       public Object getSlot(int index);
       public void setMember(String name, Object value);
       public void setSlot(int index, Object value);
       public void removeMember(String name);
       public Object call(String methodName, Object args[]);
       public Object eval(String s);
       public String toString();
       protected void finalize();
}
```

- Метод `getWindow()` возвращает объект типа `JSObject`, представляющий окно браузера, содержащее апплет *applet*.
- Метод `getMember(name)` возвращает объект Java, представляющий свойство объекта JavaScript, заданное аргументом *name*.
- Метод `getSlot()` возвращает элемент массива JavaScript.
- Метод `setMember(name, value)` устанавливает для объекта JavaScript значение свойства *name* равным *value*.
- Метод `setSlot(index, value)` устанавливает значение элемента массива.
- Метод `removeMember(name)` удаляет свойство *name* объекта JavaScript.
- Метод `call(methodName, args[])` вызывает метод *methodName* объекта JavaScript и передает ему массив аргументов *args[]*.
- Метод `eval(s)` выполняет строку *s*, которая содержит интерпретируемый код JavaScript.
- Метод `toString()` служит для преобразования различных значений в строковую форму.

Все объекты JavaScript образуют иерархическую структуру, имеющую в качестве корня текущее окно браузера. Для доступа к объектам JavaScript нужно сначала получить доступ к этому окну.

Необходимо выполнить несколько условий, чтобы предоставить апплетам Java доступ к свойствам и методам объектов JavaScript.

- Первое, что нужно сделать, — это обеспечить компиляцию самого апплета. Для этого надо сообщить компилятору, где находятся классы, входящие в пакет `netscape.javascript`. Браузеры сохраняют этот пакет в составе архивных файлов: `java40.jar` (Netscape 4.x) и `classes.zip` (Internet Explorer 4.0 и выше). Надо уточнить местоположение этих файлов и указать его в переменной среды `CLASSPATH`, например:

```
set CLASSPATH= .;c:\winnt\Java\classes.zip;
c:\Program Files\Netscape\Communicator\Program\Java\Classes\java40.jar;
```

В Windows это можно сделать также через Панель управления.

- В текст апплета добавить строку

```
import netscape.javascript.*
```

- Для того чтобы апплет имел доступ к объектам `JSObject`, необходимо в HTML-документе добавить в тэг `<APPLET>` параметр `MAYSCRIPT`.
- Внутри апплета создать для доступа к объектам JavaScript объект, представляющий окно браузера, содержащее данный апплет:

```
JSObject applet_win = JSObject.getWindow(this);
```

Рассмотрим несколько примеров взаимодействия Java с JavaScript.

Пример 11.12. Вызов функции JavaScript из апплета

а) Текст апплета `AppletCallsJScript.class`

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import netscape.javascript.*;
public class AppletCallsJScript extends Applet implements ActionListener {
       Button b;
       TextField t;
       public void init() {
              t = new TextField(40);
              add(t);
              b = new Button("Выполнить");
              add(b);
              b.addActionListener(this);
       }
       public void actionPerformed(ActionEvent ae) {
              if (ae.getSource() == b) {
                 JSObject mywin = JSObject.getWindow(this);
                 mywin.eval(t.getText());
              }
       }
}
```

б) Текст HTML-документа

```
<HTML><HEAD></HEAD><BODY>
<h3>
Здесь размещен апплет, вызывающий выполнение функции JavaScript
</h3>
```

```
Введите функцию и нажмите кнопку<br>
<APPLET CODE=AppletCallsJScript.class
        MAYSCRIPT
        WIDTH=300
        HEIGHT=80>
</APPLET></BODY></HTML>
```

В этом примере апплет, встроенный в HTML-документ, содержит текстовое поле и кнопку **Выполнить**. В текстовое поле можно ввести строку, которая содержит обращение к функции JavaScript. Нажатие кнопки обрабатывается методом `actionPerformed()`. Метод `actionPerformed()` создает объект, представляющий окно браузера, вызывает метод этого объекта `eval()` и передает ему для выполнения строку, введенную в текстовое поле. На рис. 11.29 показана работа апплета, который вызывает функцию `alert('HELLO, WORLD')`, осуществляющую вывод сообщения в диалоговое окно.

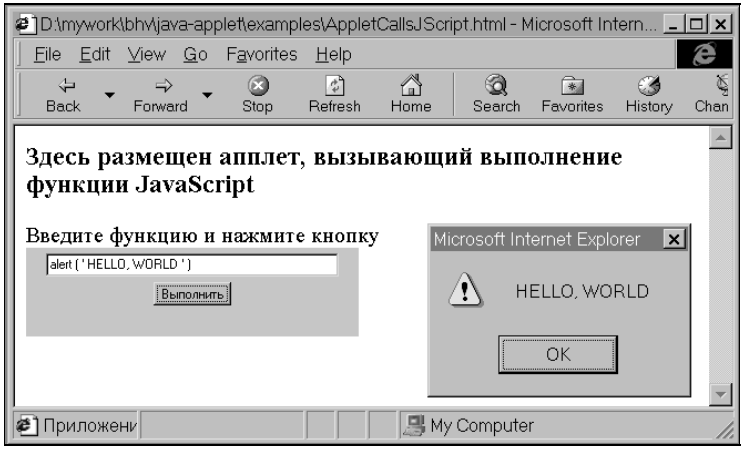


Рис. 11.29. Вызов функции JavaScript из апплета Java

Пример 11.13. Динамическое создание HTML-документа

а) Текст апплета `CreateHtmlForm.java`

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import netscape.javascript.*;
public class CreateHtmlForm extends Applet implements ActionListener {
   Button aButton;
   public void init(){
     setLayout(new FlowLayout());
```

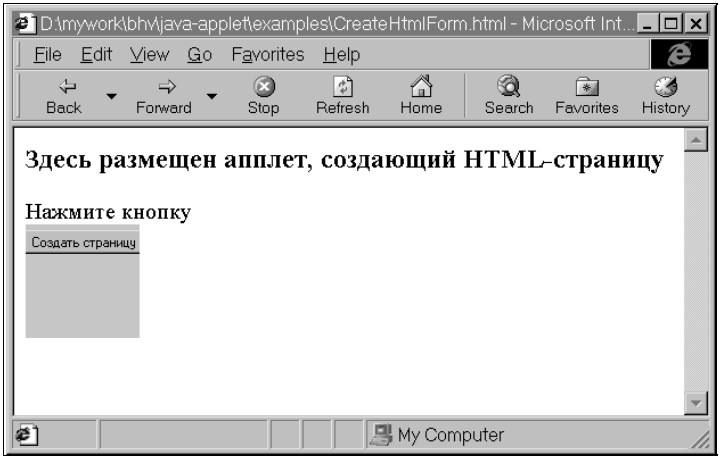
```
      aButton = new Button("Создать страницу");
      add(aButton);
      aButton.addActionListener(this);
      }
   public void actionPerformed(ActionEvent ae) {
      if (ae.getSource() == aButton){
        String HTML = "<html>";
        HTML +="<head><title>Создание страницы</title></head>";
        HTML +="<body>";
        HTML +="<h3>Эта страница создана из апплета Java</h3>";
        HTML +="<form method='get' action='/cgi-bin/first.cgi'>";
        HTML +="Ввод строки<input type='text' name='textfield'><br><br>";
        HTML +="Кнопка передачи <input type=submit
name=subname><br><br>";
        HTML +="Кнопка сброса <input type=reset></p>";
        HTML +="</form></body></html>";
        JSObject win = JSObject.getWindow(this);
        win.eval("createHTML(\"" + HTML +"\");");
      }
   }
}
```

б) Текст HTML-документа со встроенным сценарием JavaScript

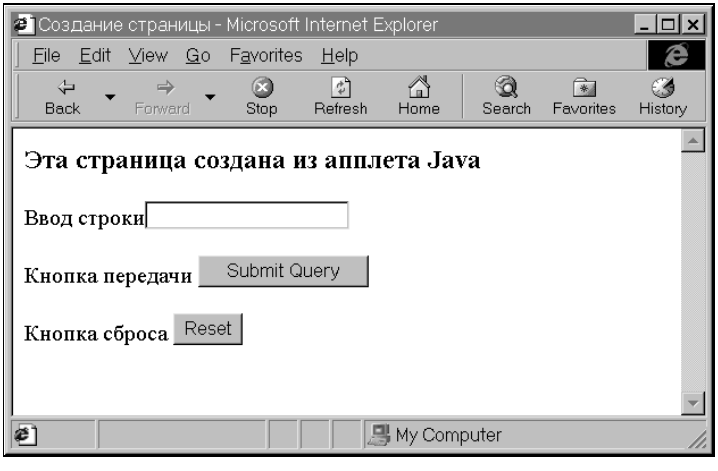
```
<HTML><HEAD></HEAD><BODY>
<SCRIPT>
function createHTML(s) {
    document.write(s);
    document.close();
    }
</SCRIPT>
<h3>
Здесь размещен апплет, создающий HTML-страницу
</h3>
Нажмите кнопку<br>
<APPLET CODE=CreateHtmlForm.class
         MAYSCRIPT
         WIDTH=120
         HEIGHT=100>
</APPLET></BODY></HTML>
```

В примере 11.13 (б) HTML-документ содержит описание функции JavaScript с именем `createHTML()`. Эта функция вызывает метод `write(s)` объекта `document` для записи в объект `document` строки `s`. Объект `document` представляет HTML-документ, выведенный на экран в данный момент. Если выводимые строки являются строками HTML-кода, то таким образом создается новый HTML-документ. В HTML-документ встроен апплет, содержащий

одну кнопку с надписью **Создать страницу**. При нажатии кнопки вызывается метод `ActionPerfomed()`, обрабатывающий это событие. Обработка нажатия кнопки заключается в том, что формируется новый текст HTML-документа, который передается JavaScript-функции `createHTML()` для записи в документ. Вид исходного документа в окне браузера (*а*) и результат выполнения апплета (*б*) представлены на рис. 11.30.



а



б

Рис 11.30. Динамическое создание HTML-страницы

Пример 11.14. Обмен данными между апплетом и полем HTML-формы

а) Текст апплета `FormReadWrite.java`

```
import java.applet.*;
import java.awt.event.*;
```

```
import java.awt.*;
import netscape.javascript.*;
public class FormReadWrite extends Applet implements ActionListener {
       Button button1,button2;
       TextField text_field;
       JSObject window;
       public void init(){
              setLayout(new FlowLayout());
              text_field = new TextField(30);
              button1 = new Button("Запись в форму");
              button2 = new Button("Считывание из формы");
              button1.addActionListener(this);
              button2.addActionListener(this);
              add(text_field);
              add(button1);
              add(button2);
       }
       public void actionPerformed(ActionEvent ae) {
              if (ae.getSource() == button1) {
                     JSObject window = JSObject.getWindow(this);
              window.eval("setHTMLText('"+text_field.getText()+"');");
              }
              if (ae.getSource() == button2) {
                     JSObject window = JSObject.getWindow(this);
              text_field.setText((String)window.eval("getHTMLText();"));
              }
       }
}
```

б) Текст HTML-документа

```
<HTML><HEAD><SCRIPT>
 function getHTMLText(){
   return document.forms[0].elements[0].value;
   }
 function setHTMLText(s){
   document.forms[0].elements[0].value = s;
   }
 </SCRIPT></HEAD><BODY>
Это текстовое поле формы<br>
 <FORM>
   <INPUT TYPE=text SIZE=30>
 </FORM>
А это апплет с текстовым полем и двумя кнопками<br>
<APPLET ALIGN=TOP CODE=FormReadWrite.class MAYSCRIPT WIDTH=300 HEIGTH=60>
</APPLET>
</BODY></HMTL>
```

HTML-документ данного примера содержит форму, состоящую из одного текстового поля, и апплет, содержащий текстовое поле и две кнопки **Запись в форму** и **Считывание из формы**. В документе определены также две функции JavaScript: `getHTMLText()` и `setHTMLText()`. Функция `getHTML()` возвращает, а функция `setHTMLText()` устанавливает значение первого поля первой на данной странице формы. Так как на данной странице всего одна форма, которая содержит только одно текстовое поле, то операции чтения и записи применяются именно к нему. Функции JavaScript вызываются внутри апплета методом `actionPerformed()`, обрабатывающим нажатие кнопок. Нажатие кнопки **Считывание из формы** вызывает копирование данных из текстового поля формы в текстовое поле апплета, нажатие кнопки **Запись в форму** — наоборот, копирование данных из текстового поля апплета в текстовое поле формы. Окно браузера, отображающее документ, представлено на рис. 11.31.

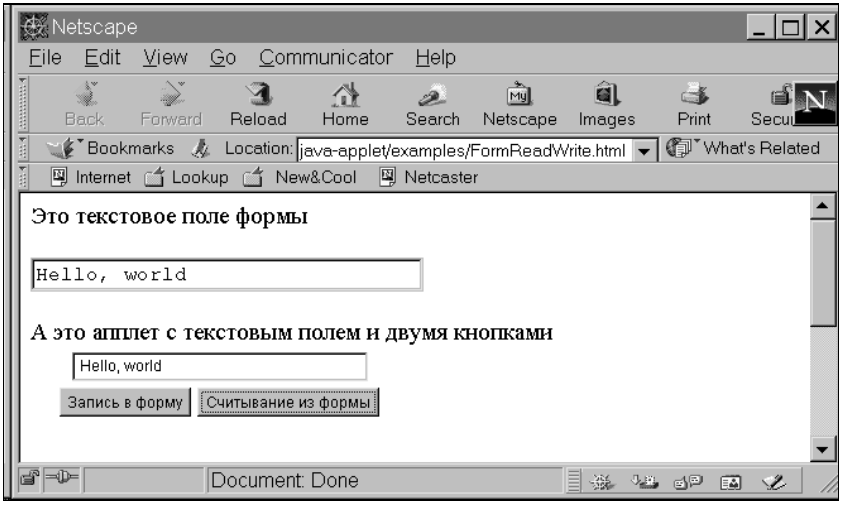


Рис 11.31. Обмен данными между апплетом и полем HTML-формы

Заключение

Данный раздел является элементарным введением в апплеты, предназначенным для того, чтобы дать начальное представление об этой развивающейся области Web-программирования. Возможно он, как и другие разделы этой книги, вызовет интерес к продолжению изучения затронутой темы. Многие вопросы, связанные с созданием и применением апплетов, остались за пределами изложения. Для их изучения можно воспользоваться литературой, посвященной технологии Java и, в частности, апплетам, а также огромными ресурсами Интернета. Лучший способ изучения — создание собственных апплетов. В качестве образца можно использовать примеры, в большом

количестве размещенные на различных сайтах. Отправной точкой поиска может послужить один из следующих адресов:

http://www.gamelan.com

http://www.javaworld.com

http://www.ibm.com/java/

http://www.javasoft.com

http://javaboutique.com

ПРИЛОЖЕНИЕ 1

Поддержка тэгов и параметров HTML-браузерами

В нижеследующей таблице показана поддержка тэгов и параметров тэгов тремя ведущими браузерами. Приводимые данные справедливы для следующих версий браузеров:

- Internet Explorer, версии 4.01 (Windows 98/95/NT)
- Netscape, версия 4.04 (Windows 98/95/NT)
- Mosaic, версия 3.0 (Windows 95)

Если в ячейке таблицы фигурирует вопросительный знак, то это означает, что поддержка данного элемента была заявлена разработчиками браузера, но при проверке этого не оказалось.

В колонке "Спецификация" приводится номер спецификации HTML (1.0, 2.0, 3.2 или 4.0), в которой был введен данный тэг или параметр. Если элемент не вошел в спецификацию, то указывается версия браузера, в который впервые был применен данный элемент. Условные сокращения:

NS: Netscape

IE: Internet Explorer

Mos: Mosaic

Все тэги, которые допустимо использовать в разделе `<BODY>` HTML-документа, могут иметь параметры `CLASS`, `ID`, `LANG`, `LANGUAGE`, `STYLE` и `TITLE`. Из соображений компактности таблицы эти параметры не приводятся.

Параметры `CLASS`, `ID`, `STYLE` поддерживаются Internet Explorer, начиная с версии 3.0, и Netscape, начиная с версии 4.0.

Параметры `LANG`, `LANGUAGE`, `TITLE` — поддерживаются только Internet Explorer, начиная с версии 4.0.

Поддержка тэгов и параметров HTML-браузерами

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
`<A>`	Да	Да	Да	1.0
`ACCESSKEY`	Да	Нет	Нет	4.0

(продолжение)

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
DATAFLD	Да	Нет	Нет	IE 4.0
DATASRC	Да	Нет	Нет	IE 4.0
HREF	Да	Да	Да	2.0
INDEXSTRING	Нет	Да	Нет	NS 4.0
METHODS	?	?	?	2.0
NAME	Да	Да	Да	2.0
REL	?	?	?	2.0
REV	?	?	?	2.0
TARGET	Да	Да	Нет	4.0
TOCSTRING	Нет	Да	Нет	NS 4.0
URN	?	?	?	2.0
<ACRONYM>	Да	Нет	Нет	4.0
<ADDRESS>	Да	Да	Да	2.0
<APPLET>	Да	Да	Нет	3.2
ALIGN	Да	Да	Нет	3.2/NS 2.0
ALT	Да	Да	Нет	3.2/NS 2.0
ARCHIVE	Нет	Да	Нет	3.2/NS 3.0
CODE	Да	Да	Нет	3.2/NS 2.0
CODEBASE	Да	Да	Нет	3.2/NS 2.0
DATAFLD	Да	Нет	Нет	IE 4.0
DATASRC	Да	Нет	Нет	IE 4.0
HEIGHT	Да	Да	Нет	3.2/NS 2.0
HSPACE	Да	Да	Нет	3.2/NS 2.0
MAYSCRIPT	Нет	Да	Нет	NS 3.0
NAME	Да	Да	Нет	3.2/NS 2.0
SRC	Да	Да	Нет	3.2/NS 2.0
WIDTH	Да	Да	Нет	3.2/NS 2.0
<AREA>	Да	Да	Да	3.2
ALT	Да	Да	Да	3.2/IE 2.0

(продолжение)

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
COORDS	Да	Да	Да	3.2/IE 2.0
HREF	Да	Да	Да	3.2/IE 2.0
NAME	Да	Да	Нет	3.2/IE 2.0
SHAPE	Да	Да	Да	3.2/IE 2.0
TARGET	Да	Да	Нет	3.2/IE 2.0
<B>	Да	Да	Да	3.2
<BASE>	Да	Да	Да	3.2
HREF	Да	Да	Нет	2.0
TARGET	Да	Да	Нет	2.0/NS 2.0
<BASEFONT>	Да	Да	Нет	4.0
COLOR	Да	Да	Нет	4.0
FACE	Да	Нет	Нет	4.0/IE 2.0
SIZE	Да	Да	Нет	4.0
<BGSOUND>	Да	Нет	Да	IE 2.0
BALANCE	Да	Нет	Нет	IE 4.0
LOOP	Да	Нет	Нет	IE 2.0
SRC	Да	Нет	Нет	IE 2.0
VOLUME	Да	Нет	Нет	IE 2.0
<BIG>	Да	Да	Да	3.2
<BLOCKQUOTE>	Да	Да	Да	2.0
<BODY>	Да	Да	Да	2.0
ALINK	Да	Да	Нет	3.2
BACKGROUND	Да	Да	Нет	3.2
BGCOLOR	Да	Да	Нет	3.2
BGPROPERTIES	Да	Нет	Нет	IE 2.0
BOTTOMMARGIN	Да	Нет	Нет	IE 4.0
LEFTMARGIN	Да	Нет	Нет	IE 4.0
LINK	Да	Да	Нет	3.2

(продолжение)

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
RIGHTMARGIN	Да	Нет	Нет	IE 4.0
SCROLL	Да	Нет	Нет	IE 4.0
TEXT	Да	Да	Нет	3.2
TOPMARGIN	Да	Нет	Нет	IE 4.0
VLINK	Да	Да	Нет	3.2
 	Да	Да	Да	2.0
CLEAR	Да	Да	Да	3.2
<BUTTON>	Да	Нет	Нет	4.0
ACCESSKEY	Да	Нет	Нет	4.0
DATAFLD	Да	Нет	Нет	IE 4.0
DATAFORMATAS	Да	Нет	Нет	IE 4.0
DATASRC	Да	Нет	Нет	IE 4.0
DISABLED	Да	Нет	Нет	4.0
TYPE	Да	Нет	Нет	4.0
<CAPTION>	Да	Да	Да	3.2
ALIGN	Да	Да	Нет	3.2
VALIGN	Да	Нет	Нет	4.0/IE 2.0
<BIG>	Да	Да	Да	3.2
<CITE>	Да	Да	Да	2.0
<CODE>	Да	Да	Да	2.0
<COL>	Да	Нет	Да	4.0
ALIGN	Да	Нет	Нет	4.0
VALIGN	Да	Нет	Нет	4.0
<COLGROUP>	Да	Нет	Да	4.0
ALIGN	Да	Нет	Нет	4.0
SPAN	Да	Нет	Нет	4.0
VALIGN	Да	Нет	Нет	4.0
WIDTH	Да	Нет	Нет	4.0

(продолжение)

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
<COMMENT>	Да	Нет	Да	4.0
<DD>	Да	Да	Да	2.0
<DEL>	Да	Нет	Нет	4.0
CITE	Да	Нет	Нет	4.0
DATETIME	Да	Нет	Нет	4.0
<DFN>	Да	Нет	Нет	3.2
<DIR>	Да	Да	Да	2.0
<DIV>	Да	Да	Нет	3.2
ALIGN	Да	Да	Нет	3.2
DATAFLD	Да	Нет	Нет	IE 4.0
DATAFORMATAS	Да	Нет	Нет	IE 4.0
DATASRC	Да	Нет	Нет	IE 4.0
COMPACT	Да	Да	Нет	3.2
<DT>	Да	Да	Да	2.0
<EM>	Да	Да	Да	2.0
<EMBED>	Да	Да	Нет	3.2
ALIGN	Да	Да	Нет	3.2
ALT	Да	Да	Нет	3.2
BORDER	Да	Да	Нет	3.2
HIDDEN	Нет	Да	Нет	3.2
HSPACE	Да	Да	Нет	3.2
NAME	Да	Да	Нет	3.2
PALETTE	Да	Да	Нет	3.2
PLUGINSPACE	Да	Да	Нет	3.2
VSPACE	Да	Да	Нет	3.2
<FIELDSET>	Да	Нет	Нет	4.0

(продолжение)

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
<FONT>	Да	Да	Да	3.2
COLOR	Да	Да	Нет	3.2
FACE	Да	Да	Нет	3.2
POINT-SIZE	Нет	Да	Нет	NS 4.0
SIZE	Да	Да	Нет	3.2
WEIGHT	Нет	Да	Нет	NS 4.0
<FORM>	Да	Да	Да	2.0
ACTION	Да	Да	Да	2.0
ENCTYPE	Да	Да	Да	2.0
METHOD	Да	Да	Да	2.0
NAME	Да	Да	Да	2.0
TARGET	Да	Да	Нет	3.2
<FRAME>	Да	Да	Нет	4.0
BORDERCOLOR	Да	Да	Нет	4.0
DATAFLD	Да	Нет	Нет	IE 4.0
DATASRC	Да	Нет	Нет	IE 4.0
FRAMEBORDER	Да	Да	Нет	4.0
FRAMESPACING	Да	Да	Нет	4.0
MARGINHEIGHT	Да	Да	Нет	4.0
MARGINWIDTH	Да	Да	Нет	4.0
NORESIZE	Да	Да	Нет	4.0
SCROLLING	Да	Да	Нет	4.0
SRC	Да	Да	Нет	4.0
<FRAMESET>	Да	Да	Нет	4.0
BORDER	Да	Да	Нет	4.0
BORDERCOLOR	Да	Да	Нет	4.0
COLS	Да	Да	Нет	4.0
FRAMEBORDER	Да	Да	Нет	4.0
FRAMESPACING	Да	Да	Нет	4.0
ROWS	Да	Да	Нет	4.0

(продолжение)

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
<HR>	Да	Да	Да	2.0
ALIGN	Да	Да	Да	3.2
COLOR	Да	Нет	Да	3.2
NOSHADE	Да	Да	Да	3.2
SIZE	Да	Да	Да	3.2
WIDTH	Да	Да	Да	3.2
<Hx>	Да	Да	Да	2.0
ALIGN	Да	Да	Да	3.2
<I>	Да	Да	Да	2.0
<IFRAME>	Да	Нет	Нет	4.0
ALIGN	Да	Нет	Нет	4.0
BORDER	Да	Нет	Нет	4.0
BORDERCOLOR	Да	Нет	Нет	4.0
DATAFLD	Да	Нет	Нет	IE 4.0
DATASRC	Да	Нет	Нет	IE 4.0
FRAMEBORDER	Да	Нет	Нет	4.0
HEIGHT	Да	Нет	Нет	4.0
HSPACE	Да	Нет	Нет	4.0
NAME	Да	Нет	Нет	4.0
SCROLLING	Да	Нет	Нет	4.0
SRC	Да	Нет	Нет	4.0
VSPACE	Да	Нет	Нет	4.0
WIDTH	Да	Нет	Нет	4.0
<ILAYER>	Нет	Да	Нет	NS 4.0
ABOVE	Нет	Да	Нет	NS 4.0
BACKGROUND	Нет	Да	Нет	NS 4.0
BELOW	Нет	Да	Нет	NS 4.0
BGCOLOR	Нет	Да	Нет	NS 4.0
LEFT	Нет	Да	Нет	NS 4.0

(продолжение)

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
NAME	Нет	Да	Нет	NS 4.0
SRC	Нет	Да	Нет	NS 4.0
TOP	Нет	Да	Нет	NS 4.0
VISIBILITY	Нет	Да	Нет	NS 4.0
WIDTH	Нет	Да	Нет	NS 4.0
Z-INDEX	Нет	Да	Нет	NS 4.0
<IMG>	Да	Да	Да	2.0
ALIGN	Да	Да	Да	2.0
ALT	Да	Да	Да	2.0
BORDER	Да	Да	Да	2.0
CONTROLS	Да	Нет	Нет	IE 2.0
DATAFLD	Да	Нет	Нет	IE 4.0
DATASRC	Да	Нет	Нет	IE 4.0
DYNSRC	Да	Нет	Нет	IE 2.0
HEIGHT	Да	Да	Да	2.0
HSPACE	Да	Да	Да	3.2
ISMAP	Да	Да	Да	2.0
LOOP	Да	Нет	Нет	IE 2.0
LOOPDELAY	Да	Нет	Нет	IE 2.0
LOWSRC	Да	Да	Нет	3.2
NAME	Да	Да	Нет	3.2
SRC	Да	Да	Да	2.0
START	Да	Нет	Нет	IE 2.0
USEMAP	Да	Да	Да	3.2
VRML	Да	Нет	Нет	IE 3.0
VSPACE	Да	Да	Да	3.2
WIDTH	Да	Да	Да	2.0
<INPUT>	Да	Да	Да	2.0
ACCESSKEY	Да	Нет	Нет	4.0
ALIGN	Да	Да	Да	2.0

(продолжение)

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
CHECKED	Да	Да	Да	2.0
DATAFLD	Да	Нет	Нет	IE 4.0
DATAFORMATAS	Да	Нет	Нет	IE 4.0
DATASRC	Да	Нет	Нет	IE 4.0
DISABLED	Да	Нет	Нет	4.0
MAXLENGTH	Да	Да	Да	2.0
NAME	Да	Да	Да	2.0
READONLY	Да	Нет	Нет	4.0
SIZE	Да	Да	Да	2.0
SRC	Да	Да	Да	2.0
TABINDEX	Да	Нет	Нет	4.0
TYPE	Да	Да	Да	2.0
VALUE	Да	Да	Да	2.0
<INS>	Да	Нет	Нет	4.0
CITE	Да	Нет	Нет	4.0
DATETIME	Да	Нет	Нет	4.0
<ISINDEX>	Да	Да	Да	2.0
ACTION	Да	Да	Нет	2.0
PROMPT	Да	Да	Нет	2.0
<KBD>	Да	Да	Да	2.0
<KEYGEN>	Нет	Да	Нет	NS 3.0
CHALLENGE	Нет	Да	Нет	NS 3.0
NAME	Нет	Да	Нет	NS 3.0
<LABEL>	Да	Нет	Нет	4.0
ACCESSKEY	Да	Нет	Нет	4.0
DATAFLD	Да	Нет	Нет	IE 4.0
DATAFORMATAS	Да	Нет	Нет	IE 4.0
DATASRC	Да	Нет	Нет	IE 4.0
FOR	Да	Нет	Нет	4.0

(продолжение)

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
<LAYER>	Нет	Да	Нет	NS 4.0
ABOVE	Нет	Да	Нет	NS 4.0
BACKGROUND	Нет	Да	Нет	NS 4.0
BELOW	Нет	Да	Нет	NS 4.0
BGCOLOR	Нет	Да	Нет	NS 4.0
CLIP	Нет	Да	Нет	NS 4.0
LEFT	Нет	Да	Нет	NS 4.0
NAME	Нет	Да	Нет	NS 4.0
SRC	Нет	Да	Нет	NS 4.0
TOP	Нет	Да	Нет	NS 4.0
VISIBILITY	Нет	Да	Нет	NS 4.0
WIDTH	Нет	Да	Нет	NS 4.0
Z-INDEX	Нет	Да	Нет	NS 4.0
<LEGEND>	Да	Нет	Нет	4.0
ALIGN	Да	Нет	Нет	4.0
<LI>	Да	Да	Да	2.0
TYPE	Да	Да	Нет	4.0
<LINK>	Да	Да	Да	2.0
DISABLED	Да	Нет	Нет	2.0
HREF	Да	Да	Да	2.0
REL	Да	Да	Да	2.0
REV	Нет	Нет	Да	2.0
TYPE	Да	Да	Да	2.0
<LISTING>	Да	Да	Да	2.0
<MAP>	Да	Да	Да	3.2
NAME	Да	Да	Да	3.2
<MARQUEE>	Да	Нет	Нет	IE 2.0
ALIGN	Да	Нет	Нет	IE 2.0

(продолжение)

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
BEHAVIOR	Да	Нет	Нет	IE 2.0
BGCOLOR	Да	Нет	Нет	IE 2.0
DATAFLD	Да	Нет	Нет	IE 4.0
DATAFORMATAS	Да	Нет	Нет	IE 4.0
DATASRC	Да	Нет	Нет	IE 4.0
DIRECTION	Да	Нет	Нет	IE 2.0
HEIGHT	Да	Нет	Нет	IE 2.0
HSPACE	Да	Нет	Нет	IE 2.0
LOOP	Да	Нет	Нет	IE 2.0
SCROLLAMOUNT	Да	Нет	Нет	IE 2.0
SCROLLDELAY	Да	Нет	Нет	IE 2.0
TRUESPEED	Да	Нет	Нет	IE 4.0
VSPACE	Да	Нет	Нет	IE 2.0
WIDTH	Да	Нет	Нет	IE 2.0
<MENU>	Да	Да	Да	2.0
<META>	Да	Да	Да	2.0
CONTENT	Да	Да	Да	2.0
HTTP-EQUIV	Да	Да	Да	2.0
NAME	Да	Да	Да	2.0
<MULTICOL>	Нет	Да	Нет	NS 3.0
COLS	Нет	Да	Нет	NS 3.0
GUTTER	Нет	Да	Нет	NS 3.0
WIDTH	Нет	Да	Нет	NS 3.0
<OBJECT>	Да	Нет	Да	4.0
ACCESSKEY	Да	Нет	Нет	4.0
ALIGN	Да	Нет	Да	3.2
BORDER	Да	Нет	Нет	4.0
CLASSID	Да	Нет	Да	4.0
CODE	Да	Нет	Нет	4.0

(продолжение)

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
CODEBASE	Да	Нет	Да	4.0
DATA	Да	Нет	Нет	4.0
DATAFLD	Да	Нет	Нет	IE 4.0
DATASRC	Да	Нет	Нет	IE 4.0
HEIGHT	Да	Нет	Нет	4.0
HSPACE	Да	Нет	Нет	4.0
NAME	Да	Нет	Нет	4.0
TABINDEX	Да	Нет	Нет	4.0
TYPE	Да	Нет	Нет	4.0
VSPACE	Да	Нет	Нет	4.0
WIDTH	Да	Нет	Нет	4.0
<OL>	Да	Да	Да	2.0
START	Да	Да	Нет	3.2
TYPE	Да	Да	Нет	3.2
<OPTION>	Да	Да	Да	2.0
NAME	Да	Да	Нет	3.2
SELECTED	Да	Да	Да	2.0
TYPE	Да	Да	Нет	3.2
VALUE	Да	Да	Нет	2.0
<P>	Да	Да	Да	2.0
ALIGN	Да	Да	Нет	3.2
<PARAM>	Да	Да	Да	3.2
DATAFLD	Да	Нет	Нет	IE 4.0
DATAFORMATAS	Да	Нет	Нет	IE 4.0
DATASRC	Да	Нет	Нет	IE 4.0
NAME	Да	Да	Да	3.2
VALUE	Да	Да	Да	3.2
<PLAINTEXT>	Да	Да	Да	2.0

(продолжение)

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
<PRE>	Да	Да	Да	2.0
WIDTH	Да	Да	Да	2.0
<Q>	Да	Нет	Нет	4.0
CITE	Да	Нет	Нет	4.0
<S>	Да	Да	Да	2.0
<SAMP>	Да	Да	Да	2.0
<SELECT>	Да	Да	Да	2.0
ACCESSKEY	Да	Нет	Нет	4.0
ALIGN	Да	Да	Да	2.0
DATAFLD	Да	Нет	Нет	IE 4.0
DATASRC	Да	Нет	Нет	IE 4.0
DISABLED	Да	Нет	Нет	4.0
MULTIPLE	Да	Да	Да	2.0
NAME	Да	Да	Да	2.0
SIZE	Да	Да	Да	2.0
TABINDEX	Да	Нет	Нет	4.0
TYPE	Да	Да	Нет	4.0
<SMALL>	Да	Да	Да	3.2
<SOUND>	Нет	Нет	Да	Mos 3.0
DELAY	Нет	Нет	Да	Mos 3.0
LOOP	Нет	Нет	Да	Mos 3.0
SRC	Нет	Нет	Да	Mos 3.0
<SPACER>	Нет	Да	Нет	NS 3.0
ALIGN	Нет	Да	Нет	NS 3.0
HEIGHT	Нет	Да	Нет	NS 3.0
SIZE	Нет	Да	Нет	NS 3.0
TYPE	Нет	Да	Нет	NS 3.0
WIDTH	Нет	Да	Нет	NS 3.0

(продолжение)

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
<SPAN>	Да	Да	Нет	3.2
DATAFLD	Да	Нет	Нет	IE 4.0
DATAFORMATAS	Да	Нет	Нет	IE 4.0
DATASRC	Да	Нет	Нет	IE 4.0
DIR	Да	Нет	Нет	3.2
<STRIKE>	Да	Да	Да	2.0
<STRONG>	Да	Да	Да	2.0
<STYLE>	Да	Да	Нет	3.2
DISABLED	Да	Да	Нет	4.0
MEDIA	Да	Да	Нет	4.0
TYPE	Да	Да	Нет	3.2
<SUB>	Да	Да	Да	2.0
<SUP>	Да	Да	Да	2.0
<TABLE>	Да	Да	Да	3.2
ALIGN	Да	Да	Да	3.2
BACKGROUND	Да	Да	Нет	IE 3.0
BGCOLOR	Да	Да	Нет	IE 3.0
BORDER	Да	Да	Нет	3.2
BORDERCOLOR	Да	Да	Нет	IE 3.0
BORDERCOLORDARK	Да	Нет	Нет	IE 3.0
BORDERCOLORLIGHT	Да	Нет	Нет	IE 3.0
CELLPADDING	Да	Да	Да	3.2
CELLSPACING	Да	Да	Да	3.2
COLS	Да	Нет	Нет	4.0
DATAFLD	Да	Нет	Нет	IE 4.0
DATAPAGESIZE	Да	Нет	Нет	IE 4.0
DATASRC	Да	Нет	Нет	IE 4.0
FRAME	Да	Нет	Нет	3.2

(продолжение)

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
HEIGHT	Да	Да	Да	3.2
RULES	Да	Нет	Нет	3.2
VALIGN	Да	Да	Нет	4.0
WIDTH	Да	Да	Да	3.2
<TBODY>	Да	Нет	Нет	3.2
ALIGN	Да	Нет	Нет	4.0
BGCOLOR	Да	Нет	Нет	IE 4.0
VALIGN	Да	Нет	Нет	4.0
<TD>	Да	Да	Да	3.2
ALIGN	Да	Да	Нет	3.2
BACKGROUND	Да	Да	Нет	IE 3.0
BGCOLOR	Да	Да	Нет	IE 3.0
BORDERCOLOR	Да	Да	Нет	IE 3.0
BORDERCOLORDARK	Да	Нет	Нет	IE 3.0
BORDERCOLORLIGHT	Да	Нет	Нет	IE 3.0
COLSPAN	Да	Да	Да	3.2
HEIGHT	Да	Да	Нет	3.2
NOWRAP	Да	Да	Да	3.2
ROWSPAN	Да	Да	Да	3.2
VALIGN	Да	Да	Нет	3.2
WIDTH	Да	Да	Нет	3.2
<TEXTAREA>	Да	Да	Да	2.0
ACCESSKEY	Да	Нет	Нет	4.0
ALIGN	Да	Да	Да	2.0
COLS	Да	Да	Да	2.0
DATAFLD	Да	Нет	Нет	IE 4.0
DATASRC	Да	Нет	Нет	IE 4.0
DISABLED	Да	Нет	Нет	IE 4.0
NAME	Да	Да	Да	2.0

(продолжение)

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
READONLY	Да	Нет	Нет	4.0
ROWS	Да	Да	Да	2.0
TABINDEX	Да	Нет	Нет	4.0
WRAP	Да	Да	Нет	3.2
<TFOOT>	Да	Нет	Нет	3.2
ALIGN	Да	Нет	Нет	4.0
BGCOLOR	Да	Нет	Нет	IE 4.0
VALIGN	Да	Нет	Нет	4.0
<TH>	Да	Да	Да	3.2
ALIGN	Да	Да	Да	3.2
BACKGROUND	Да	Да	Нет	IE 3.0
BGCOLOR	Да	Да	Нет	IE 3.0
BORDERCOLOR	Да	Да	Нет	IE 3.0
BORDERCOLORDARK	Да	Нет	Нет	IE 3.0
BORDERCOLORLIGHT	Да	Нет	Нет	IE 3.0
COLSPAN	Да	Да	Да	3.2
HEIGHT	Да	Да	Да	3.2
NOWRAP	Да	Да	Да	3.2
ROWSPAN	Да	Да	Да	3.2
VALIGN	Да	Да	Да	3.2
WIDTH	Да	Да	Да	3.2
<THEAD>	Да	Нет	Нет	3.2
ALIGN	Да	Нет	Нет	4.0
BGCOLOR	Да	Нет	Нет	IE 4.0
VALIGN	Да	Нет	Нет	4.0
<TR>	Да	Да	Да	3.2
ALIGN	Да	Да	Да	3.2
BACKGROUND	Нет	Да	Нет	NS 3.0
BGCOLOR	Да	Да	Да	3.2

(окончание)

<Тэг>/параметр	Internet Explorer	Netscape	Mosaic	Спецификация
BORDERCOLOR	Да	Да	Нет	IE 3.0
BORDERCOLORDARK	Да	Нет	Нет	IE 3.0
BORDERCOLORLIGHT	Да	Нет	Нет	IE 3.0
VALIGN	Да	Да	Да	3.2
<TT>	Да	Да	Да	2.0
<U>	Да	Да	Да	2.0
<UL>	Да	Да	Да	2.0
TYPE	Да	Да	Да	3.2
<VAR>	Да	Да	Да	2.0
<XMP>	Да	Да	Да	2.0

ПРИЛОЖЕНИЕ 2

Названия и коды цветов для HTML

В нижеследующей таблице приведены в алфавитном порядке названия 140 стандартных цветов, которые можно использовать в HTML-документах по их названию. Поддержка этих названий впервые была реализована в Netscape Navigator 3.0. В дальнейшем их поддержка была включена и в браузер Microsoft Internet Explorer 4.0. В таблице отмечены звездочками 16 стандартных цветов, поддержка которых обеспечивалась и ранее.

В таблице даны также значения составляющих для каждого цвета (RGB) в шестнадцатеричном и десятичном виде.

Названия и коды цветов для HTML

	Название цвета	Шестнадцатеричное значение RGB	Десятичные значения		
			R	G	B
1.	aliceblue	#F0F8FF	240	248	255
2.	antiquewhite	#FAEBD7	250	235	215
3.	*aqua	#00FFFF	0	255	255
4.	aquamarine	#7FFFD4	127	255	212
5.	azure	#F0FFFF	240	255	255
6.	beige	#F5F5DC	245	245	220
7.	bisque	#FFE4C4	255	228	196
8.	*black	#000000	0	0	0
9.	blanchedalmond	#FFEBCD	255	235	205
10.	*blue	#0000FF	0	0	255
11.	blueviolet	#8A2BE2	138	43	226
12.	brown	#A52A2A	165	42	42
13.	burlywood	#DEB887	222	184	135

(продолжение)

	Название цвета	Шестнадцатеричное значение RGB	Десятичные значения		
			R	G	B
14.	cadetblue	#5F9EA0	95	158	160
15.	chartreuse	#7FFF00	127	255	0
16.	chocolate	#D2691E	210	105	30
17.	coral	#FF7F50	255	127	80
18.	cornflowerblue	#6495ED	100	149	237
19.	cornsilk	#FFF8DC	255	248	220
20.	crimson	#DC143C	220	20	60
21.	cyan	#00FFFF	0	255	255
22.	darkblue	#00008B	0	0	139
23.	darkcyan	#008B8B	0	139	139
24.	darkgoldenrod	#B8860B	184	134	11
25.	darkgray	#A9A9A9	169	169	169
26.	darkgreen	#006400	0	100	0
27.	darkkhaki	#BDB76B	189	183	107
28.	darkmagenta	#8B008B	139	0	139
29.	darkolivegreen	#556B2F	85	107	47
30.	darkorange	#FF8C00	255	140	0
31.	darkorchid	#9932CC	153	50	204
32.	darkred	#8B0000	139	0	0
33.	darksalmon	#E9967A	233	150	122
34.	darkseagreen	#8FBC8F	143	188	143
35.	darkslateblue	#483D8B	72	61	139
36.	darkslategray	#2F4F4F	47	79	79
37.	darkturquoise	#00CED1	0	206	209
38.	darkviolet	#9400D3	148	0	211
39.	deeppink	#FF1493	255	20	147
40.	deepskyblue	#00BFFF	0	191	255
41.	dimgray	#696969	105	105	105
42.	dodgerblue	#1E90FF	30	144	255

(продолжение)

	Название цвета	Шестнадцатеричное значение RGB	Десятичные значения		
			R	G	B
43.	firebrick	#B22222	178	34	34
44.	floralwhite	#FFFAF0	255	250	240
45.	forestgreen	#228B22	34	139	34
46.	*fuchsia	#FF00FF	255	0	255
47.	ghostwhite	#F8F8FF	248	248	255
48.	gainsboro	#DCDCDC	220	220	220
49.	gold	#FFD700	255	215	0
50.	goldenrod	#DAA520	218	165	32
51.	*gray	#808080	128	128	128
52.	*green	#008000	0	128	0
53.	greenyellow	#ADFF2F	173	255	47
54.	honeydew	#F0FFF0	240	255	240
55.	hotpink	#FF69B4	255	105	180
56.	indianred	#CD5C5C	205	92	92
57.	indigo	#4B0082	75	0	130
58.	ivory	#FFFFF0	255	255	240
59.	khaki	#F0E68C	240	230	140
60.	lavender	#E6E6FA	230	230	250
61.	lavenderblush	#FFF0F5	255	240	245
62.	lawngreen	#7CFC00	124	252	0
63.	lemonchiffon	#FFFACD	255	250	205
64.	lightblue	#ADD8E6	173	216	230
65.	lightcoral	#F08080	240	128	128
66.	lightcyan	#E0FFFF	224	255	255
67.	lightgoldenrodyellow	#FAFAD2	250	250	210
68.	lightgreen	#90EE90	144	238	144
69.	lightgrey	#D3D3D3	211	211	211
70.	lightpink	#FFB6C1	255	182	193
71.	lightsalmon	#FFA07A	255	160	122

(продолжение)

	Название цвета	Шестнадцатеричное значение RGB	Десятичные значения		
			R	G	B
72.	lightseagreen	#20B2AA	32	178	170
73.	lightskyblue	#87CEFA	135	206	250
74.	lightslategray	#778899	119	136	153
75.	lightsteelblue	#B0C4DE	176	196	222
76.	lightyellow	#FFFFE0	255	255	224
77.	*lime	#00FF00	0	255	0
78.	limegreen	#32CD32	50	205	50
79.	linen	#FAF0E6	250	240	230
80.	magenta	#FF00FF	255	0	255
81.	*maroon	#800000	128	0	0
82.	mediumaquamarine	#66CDAA	102	205	170
83.	mediumblue	#0000CD	0	0	205
84.	mediumorchid	#BA55D3	186	85	211
85.	mediumpurple	#9370DB	147	112	219
86.	mediumseagreen	#3CB371	60	179	113
87.	mediumslateblue	#7B68EE	123	104	238
88.	mediumspringgreen	#00FA9A	0	250	154
89.	mediumturquoise	#48D1CC	72	209	204
90.	mediumvioletred	#C71585	199	21	133
91.	midnightblue	#191970	25	25	112
92.	mintcream	#F5FFFA	245	255	250
93.	mistyrose	#FFE4E1	255	228	225
94.	moccasin	#FFE4B5	255	228	181
95.	navajowhite	#FFDEAD	255	222	173
96.	*navy	#000080	0	0	128
97.	oldlace	#FDF5E6	253	245	230
98.	*olive	#808000	128	128	0
99.	olivedrab	#6B8E23	107	142	35
100.	orange	#FFA500	255	165	0

(продолжение)

	Название цвета	Шестнадцатеричное значение RGB	Десятичные значения		
			R	G	B
101.	orangered	#FF4500	255	69	0
102.	orchid	#DA70D6	218	112	214
103.	palegoldenrod	#EEE8AA	238	232	170
104.	palegreen	#98FB98	152	251	152
105.	paleturquoise	#AFEEEE	175	238	238
106.	palevioletred	#DB7093	219	112	147
107.	papayawhip	#FFEFD5	255	239	213
108.	peachpuff	#FFDAB9	255	218	185
109.	peru	#CD853F	205	133	63
110.	pink	#FFC0CB	255	192	203
111.	plum	#DDA0DD	221	160	221
112.	powderblue	#B0E0E6	176	224	230
113.	*purple	#800080	128	0	128
114.	*red	#FF0000	255	0	0
115.	rosybrown	#BC8F8F	188	143	143
116.	royalblue	#4169E1	65	105	225
117.	saddlebrown	#8B4513	139	69	19
118.	salmon	#FA8072	250	128	114
119.	sandybrown	#F4A460	244	164	96
120.	seagreen	#2E8B57	46	139	87
121.	seashell	#FFF5EE	255	245	238
122.	sienna	#A0522D	160	82	45
123.	*silver	#C0C0C0	192	192	192
124.	skyblue	#87CEEB	135	206	235
125.	slateblue	#6A5ACD	106	90	205
126.	slategray	#708090	112	128	144
127.	snow	#FFFAFA	255	250	250
128.	springgreen	#00FF7F	0	255	127
129.	steelblue	#4682B4	70	130	180

(окончание)

	Название цвета	Шестнадцатеричное значение RGB	Десятичные значения		
			R	G	B
130.	tan	#D2B48C	210	180	140
131.	*teal	#008080	0	128	128
132.	thistle	#D8BFD8	216	191	216
133.	tomato	#FF6347	255	99	71
134.	turquoise	#40E0D0	64	224	208
135.	violet	#EE82EE	238	130	238
136.	wheat	#F5DEB3	245	222	179
137.	*white	#FFFFFF	255	255	255
138.	whitesmoke	#F5F5F5	245	245	245
139.	*yellow	#FFFF00	255	255	0
140.	yellowgreen	#9ACD32	154	205	50

Примечания

1. Цвет aliceblue (первый в списке) отображается браузером Netscape версии 3.x не так, как приведено в таблице. В последующих версиях браузера (4.0 и выше) эта ошибка была исправлена.
2. Заметим, что в перечне наименований имеются названия, которые описывают одинаковые по содержанию цвета, — aqua и cyan, а также fuchsia и magenta.
3. Несмотря на то, что приведенные названия цветов предложены компанией Netscape, при их отображении в режиме 256 цветов будет выполняться их коррекция для приведения к палитре Netscape. Палитра Netscape включает 16 стандартных цветов, а также 216 цветов, получаемых всевозможными комбинациями чисел 0, 51, 102, 153, 204 и 255, используемых в качестве значений для R, G и B. Отображение цвета в точном соответствии с данной таблицей будет выполняться только для режимов с большей глубиной цвета – HiColor и TrueColor.
4. При задании цвета по названию необходимо точно придерживаться его написания. При сомнениях в написании названия по памяти можно свериться с приводимой таблицей. Смысл этого совета состоит в следующем. Некоторые названия имеют довольно длинное трудно запоминаемое написание. Есть отдельные странности в названиях. Так, слово "серый" в английском языке имеет два написания – "gray" и "grey". Обратите внимание, как записаны следующие названия: gray, darkgray, lightgrey.

ПРИЛОЖЕНИЕ 3

Совместимость приводов CD-ROM и программ для вычленения звуковых файлов

Производитель	Модель	Тип	Дейст-вует?	Скорость	Программа
Acer	CD607E/O	IDE/ATAPI	Нет	—	—
Acer	CD616A	IDE/ATAPI	Да	4 X	WinDAC32
Acer	CD620A	IDE/ATAPI	Да	4 X	WinDAC32
Acer	CD624A 003	IDE/ATAPI	Да	8 X	CDDA — DAC
Acer	CD665A	IDE/ATAPI	Нет	—	—
Acer	CD685A	IDE/ATAPI	Нет	—	—
Acer	CD743E	IDE/ATAPI	Нет	—	—
Acer	CD767E/S	IDE/ATAPI	Нет	—	—
Acer	CD-787E/SAS	IDE/ATAPI	Нет	—	—
Acer	CD-910E/JAS	IDE/ATAPI	Нет	—	—
Acer	CD-912E/ATK	IDE/ATAPI	Нет	—	—
Acer (AOpen)	CD-916E/ATK	IDE/ATAPI	Да	1 X	CD Worx 95"CDDA
Acer (AOpen)	CD-924E/AKO	IDE/ATAPI	Да	4 X	AudioGrabber – CDDA
Acer (AOpen)	CD-924R/AKO	IDE/ATAPI	Нет	—	—
Aiwa	ACD-620	SCSI	Нет	—	—
Aztech	AZT 66801I	IDE/ATAPI	Нет	—	—
Aztech	AZT 868ISE	IDE/ATAPI	Да	2 X	DAC — CD2WAV
BTC	SLL24	IDE/ATAPI	Да	3 X	WinDAC32
ChiHетn	CDS-545	—	Да	—	WinDAC32
Compaq	CR-503BCQ	SCSI	Да	1 X	CDDA — WinDAC32

(продолжение)

Производитель	Модель	Тип	Действует?	Скорость	Программа
CyberDrive	120D	IDE/ATAPI	Да	1 X	CD-Grab Audio/CD-Grab Pro
CyberDrive	TW240D	IDE/ATAPI	Да	3 X	WinDAC32
CreativeLabs	CD 620	IDE/ATAPI	Нет	—	—
CreativeLabs	CD 820	IDE/ATAPI	Да	—	CD Worx 95
CreativeLabs	CD 821E	IDE/ATAPI	Да	1 X	WinDAC32
CreativeLabs	CDR 511	SCSI	Нет	—	—
CreativeLabs	CDR 2423	IDE/ATAPI	Да	2 X	WinDAC32
CreativeLabs	INFRA 1800	IDE/ATAPI	Да	4 X	DAC — WinDAC32
CreativeLabs	INFRA 2400	IDE/ATAPI	Да	4 X	WinDAC32
CreativeLabs	INFRA 3600	IDE/ATAPI	Да	4 X	WinDAC32
CreativeLabs	PC-DVD Encore	IDE/ATAPI	Да	2 X	Easy CD Creator
Delta	ODC-6101	IDE/ATAPI	Нет	—	—
Dysan	CD-242E	IDE/ATAPI	Да	10 X	AudioGrabber — CD2WAV
Funai	E295X	IDE/ATAPI	Да	16 X	WinDAC32
Funai	E2850UA	IDE/ATAPI	Да	6 X	WinDAC32
Funai	E2650UA	IDE/ATAPI	Нет	—	—
Funai	E2920/2950	IDE/ATAPI	Да	—	WinDAC32
Gateway	8X for the Solo 2100 (laptop)	IDE/ATAPI	Нет	—	—
Goldstar	CRD-8160B	IDE/ATAPI	Да	—	DAC
Goldstar	CRD-8240B	IDE/ATAPI	Да	8 X	WinDAC32
Goldstar	GCD-R540C	IDE/ATAPI	Да	1 X	CDDA — WinDAC32
Goldstar	GCD-R560B	IDE/ATAPI	Да	—	CD Worx 95
Goldstar	GCD-R580B	IDE/ATAPI	Да	1 X	WinDAC32
Hewlett Packard	SURESTORE4020	SCSI	Да	4 X	WinDAC32
Hewlett Packard	SURESTORE4020	IDE/ATAPI	Да	2 X	WinDAC32
Hewlett Packard	6020I	SCSI	Да	6 X	Easy CD Pro 2.0
Hewlett Packard	7110I	IDE/ATAPI	Да	2 X	Easy CD Pro 2.0
Hitachi	CDR 7730	IDE/ATAPI	Да	1 X	WinDAC32
Hitachi	CDR 7830	IDE/ATAPI	Да	2.5 X	WinDAC32

(продолжение)

Производитель	Модель	Тип	Действует?	Скорость	Программа
Hitachi	CDR 7930	IDE/ATAPI	Да	4 X	WinDAC32
Hitachi	CDR 8130	IDE/ATAPI	Да	8 X	CDDA32
Hitachi	CDR 8230	IDE/ATAPI	Нет	—	—
JVC	2022	SCSI	Да	—	WinDAC32
LG Electronics	CRD-8160B	IDE/ATAPI	Нет	—	—
LG Electronics	CRD-8241B	IDE/ATAPI	Да	7 X	WinDAC32
Liteon	LTN242	IDE/ATAPI	Да	4 X	WinDAC32
Liteon	LTN244	IDE/ATAPI	Да	5 X	CD Copy
Liteon	LTN264	IDE/ATAPI	Да	4 X	WinDAC32
Matshita	CR 52X	IDE/ATAPI	Нет	—	—
Matshita	CR 504	SCSI	Нет	—	—
Matshita	CR 508	SCSI	Да	8 X	CDRWIN
Matshita	CR 562	IDE/ATAPI	Да	2 X	WinDAC32
Matshita	CR 562-B	IDE/ATAPI	Нет	—	—
Matshita	CR 563-B	—	Да	1.5 X	CDDA
Matshita	CR 574	IDE/ATAPI	Да	—	WinDAC32
Matshita	CR 581	IDE/ATAPI	Да	4 X	WinDAC32
Matshita	CR 583	IDE/ATAPI	Да	1 X	DAC
Matshita	CR 584	IDE/ATAPI	Да	9 X	WinDAC32 — CDDA
Matshita	CR 585	IDE/ATAPI	Да	8 X	WinDAC32
Mitsumi	CDR2600TE	—	Да	2 X	WinDAC32 — CD Copy
Mitsumi	FX001D	IDE/ATAPI	Нет	—	—
Mitsumi	FX120T !B	IDE/ATAPI	Да	12 X	WinDAC32
Mitsumi	FX140	IDE/ATAPI	Да	—	CD2WAV
Mitsumi	FX140S !B	—	Да	10 X	CDDA 1.6
Mitsumi	FX162	IDE/ATAPI	Да	8 X	WinDAC32
Mitsumi	FX240S !B	IDE/ATAPI	Да	8 X	WinDAC32
Mitsumi	FX400B	IDE/ATAPI	Нет	—	—
Mitsumi	FX400D	IDE/ATAPI	Нет	—	—
Mitsumi	FX410A!B	IDE/ATAPI	Нет	—	—
Mitsumi	FX600S	IDE/ATAPI	Нет	—	—

(продолжение)

Производитель	Модель	Тип	Действует?	Скорость	Программа
Mitsumi	FX800S !B	IDE/ATAPI	Да	8 X	WinDAC32
Mitsumi	FX810S !B	—	Да	4 X	WinDAC32
Mitsumi	FX810T	IDE/ATAPI	Да	2.6 X	WinDAC32 — CDDA
Nakamichi	MBR-7	SCSI	Нет	—	—
Nakamichi	MJ 4.4	IDE/ATAPI	Нет	—	—
Nakamichi	MJ 5.16	SCSI	Да	5 X	WinDAC32
Nec	251	—	Да	1 X	WinDAC32
Nec	252	—	Нет	—	—
Nec	260	IDE/ATAPI	Да	1 X	WinDAC32
Nec	272	IDE/ATAPI	Да	2 X	WinDAC32
Nec	273	IDE/ATAPI	Да	2 X	WinDAC32
Nec	282	—	Да	1.8 X	WinDAC32
Nec	288	IDE/ATAPI	Да	8 X	WinDAC32
Nec	462	SCSI	Да	1 X	WinDAC32
Nec	463	SCSI	Да	1 X	WinDAC32
Nec	4Xi	SCSI	Нет	—	—
Nec	502	SCSI	Да	1 X	WinDAC32
Nec	CDR-84-1	SCSI	Нет	—	—
Nec	CDR-1400B	IDE/ATAPI	Да	2 X	WinDAC32
Nec	CDR-1600	IDE/ATAPI	Нет	—	—
Nec	CDR-1610A	SCSI	Да	1 X	WinDAC32
Nec	CDR-1800	IDE/ATAPI	Да	8 X	WinDAC32 — CD Worx 95
Nec	CDR-1800A	IDE/ATAPI	Да	6 X	WinDAC32
Nec	CDR-1900A	IDE/ATAPI	Да	2 X	WinDAC32
Nec	Multi Spin 8V	—	Да	4 X	DAC
Olympus	CDR 2x6	SCSI	Да	2 X	Easy CD Creator
Panasonic	CR-506-C	SCSI	Да	8 X	CD Worx NT
Panasonic	CR-508-B	SCSI	Да	8 X	Spin Doctor (Easy CD Creator), CDRWIN
Panasonic	CR-562-B	IDE/ATAPI	Да	—	—

(продолжение)

Производитель	Модель	Тип	Действует?	Скорость	Программа
Panasonic	CR-572	—	Да	2 X	WinDAC32
Panasonic	CR-583	IDE/ATAPI	Да	2 X	WinDAC32 — CDDA
Panasonic	CR-584	IDE/ATAPI	Да	12 X	CD Copy
Panasonic	CR-585	IDE/ATAPI	Да	3 X	WinDAC32
Panasonic	CR-585-B	IDE/ATAPI	Да	4 X-24 X	WinDAC32
Panasonic	KXL-783A	SCSI	Нет	—	—
Philips	CDD 2000	SCSI	Да	4 X	WinDAC32
Philips	CDD 2600	SCSI	Да	6X	WinDAC32 — CD Copy
Philips	CDD 3610	IDE/ATAPI	Да	2 X	WinDAC32 — Easy CD Creator
Philips	CM205	IDE/ATAPI	Нет	—	—
Philips	PCA123CD	IDE/ATAPI	Да	1 X	CDDA
Philips	PCA162CD/M2	IDE/ATAPI	Да	3 X	WinDAC32
Philips	PCA202CD	IDE/ATAPI	Да	3 X	WinDAC32
Philips	PCA243CD/M2	IDE/ATAPI	Да	4 X	CDDA
Philips	ROD 1269	IDE/ATAPI	Да	2 X	CD Copy — CD Worx NT
Pioneer	DR-A01S	IDE/ATAPI	Да	3 X	WinDAC32 — CDDA
Pioneer	DR-A10X	IDE/ATAPI	Да	9 X	WinDAC32 — CDDA
Pioneer	DR A12X	IDE/ATAPI	Да	4 X	DAC — WinDAC32
Pioneer	DR A24X	IDE/ATAPI	Да	3 X	WinDAC32
Pioneer	DR-U03X	SCSI	Да	3 X	WinDAC32 — CDDA
Pioneer	DR-U12X	SCSI	Да	12 X	WinDAC32
Pioneer	DR-U24X	SCSI	Да	2.5 X	WinDAC32
Pioneer	DVD-A01	IDE/ATAPI	Да	7 X	WinDAC32
Plextor	PX-12TS	SCSI	Да	8 X	WinDAC32 — CDDA
Plextor	PX-20TS	SCSI	Да	10 X	WinDAC32
Plextor	PX-43CH	—	Да	4 X	WinDAC32
Plextor	UltraPlex 32	SCSI	Да	16 X	WinDAC32
Ricoh	1420c	—	Да	2 X	Easy CD Creator dlx 3.0
Ricoh	MP6200A	ICE/ATAPI	Да	4 X	WinDAC32

(продолжение)

Производитель	Модель	Тип	Действует?	Скорость	Программа
Ricoh	MP6200S	SCSI	Да	2 X	—
Samsung	SCR-830	IDE/ATAPI	Да	1 X	DAC
Samsung	SCR-1231 rev 1.01	IDE/ATAPI	Да	1 X	CDDA
Samsung	SCR-2030 JS103	IDE/ATAPI	Да	9 X	WinDAC32
Samsung	SCR-2430	—	Да	10 X	WinDAC32
Samsung	SCR-2431	IDE/ATAPI	Да	2 X	WinDAC32 — CDDA
Sanyo	CRD-254P	IDE/ATAPI	Нет	—	—
Sanyo	CRD-256P	—	Нет	—	—
Sanyo	CRD-820P	IDE/ATAPI	Нет	—	—
Smart and friendly	CD-R 1002	SCSI	Да	2 X	WinDAC32
Smart and friendly	SAF-2006	SCSI	Да	6 X	—
Sony	CDU31A	—	Да	—	CDDA
Sony	CDU31A	Propietary	Да	1 X	DAC
Sony	CDU55S	SCSI	Да	2 X	WinDAC32
Sony	CDU76E-Q	IDE/ATAPI	Нет	—	—
Sony	CDU77E-Q	IDE/ATAPI	Нет	—	—
Sony	CDU311	IDE/ATAPI	Да	8 X	WinDAC32 — ReadCDA
Sony	CDU415	SCSI	Да	3 X	WinDAC32
Sony	CDU511	IDE/ATAPI	Да	16 X	WinDAC32
Sony	CDU55E	—	Да	1 X	CD2WAV
Sony	CDU611	—	Да	9 X	WinDAC32
Sony	CDU76S	SCSI	Да	—	WinDAC32
Sony	CDU920S	SCSI	Да	2 X	WinDAC32
Sony	CDU926S	—	Да	6 X	WinDAC32 — Adaptec Software
Sony	CDU928E	IDE/ATAPI	Да	6 X	WinDAC32
Stingray	8422	IDE/ATAPI	Нет	—	—
Teac	CD-44E	Laptop	Да	1 X	WinDAC32
Teac	CD-46E	Laptop	Да	5 X	WinDAC32
Teac	CD-55A	IDE/ATAPI	Нет	—	—

(продолжение)

Производитель	Модель	Тип	Действует?	Скорость	Программа
Teac	CD-56E	IDE/ATAPI	Да	6 X	WinDAC32 — CDDA
Teac	CD-58E	IDE/ATAPI	Да	0.01 X	WinDAC32 — Easy Creator 3.0
Teac	CD-512E	—	Да	0.2 X	WinDAC32
Teac	CD-516S	SCSI	Да	4 X	WinDAC32 — CDDA
Teac	CD-524E	IDE/ATAPI	Да	12 X	WinDAC32
Teac	CD-532E	IDE/ATAPI	Да	10 X	WinDAC32
Teac	CD-R50S	SCSI	Да	4 X	WinDAC32
Teac	CD-R55S	SCSI	Да	8 X	WinDAC32
Techmedia	CDD-6100	—	Да	3 X	Any DOS based program
Techmedia	CDD-7120	IDE/ATAPI	Да	16 X	CD2WAV
Techmedia	CDD-7200	IDE/ATAPI	Нет	—	—
Torisan	CDR-N16	IDE/ATAPI	Bad	Slow	CD Worx 95
Torisan	CDR-S1G	IDE/ATAPI	Нет	—	—
Toshiba	SD-M1002 (DVD)	IDE/ATAPI	Да	1 X	WinDAC32
Toshiba	XM-3401TA	SCSI	Да	0.4 X	WinDAC32
Toshiba	XM-3701TA	SCSI	Да	0.5 X	WinDAC32
Toshiba	XM-5201B	SCSI	Нет	1 X	WinDAC32
Toshiba	XM-5302B	IDE/ATAPI	Да	1 X	CDDA — DAC
Toshiba	XM-5402TA	IDE/ATAPI	Да	1 X	WinDAC32 — ReadCDA
Toshiba	XM-5522B	—	Да	—	WinDAC32
Toshiba	XM-5602B	IDE/ATAPI	Да	1 X	WinDAC32 — CDDA
Toshiba	XM-5701	SCSI	Да	4 X	WinDAC32
Toshiba	XM-5702	IDE/ATAPI	Да	—	CDDA
Toshiba	XM-6002B	IDE/ATAPI	Да	1 X	WinDAC32
Toshiba	XM-6102B	IDE/ATAPI	Да	1 X	WinDAC32
Toshiba	XM-6202B	IDE/ATAPI	Да	4 X	WinDAC32
Octek	CDR-810	IDE/ATAPI	Нет	—	—
Vertos	V400HTDB	IDE/ATAPI	Нет	—	—
Wearnes	CDD-220	—	Нет	—	—

(окончание)

Производитель	Модель	Тип	Дейст-вует?	Скорость	Программа
Wearnes	CDD-620	IDE/ATAPI	Нет	—	—
Wearnes	CDD-820	IDE/ATAPI	Да	2 X	WinDAC32
Wearnes	CDD-2420	IDE/ATAPI	Да	2 X	WinDAC32
Yamaha	400T	SCSI	Да	7 X	Disc-At-Once — WinCD
Yamaha	401T	SCSI	Да	5 X	—
Yamaha	CDR100	SCSI	Да	2 X	WinDAC32 — CD Worx 95
Yamaha	CDR200T	SCSI	Да	2 X	WinCD
Yamaha	CRW400T	SCSI	Да	4 X	WinDAC32
Yamaha	CRW4001	IDE/ATAPI	Да	5 X	WinDAC32

Предметный указатель

B

Bitrate 251

C

CGI 401
CGI-сценарий 402, 414

D, F

Dither 100
Front-end 267

J

Java Development Kit (JDK) 598
JavaScript
  взаимодействие апплетов
    с JavaScript 622
  объект JavaArray 623
  объект JavaClass 623
  объект JavaObject 623
  объект JavaPackage 623
J-код 597

L, S

LiveConnect 622
Skin 259

T, U

Thumbnail 88
URL-кодирование 411

А

Апплеты 599, 603
  безопасность 600
  вставка рисунков 620
  встраивание в HTML-документ 607
  жизненный цикл 608
  идентификатор this 618
  методы класса Applet 603—606
  обработка событий 615
Архивы 621
  формат JAR 622
Ассоциативный массив
  %ENV 428
  @EXPORT 443
  @EXPORT_OK 443
Атрибуты обработки событий 329, 359

Б

Безопасность компьютера 573
Безопасные элементы управления ActiveX
  в сценарии 577
  при инициализации 577
  цифровая подпись 574
Библиотека 442
Блок 438

В

Виртуальная машина Java 598
Встраивание сценария 327
Выражения JavaScript 335
  арифметические 336
  логические 337
  строковые 338

Г

Графический интерфейс пользователя 612
AWT (Abstract Windowing Toolkit) 612
Swing 612

Д

Дескриптор файла 422
Дополнительная информация о пути 413

З

Заголовок
запроса 403
ответа 417
Загружаемые шрифты 553
свойство @fontdef 554
Запрос клиента 403, 412

И

Интерфейс 602
Источники данных
MSHTML 542
RDS.DataControl 538
TDC 537

К

Карта-изображение 194
клиентский вариант 198
серверный вариант 198
типы областей 203
формат CERN 202
формат NCSA 203
Каскадные таблицы стилей 451
вес 460
внедрение 453
группирование селекторов 454
импорт 453
класс 456
контекстный селектор 458
наследование свойств 455
определение 451
правило 451
приоритетность 460
псевдоклассы 458
псевдоклассы связей 459
псевдоэлементы 458
свойство filter 520
свойство, значение 451
связывание 452
селектор 451, 455
селектор-идентификатор 457
синтаксис JavaScript 543
специфичность селектора 461
Классы 598
внутренние классы 618
Ключевые слова JavaScript
this 333, 362
Ключевые слова VBScript
ByVal 385
Preserve 375

Л

Литералы VBScript
булевы 372
даты и времени 372
строковые 372
числовые 371
Лицензирование 579
во время выполнения 579
во время разработки 579

М

Массив 422
@_ 440
Массивы VBScript
динамические 374
статические 373
Медиа-тип 404
Менеджер компоновки
BorderLayout 614
CardLayout 615
FlowLayout 615
GridBagLayout 615
GridLayout 615
Метод пересылки данных
GET 403, 412
POST 403, 413
Методы апплета 598
destroy() 609
init() 608

Методы апплета *(прод.)*
paint() 609
repaint() 609
start() 609
stop() 609
update() 609
метод main() 601
Методы объектов
getAttribute() 495
removeAttribute() 495
scrollIntoView() 496
setAttribute() 495
Методы переходов
apply() 531
play() 531
stop() 531
Методы фильтра light
addAmbient() 527
addCone() 528
addPoint() 528
Модель форматирования 462
абсолютные единицы 466
блок содержимого 462
блоковый элемент 463
встроенный элемент 465
граница 462
относительные единицы 466
отступ 462
поле 462
Модуль 442

Н

Наборы объектной модели
all 491
children 491, 507
filters 522
style 493, 495
метод tags() 494
свойство length 492

О

Обработка событий 615
Обработка событий элемента управления ActiveX 562
обработчик события в VBScript 562
Объектная модель DHTML 490
объект document 491
свойства CSS 493
свойства и методы объектов 494
Объектная модель документа 488
отношения "родства" 506
реализация 490
структура документа 489
структурный изоморфизм 489
узел 503
Объектная модель сценария 543
конструктор Layer() 551
метод load() 551
методы слоя 550
набор layers 548
объект layer 547
свойства слоя 549
слой 547
Объекты 598
Объекты HTML
Object 561
Объекты JavaScript
Array 339
Data 341
document 356
event 362
Form 357
Frame 355
history 357
location 357
Math 342
String 343
window 352
иерархия 351
Объекты VBScript
FileSystemObjects 391
TextStream 391
Оператор
foreach 425
if ...elsif...else 436
Операторы JavaScript
break 347, 348
continue 348
строковые 338
for 347
for...in 349
function 361
if 345
switch 346
var 335
while 348

Операторы JavaScript *(прод.)*
with 350
арифметические 336
логические 338
поразрядных действий 339
присваивания, = 335
присваивания, сокращенные формы 337
сравнения 337
условный 338
Операторы VBScript
Const 375
Dim 373
Do Until 380
Do While 379
Exit Do 382
Exit For 382
For Each...Next 381
For...Next 380
If...Then 377
If...Then...Else 378
ReDim 374
Select Case 378
Set 391
With 397
арифметические 376
конкатенации 377
логические 377
сравнения 376
Ответ сервера 416
Отображение списков 486
Отражение страницы 324

П

Пакет 438
Пакеты 598
Параметр
ACCESSKEY 561
ALIGN 78, 116, 124, 128
ALT 86, 207
BACKGROUND 76, 140
BGCOLOR 76
BORDER 85, 120, 181
BORDERCOLOR 139, 181
BORDERCOLORDARK 140
BORDERCOLORLIGHT 139
CELLPADDING 122
CELLSPACING 121
CLASS 456
CLASSID 559
CLEAR 83, 126
CLSID 557
CODEBASE 557, 560
COLS 143, 157
COLSPAN 130
COLSPEC 137
COMPACT 58
COORDS 206
DATAFLD 535, 542
DATAFORMATAS 535
DATAPAGESIZE 536
DATASRC 535
FRAME 141
FRAMEBORDER 181
FRAMESPACING 184
HEIGHT 83, 123, 129, 185, 560
HREF 206
HSPACE 84, 185
ID 457
ISMAP 201
LOWSRC 88
MARGINHEIGHT 161
MARGINWIDTH 161
NAME 160
NOHREF 206
NORESIZE 161
NOWRAP 129
ROWS 157
ROWSPAN 130
RULES 141
SCROLLING 161
SHAPE 205
SPAN 137
SRC 78, 160
START 63
TARGET 161, 168, 207
USEMAP 204
VALIGN 116, 128
VSPACE 84, 185
WIDTH 83, 123, 129, 185, 560
Передача параметров
по значению 440
по ссылке 440
Переменная
PATH_INFO 413
QUERY_STRING 412

Переменные в языке Perl
ассоциативные массивы 424
массивы 422
скалярные переменные 418
арифметические и логические операции 419
операции сравнения 420
Переменные среды CGI 427
Переход 519
Платформа Java 598
Подключаемый программный модуль
Crescendo 234
Cthugha 258
EchoSpeech 233
LiveAudio 226
RealAudio 235
ToolVox 233
TrueSpeech 233
Подпрограммы 438
Подтипы типа Variant 370
Empty 371
Error 371
Null 371
Позиционирование
абсолютное 479
относительное 481
статическое 481, 482
Приложения 599
Программа
AudioActive 267
AudioCatalyst 269
AudioGrabber 266
BladeEnc 270
CD2WAV 266
CDCopy 266
CDDA 266
CDex 266
FrameGang 187
Frame-It 190
GIF Construction Set 100, 102
K-Jofol 275
Map THIS! 217
MapEdit 212
Microsoft ActiveMovie 231
MP3 Compressor 267
MP3 Producer 267
RealPlayer 238
VideoCraft GIF Animator 100
Winamp 252
WinDAC 266
XingMP3 Encoder 269
Программный объект 321
взаимодействие 323
инкапсуляция 322
методы 321
свойства 321
Процедуры JavaScript 330
Процедуры VBScript
Function 384
Sub 383
передача параметров 385

Р

Регулярное выражение 430
Редактор FrontPage 565

С

Свойства визуализации
clip 483
display 483
overflow 484
visibility 482
z-index 485
Свойства объекта event
altKey 501
button 502
cancelBubble 499, 501
clientX 501
clientY 501
ctrlKey 501
fromElement 502
offsetX 501
offsetY 501
returnValue 501
screenX 501
screenY 501
shiftKey 501
srcElement 500
toElement 502
x 501
y 501
Свойства объекта Node
Child 589
Children 589

Свойства объекта Node *(прод.)*
FirstSibling 589
ImageList 592
LastSibling 589
Next 589
Parent 589
Previous 589
Root 589
Свойства объектов
innerHTML 496
innerText 496
outerHTML 496
outerText 496
sourceIndex 495
style 495
tagName 492
Свойства позиционирования
height 481
left 480
position 479
top 480
width 481
Свойства фильтров
add 524
color 523
direction 524
enabled 523
strength 523
Свойства форматирования
граница 477
отступ 477
поле 476
Свойства шрифтов
@font-face 471
font 471
font-family 468
font-size 470
font-style 469
font-variant 469
font-weight 470
Свойства элемента управления TabStrip
MultiRow 582
Style 582
Связывание данных 533
архитектура привязки 533
объект-источник 533
объект-потребитель 533
программа-посредник 533
табличный повторитель 533
Событие 324, 497
объект event 497, 500
процедура обработки события 324
цикл жизни 497
События элементов 499
Список
вложенный 68
маркированный 56
нумерованный 61
определений 65
Стандартные функции VBScript
CreateObject 391
InputBox 387
MsgBox 388
времени и даты 386
математические 386
строковые 387
Строка запроса 412

Т

Таблица 114
Текстовые свойства
letter-spacing 474
line-height 476
text-align 475
text-decoration 475
text-indent 475
text-transform 475
vertical-align 475
Технология COM 556
Типы данных JavaScript
строковый 334
булевый 334
вещественный 334
целый 334
Типы данных VBScript
Variant 370
Тэг
<!-- 46
<!DOCTYPE> 19
<A HREF> 51
<A NAME> 51
<A> 87
<ABBR> 27
<ACRONYM> 27
<ADDRESS> 48
<APPLET> 607

Тэг *(прод.)*
<AREA> 205
<B> 31
<BASE> 21, 168
<BASEFONT> 37
<BDO> 55
<BIG> 32
<BLINK> 33
<BLOCKQUOTE> 47
<BODY> 24

 40, 83
<CAPTION> 116, 118
<CENTER> 46, 124
<CITE> 27
<CODE> 28
<COL> 137
<COLGROUP> 137
<DD> 65
<DEL> 28
<DFN> 28
<DIR> 67
<DIV> 45
<DL> 65
<DT> 65
<EM> 29
<EMBED> 227
<FIELDSET> 55
<FONT> 34
<FORM> 403
<FRAME> 157, 159
<FRAMESET> 157
<HEAD> 19
<HR> 43
<HTML> 18
<Hx> 42
<I> 32
<IFRAME> 185
<ILAYER> 55
<IMG> 201
<INPUT> 405
<INS> 29
<ISINDEX> 55
<KBD> 29
<KEYGEN> 55
<LAYER> 55
<LEGEND> 55
<LH> 58
<LI> 57, 61
<LINK> 21
<LISTING> 45
<MAP> 205
<MARQUEE> 55
<MENU> 67
<META> 22
<MULTICOL> 55
<NEXTID> 55
<NOBR> 42
<NOFRAMES> 162
<OBJECT> 557, 559, 607
<OL> 61
<P> 39
<PARAM> 557, 561
<PLAINTEXT> 45
<PRE> 44, 114
<Q> 29
<S> 32
<SAMP> 29
<SELECT> 407
<SMALL> 32
<SOUND> 55
<SPACER> 55
<SPAN> 33
<STRIKE> 32
<SUB> 33
<SUP> 33
<TABLE> 116
<TBODY> 140
<TD> 116
<TEXTAREA> 408
<TFOOT> 140
<TH> 116
<THEAD> 140
<TITLE> 19
<TR> 116
<TT> 32
<U> 32
<UL> 57
<VAR> 30
<WBR> 42
<XMP> 45
--> 46

Ф

Фильтр
динамический 519
статический 519

Фильтры и переходы 519
alpha 527
blendTrans 530
dropShadow 526
light 527
revealTrans 531
wave 525
свойство filter 520
Формат
AAC 277
AIFF 226
AU 226
GIF 90, 197
GIF87a 91
GIF89a 91
JPG 96, 197
LQT 279
MIDI 226
MP3 250
MP4 281
MPEG 249
PAC 280
PCM 233
PCX 74
PNG 213
RA 239
RAM 240
RPM 240
VQF 274
WAV 226
WMA 282
Формы 402
Фрейм 148
Функция
close 448
hex 433
import 443
join 424
keys 425
local 439
my 439
open 448
pack 433
pop 423
print 416
push 424
require 442
return 439
reverse 424
shift 423
sort 424
splice 423
split 433
unshift 423
use 443
values 425

Ц

Цветовые свойства
background 474
background-attachment 472
background-color 472
background-image 472
background-position 473
background-repeat 472
color 472

Э

Элемент управления ImageList 588
Элемент управления TabStrip 582
добавление вкладок 585
набор Tabs 584
область клиента 582
объект Tab 584
Элемент управления TreeView 588
набор Nodes 589
объект Node 589
узел 588
узлы-братья 589
Элементы управления ActiveX 555
приложение-контейнер 556
уникальные идентификационные номера 557

Я

Язык Perl 415