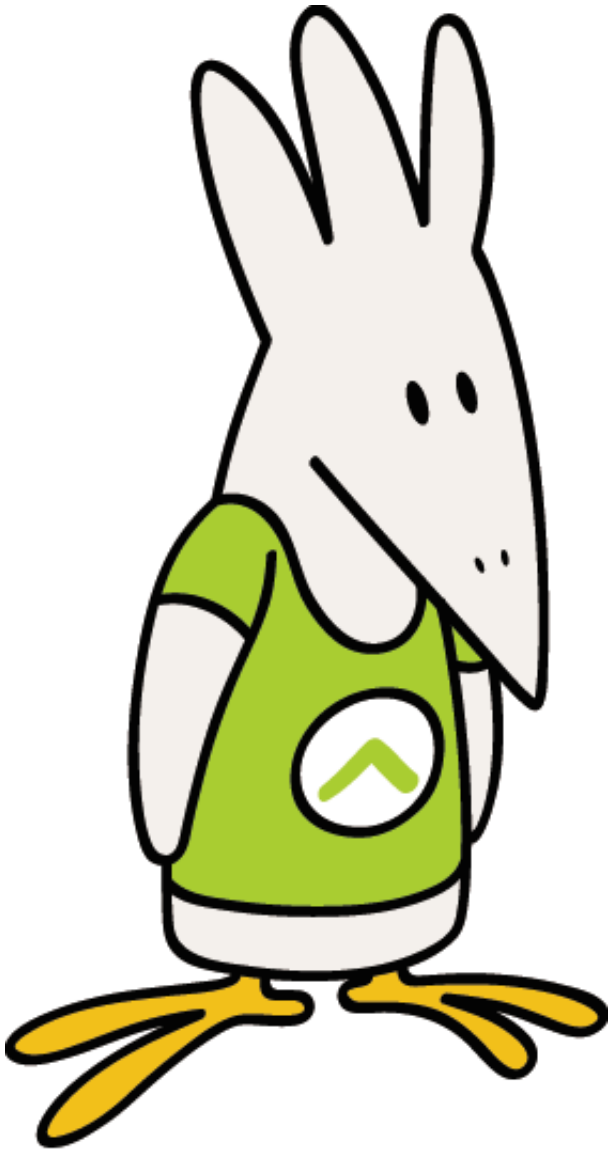

parser

Оглавление

Лицензионное соглашение

Настоящее Лицензионное соглашение (далее «Соглашение») является юридическим договором, заключаемым между Пользователем (физическим или юридическим лицом) и ЗАО «Группа компаний Артемия Лебедева» (далее «Группа компаний Артемия Лебедева») относительно программы для ЭВМ PARSER (далее «Программа» или «Программное обеспечение»), которая может включать в себя компоненты Программного обеспечения, как записанного на соответствующих носителях, так и распространяемого по каналам сети Интернет, и сопроводительную документацию в печатном и электронном виде. Устанавливая, копируя или иным образом используя Программное обеспечение, Пользователь тем самым принимает условия настоящего соглашения. Если вы не принимаете условий данного соглашения, не устанавливайте и не используйте данную Программу. Программное обеспечение защищено законами и международными соглашениями об авторских правах.

1. ПРЕДОСТАВЛЕНИЕ ЛИЦЕНЗИИ

Конечному Пользователю предоставляется неисключительная непередаваемая лицензия на использование Программы при соблюдении приведенных ниже условий.

Установка и использование.

«Группа компаний Артемия Лебедева» предоставляет право установить и использовать Программу.

Резервное копирование.

Разрешается делать копии Программы так, как этого требуют принятые процедуры архивации и резервного копирования.

Распространение.

Пользователь имеет право тиражировать, распространять, передавать во временное пользование Программу другим лицам без специального разрешения ЗАО «Группа компаний Артемия Лебедева» при условии выполнения прочих ограничений данного лицензионного соглашения.

2. ПРОЧИЕ ПРАВА И ОГРАНИЧЕНИЯ

Сохранение уведомлений об авторских правах.

Запрещается удалять или изменять какие-либо уведомления об авторских правах на всех копиях Программы и в сопроводительных материалах.

Запрет на модификацию.

Запрещается модифицировать, адаптировать, переводить на другие языки, менять структуру, декомпилировать, дизассемблировать Программу.

3. ПРЕКРАЩЕНИЕ ДЕЙСТВИЯ СОГЛАШЕНИЯ

Без ущерба для чьих-либо прав ЗАО «Группа компаний Артемия Лебедева» может прекратить действие настоящего соглашения при несоблюдении его условий и ограничений.

При прекращении действия соглашения вы обязаны уничтожить все имеющиеся копии Программы.

4. АВТОРСКОЕ ПРАВО

Все исключительные права, включая авторские права на Программу, принадлежат ЗАО «Группа компаний Артемия Лебедева» и лично разработчикам, и защищены законами и международными соглашениями об авторских правах.

5. ОТКАЗ ОТ ГАРАНТИЙ

ЗАО «Группа компаний Артемия Лебедева» явно отказывается от предоставления каких-либо гарантий по отношению к Программе. Программа и вся относящаяся к ней документация предоставляется «как есть», без какой-либо гарантии, явной или подразумеваемой, включая все без исключения подразумеваемые гарантии товарности или пригодности для какой-либо определенной цели. Пользователь принимает на себя все риски, связанные с использованием или качеством функционирования Программы.

6. ОТВЕТСТВЕННОСТЬ

ЗАО «Группа компаний Артемия Лебедева» отказывается нести ответственность за какой-либо ущерб (включая все без исключения случаи потери прибыли, прерывания деловой активности, потери деловой информации или любые убытки), связанный с использованием или невозможностью использования Программы, а также с предоставлением или невозможностью предоставления услуг по технической поддержке, даже если представитель ЗАО «Группа компаний Артемия Лебедева» был извещен заранее о возможности таких потерь.

При возникновении вопросов по данному соглашению обращайтесь в ЗАО «Группа компаний Артемия Лебедева»: legal@artlebedev.ru

Введение

Кому и зачем нужен Парсер?

В языке разметки гипертекстовых документов HTML нет переменных. Не предусмотрены в нем условные операторы, циклы, процедуры и другие средства, свойственные языкам программирования. Это делает работу HTML-кодера относительно простой, но очень увеличивает ее объем. Так, публикуя на каждой странице сайта номер контактного телефона, мы обеспечиваем себе необременительное занятие для рук в случае его изменения. При появлении на сайте нового раздела приходится обновлять списки навигационных ссылок во всех остальных разделах. Если все страницы сайта имеют единообразную структуру, например композиционно состоят из верхнего колонтитула, основного текста и нижнего колонтитула, то для ее изменения потребуется основательно перелопатить весь сайт.

Чем сложнее устроены страницы сайта, чем чаще изменяется содержимое, тем больше работы приходится выполнять HTML-кодерам. Обновление сайта становится весьма трудоемким мероприятием, а количество ошибок в коде и опечаток в тексте страниц возрастает. Эту проблему часто пытаются решить методом, который одна колоритная руководительница проекта обозначала термином «замена ручного труда женским». Однако рост феминистических настроений прежде всего и в несколько меньшей степени приведенные выше практические соображения лишают подобный подход всяческой привлекательности.

В Студии Лебедева (http://www.design.ru/) почти каждый день завершаются те или иные проекты. Неудивительно, что необходимость программы, которая взяла бы на себя большую часть рутинной работы по кодированию страниц, достаточно быстро стала для нас очевидной. Конечно, существовало и существует немало языков программирования, предназначенных для написания скриптов, генерирующих HTML-страницы. Но мы хотели получить решение, которое дало бы возможность автоматизировать работу HTML-кодера, а не заменить последнего программистом. Поэтому в 1997 г. мы сами взялись за дело и разработали первую версию Парсера. Затем на протяжении нескольких лет Парсер совершенствовался и развивался, а мы успешно использовали его при создании сайтов. Теперь мы дарим такую возможность вам.

Парсер автоматизирует создание и обновление сайтов, страницы которых схожи по оформлению, содержанию или структуре. Применение Парсера позволяет HTML-кодеру единожды описать повторяющиеся элементы информационного наполнения и разметки страниц, а затем ссылаться на них в коде каждой страницы сайта. Средствами Парсера можно дать описание структуры, а затем использовать его для формирования единообразно устроенных страниц. При этом если мы изменяем описание структуры или элемента разметки, то в дальнейшем все страницы генерируются с учетом внесенных корректив.

Парсер пригоден для решения большинства задач, которые обычно требуют программирования CGI-скриптов, например, для формирования страниц с результатами запросов к реляционным базам данных. В Парсере предусмотрены средства для работы с полями форм, строками запроса, переменными окружения веб-сервера. Немаловажно, что использование этих возможностей не требует высокой квалификации в области программирования. Задача данного руководства — помочь HTML-кодеру научиться создавать с помощью Парсера интерактивные веб-сайты, а также снабдить его необходимой справочной информацией.

Как это работает?

Парсер устанавливают на сервере. Веб-сервер конфигурируют таким образом, что HTML-файлы до пересылки их браузеру (или любому другому клиенту) обрабатываются Парсером.

Если в тексте файла обнаруживаются *операторы* Парсера, последний выполняет их *разбор* и заменяет *вычисленными значениями*. Например, вызов оператора `^uri[]` будет заменен адресом обрабатываемой страницы, вызов `^date[]` — текущей датой, вызов `^macro[vsyo_na_svete]` — результатом разбора *макроса* `vsyo_na_svete`, что бы он в себе ни таил. Остальной текст, в том числе тэги языка HTML, остается без изменений.

Может показаться, что по сути дела Парсер выполняет поиск и замену в пересылаемых веб-сервером HTML-файлах, только операция замены устроена сложнее, чем в текстовых процессорах. Это взгляд очень упрощенный, граничащий с ересью, но доля истины в нем есть, поскольку в элементарных случаях именно это и происходит. Далее, переходя от простого к сложному и от правил к исключениям из них, мы увидим, что для многих операторов это не совсем так, а для *шаблонов страниц*, в которых главенствующую роль играют не операторы, а макросы, совсем не так.

Как научиться делать сайты с помощью Парсера?

Прежде всего мы рекомендуем внимательно прочитать Главу 1. В ней разъясняются основные понятия: *оператор, макрос, шаблон страницы, таблица, уровень обработки*. Приводимые в этой главе примеры лишены практического смысла, зато иллюстративны и не загромождены лишним кодом. Тем не менее каждый пример может быть приведен в действие, если это необходимо для усиления педагогического эффекта.

В Главе 2 на примере несложного сайта показано, каким образом Парсер используется для описания структуры страниц и решения некоторых распространенных практических задач. Пожалуйста, разберитесь в этом примере, иначе в дальнейшем вам придется изобрести велосипед и не один.

Глава 3 систематически описывает все операторы Парсера. Она будет служить вам справочником при составлении кода страниц.

Глава 4 обобщает опыт применения Парсера и содержит рекомендации по составлению внятного и работоспособного кода.

Приложения содержат сведения, необходимые для установки Парсера, и другую справочную информацию.

О том, как с помощью Парсера решаются всевозможные частные задачи, читайте в разделе FAQ.

Прежде чем продолжить изучение Парсера, сделаем одно важное замечание относительно способа подачи материала, принятого в данном руководстве. Как вы убедитесь, в Парсере одни и те же конструкции могут давать разный эффект в зависимости от того, каким образом вы их употребляете. Говоря о таких конструкциях, мы не будем стараться сразу, одной общей формулировкой охватить все возможные варианты, так как их перечисление не прибавит ясности, но, напротив, загромоздит текст. Вместо этого мы сформулируем общее правило, а затем постепенно опишем возможные исключения.

Кодер, помни!

Парсер чувствителен к регистру. Например, вызовы `^var[VSYO_NA_SVETE]` и `^var[vsyo_na_svete]` будут восприниматься как разные.

Парсер учитывает пробельные символы. В отличие от традиционных языков программирования, Парсер не всегда игнорирует пробелы, символы табуляции и символы перевода строки. Например, вызов

```
^macro[show_news; ^date[];
          2
]
```

в качестве значения первого аргумента передаст макросу **show_news** текущую дату, предваряемую лишним пробелом, а в качестве значения второго аргумента — строку 2, окруженную символами перевода строки.

Условные обозначения

оператор	Термины
Замечание...	Сведения, на которые рекомендуется обратить особое внимание
Внимание!..	Предостережения
Пример	Примеры
`mnogo_deneg`	Цитируемый код
`^menu[<H1>^item[id]</H1>]`	Листинги
`>`	Символ табуляции

1. Основные конструкции языка

1.1. Веб-страницы, HTML-файлы и шаблоны страниц

Прежде чем приступить к подробному описанию конструкций Парсера, во избежание путаницы уточним значения нескольких важных терминов.

Веб-страницей или просто *страницей* будем называть результат интерпретации браузером переданного ему HTML-кода. Иными словами, страница — это то, что видит *посетитель сайта*.

HTML-файлом будем называть текстовый файл, содержащий HTML-код.

Шаблоном страницы или просто *шаблоном* будем называть HTML-файл, код которого содержит инструкции Парсера.

Проверим эту терминологию в действии. Итак, в каталоге сайта находятся *HTML-файлы*. Мы вписываем в них инструкции Парсера и получаем *шаблоны страниц*. Прежде чем переслать HTML-файл в браузер, веб-сервер передает его для обработки Парсеру. При этом обработка конструкций Парсера, размещенных в *шаблонах страниц*, порождает соответствующий HTML-код. В итоге посетитель сайта созерцает *страницу*. Похоже, работает.

1.2. Операторы

Оператором называется конструкция следующего вида:

```
^имя_оператора[аргумент1;аргумент2;...аргументN]
```

В качестве *значений аргументов* в оператор могут быть подставлены произвольные фрагменты HTML-кода, в том числе занимающие в файле несколько строк. Некоторые операторы не имеют аргументов. Набор доступных операторов определяется используемой версией Парсера.

Для того чтобы отделить аргументы друг от друга, используется символ `;` (точка с запятой). Если внутри аргумента встречается точка с запятой или символ `^` («птичка»), перед ними помещают символ `^`.

При обработке файла Парсером оператор заменяется фрагментом, который получается в результате его разбора. Этот фрагмент мы далее будем называть *значением оператора* и говорить, что оператор *возвращает* то или иное значение, *вычисляемое* Парсером.

Пример. Создадим страницу, отображающую собственный URI. Для этого поместим в ее текст используемый без параметров оператор `uri`, который возвращает URI текущей страницы.

```
<HTML>
<HEAD><TITLE>Пример</TITLE></HEAD>
<BODY>Вот мой URI: ^uri[]</BODY>
</HTML>
```

Если URI страницы **/parser/doc/examples/eg1.html**, то обработка приведенного выше кода Парсером даст следующий результат:

```
<HTML>
<HEAD><TITLE>Пример</TITLE></HEAD>
<BODY>Вот мой URI: /parser/doc/examples/eg1.html</BODY>
</HTML>
```

Парсер действительно немногим отличался бы от функции поиска и замены, если бы в аргументах операторов не могли содержаться другие операторы. В общем случае, если аргументы оператора заключают в себе другие операторы, Парсер сначала вычисляет их значения и подставляет в аргументы, которые затем использует при вычислении значения оператора. Глубина вложения операторов при этом не ограничена.

Из приведенного выше правила есть ряд исключений. Так, аргумент оператора **rem**, который используется для оформления комментариев и всегда возвращает пустую строку, не вычисляется. У оператора вызова макроса **macro** вычисляется только первый аргумент, остальные же передаются в макрос как есть. Аргументы операторов **for** и **while** могут быть вычислены несколько раз. Существуют и другие исключения, они подробно рассмотрены в описаниях операторов.

Пример. Модифицируем предыдущий пример таким образом, чтобы на странице отображалось только имя файла, а не URI. Для этого используем оператор **name**, который принимает в качестве аргумента путь к файлу, а возвращает только его имя.

```
<HTML>
<HEAD><TITLE>Пример</TITLE></HEAD>
<BODY>Вот мое имя: ^name[^uri[]]</BODY>
</HTML>
```

Если URI страницы **/parser/doc/examples/eg2.html**, то обработка приведенного выше кода Парсером даст следующий результат:

```
<HTML>
<HEAD><TITLE>Пример</TITLE></HEAD>
<BODY>Вот мое имя: eg2.html</BODY>
</HTML>
```

Результатом обработки оператора может быть не только возвращаемое им значение. Некоторые операторы дают *побочный эффект*, изменяя условия обработки других операторов и макросов. Например, оператор **var** с двумя аргументами всегда возвращает пустую строку. При этом он помещает в *переменную*, имя которой задается первым аргументом, значение, задаваемое вторым аргументом. В дальнейшем это значение может быть возвращено оператором **var** с одним аргументом и использовано, предположим, в качестве аргумента какого-нибудь другого оператора.

Пример. Результатом обработки приведенного ниже кода будет страница, содержащая ссылку на саму себя.

```
<HTML>
<HEAD><TITLE>Пример</TITLE></HEAD>
<BODY>
^var[my_URI;^uri[]]
<A HREF=^var[my_URI]>Вот я: ^var[my_URI]</A>
</BODY>
</HTML>
```

Внимание! Если аргумент реально не используется при вычислении значения оператора, то Парсер может и не вычислить его. Это повышает производительность системы, но может подвести вас, если вы поместили в этот аргумент оператор, чтобы воспользоваться его побочным эффектом.

Замечание. Если в аргументе оператора много точек с запятой, то простановка перед каждой из них символа ^ может оказаться чересчур трудоемким делом. Такой аргумент можно заключить в обратные кавычки. При этом необходимо, чтобы первая обратная кавычка следовала непосредственно за квадратной скобкой, открывающей список аргументов, или за точкой с запятой, отделяющей аргумент от предыдущего. Вторая обратная кавычка должна непосредственно предшествовать квадратной скобке, закрывающей список аргументов, или точке с запятой, отделяющей аргумент от следующего за ним. В противном случае обратные кавычки будут интерпретироваться как символы, являющиеся частью самого аргумента. Выглядит это примерно так.

```
^macro[nav_link;назад;-1;`&lt;&lt;&lt;&lt;&lt;`]
^macro[nav_link;вперед;+1;`&gt;&gt;&gt;&gt;&gt;`]
```

1.3. Макросы

1.3.1. Общее представление о макросах

Макросы применяются для описания элементов, повторяющихся на одной или нескольких страницах сайта. Это могут быть стандартные надписи, колонтитулы, ссылки, меню и т. п. Предпочитая строгости определений их внятность, скажем, что макрос — это фрагмент текста, который вставляется Парсером в обрабатываемый шаблон страницы. Текст макроса оказывается там, куда HTML-кодер поставил соответствующий *вызов*. В этом смысле макрос напоминает оператор. Принципиальное отличие состоит в том, что HTML-кодер сам создает макросы, необходимые ему для работы. В тексте макроса можно использовать операторы и вызывать другие макросы. Кроме того, в описании макроса могут быть указаны поименованные *аргументы*. При вызове макроса HTML-кодер может указать *значения* каждого из них. Тогда, встретив в тексте макроса имя аргумента, предваряемое символом **$** (знак доллара), Парсер заменит такое сочетание значением этого аргумента.

Замечание. Немного забегая вперед, поясним, что в Парсере предусмотрены два принципиально различных способа вызова макросов. Во-первых, макрос можно вызвать с помощью оператора **macro** (см. п. 1.3.4). Во-вторых, если в шаблоне описаны макросы с именами **autoexec** или **main**, они будет вызваны Парсером автоматически (см. п. 1.3.5).

1.3.2. Наглядный пример использования макроса

Прежде чем в деталях описать правила составления и вызова макросов, рассмотрим один несложный пример.

Пример. Используем макрос для создания страницы, на которой опубликован контактный телефон. У страницы должны быть две языковые версии: русская и английская. Перед номером телефона располагается вводная фраза на русском или на английском языке соответственно. Оформление вводной фразы и номера телефона в обеих версиях страницы должно быть строго одинаковым. Поскольку в ближайшем будущем количество языковых версий может увеличиться, требуется, чтобы для изменения номера телефона, его оформления или оформления вводной фразы не нужно было вручную редактировать все языковые версии.

Вынесем в макрос все, что может измениться при изменении номера телефона или его оформления. Для этого создадим в корневом каталоге сайта файл **_macro.cfg** и наберем в нем следующий текст:

_macro.cfg

```
@my_phone[text_before_phone] выводит телефон с "преамбулой"
<B><I>$text_before_phone</I> +7 (095) 229-85-23</B>
```

Это макрос. Первая строка представляет собой *заголовок макроса*. Парсер не будет вставлять его в шаблон страницы, зато в нем указано *имя макроса*, **my_phone**, и задан список его аргументов. В данном случае аргумент один — **text_before_phone**. Мы воспользуемся им для передачи макросу вводной фразы.

Теперь создадим два шаблона для русской и английской версий страницы.

russian.html

```
<HTML>
<HEAD><TITLE>Как с нами связаться</TITLE></HEAD>
<BODY>
^macro[my_phone;Наш номер телефона:]
</BODY>
</HTML>
```

english.html

```
<HTML>
<HEAD><TITLE>Contacts</TITLE></HEAD>
<BODY>
^macro[my_phone;Our phone number:]
</BODY>
</HTML>
```

В обоих шаблонах третья строка содержит вызов макроса **my_phone**. Для вызова используется оператор **macro**. В качестве первого аргумента этому оператору всегда передается имя вызываемого макроса. Последующие аргументы, начиная со второго, передаются вызываемому макросу. Иными словами, второй аргумент оператора **macro** — это первый аргумент макроса и т. д.

В результате обработки Парсером приведенных выше шаблонов получим следующие страницы.

russian.html

```
<HTML>
<HEAD><TITLE>Как с нами связаться</TITLE></HEAD>
<BODY>
<B><I>Наш номер телефона:</I> +7 (095) 229-85-23</B>
</BODY>
</HTML>
```

english.html

```
<HTML>
<HEAD><TITLE>Contacts</TITLE></HEAD>
<BODY>
<B><I>Our phone number:</I> +7 (095) 229-85-23</B>
</BODY>
</HTML>
```

1.3.3. Правила записи макросов

Макрос представляет собой конструкцию следующего вида:

```
@имя_макроса[аргумент1;аргумент2;...аргументN] комментарий
текст_макроса
```

Макрос начинается однострочным *заголовком*, первый символ которого (**@**, т. н. «собака») обязательно должен располагаться в первой позиции строки. *Имя макроса* может состоять из латинских букв, цифр и символов подчеркивания, но не должно начинаться с цифры. Такие же требования предъявляются к именам аргументов.

За списком аргументов можно поместить произвольный (но умещающийся в той же строке) *комментарий*. Количество строк, занимаемых текстом макроса, не ограничено.

Символ **$** (знак доллара) и следующее непосредственно за ним имя аргумента макроса заменяются переданным при вызове значением этого аргумента. Если же знак доллара является частью текста макроса, его следует удвоить. Например, так: **`ИТОГО: $$$total_cost`** (первые два знака доллара будут преобразованы в один, а третий вместе с именем аргумента будет заменен значением аргумента).

Описание макроса можно разместить

- в том шаблоне, откуда он вызывается;
- в файле **`_macro.cfg`**, местонахождение которого — корневой каталог веб-сервера;
- в любом другом файле.

Макросы, описанные в файле **`_macro.cfg`**, могут быть вызваны в любом шаблоне. Для того чтобы сделать доступными макросы, описанные в другом файле (но не в **`_macro.cfg`**), применяют оператор **`macro_use`** (см. п. 3.2.3). В качестве аргумента этому оператору передают путь к файлу (или к файлам) с описаниями макросов.

Внимание! Если в шаблоне встречаются определения макросов, то весь не относящийся к ним код Парсером игнорируется.

1.3.4. Вызов макросов с помощью оператора macro

Вызов макроса с помощью оператора **macro** мы уже продемонстрировали в приведенном выше примере.

```
^macro[имя_макроса;аргумент1;аргумент2;...аргументN]
```

В качестве первого аргумента указывается имя вызываемого макроса. Затем указываются значения аргументов, передаваемых макросу.

Внимание! Передаваемые макросу значения аргументов перед их вставкой в текст макроса не обрабатываются Парсером.

1.3.5. Шаблоны с описаниями макросов. Макрос main

Шаблоны, содержащие описания макросов, обрабатываются Парсером особым образом. Код, не входящий ни в один из макросов, игнорируется. Если в шаблоне описан макрос с именем **main**, он вызывается автоматически — этим обработка шаблона исчерпывается. В том случае, если макрос **main** обращается к другим макросам, описанным в том же файле или в других файлах, они будут вызваны.

Для чего используется такой механизм вызова макросов? С помощью операторов и макросов можно вставлять в код страниц разнообразные стандартные фрагменты. А что делать, если необходимо описать повторяющееся структурное решение? Допустим, все страницы сайта состоят из верхнего колонтитула, блока содержания и нижнего колонтитула. Теперь потребовалось расположить информацию из нижнего колонтитула слева от блока содержания, а не под ним. Иными словами, показать то же самое, но в другом месте. Свести подобную задачу к переписыванию вынесенных в отдельный файл общих «вставок» будет нелегко, особенно если нижние колонтитулы страниц различаются. Для ее решения лучше использовать механизм автоматического вызова макросов.

Пример. Подготовим несколько страниц, композиционно состоящих из верхнего колонтитула, блока содержания и нижнего колонтитула. Верхний колонтитул на всех страницах выглядит одинаково, а нижний колонтитул на каждой странице свой. Затем, приложив минимум усилий, мы добьемся того, что информация из нижнего колонтитула на каждой странице будет отображаться в левом поле.

Сначала опишем в файле **_macro.cfg** макросы, общие для всех страниц. Макрос **html** задает общую единообразную структуру страниц, а макрос **fixed** формирует общий верхний колонтитул.

_macro.cfg

```
@html[title;body] основной код страницы (спец. колонтитул снизу)
<HTML>
<HEAD><TITLE>$title</TITLE></HEAD>
<BODY>
   <P>^macro[fixed]</P>
   $body
   <P>^macro[custom]</P>
</BODY>
</HTML>
@fixed[] общий верхний колонтитул
   <I>Верхний колонтитул, общий для всех страниц</I>
```

Внимательный читатель заметил, что макрос **custom**, к которому обращается макрос **html**, в файле **_macro.cfg** не описан. Макрос с этим именем мы опишем в шаблоне каждой из страниц. Он будет использоваться для формирования уникального для каждой страницы нижнего колонтитула (а затем левого поля).

В отдельных HTML-файлах создадим несколько шаблонов страниц.

gray_kid.html

```
@main[] стишок про козлика
   ^macro[html;
          Серенький козлик;
          Жил-был у бабушки серенький козлик,<br>
          Вот как, вот как, серый козел.
   ]
@custom[] колонтитул для стишка про козлика
   <P><I>Нижний колонтитул, индивидуальный для каждой страницы</I></P
>
   <P>Телефон отдела копытных: 123-45-67</P>
```

kind_beetle.html

```
@main[] стишок про жука
   ^macro[html;
          Добрый жук;
          Встаньте, дети, встаньте в круг,<br>
          Жил на свете добрый жук.
   ]
@custom[] колонтитул для стишка про жука
   <P><I>Нижний колонтитул, индивидуальный для каждой страницы</I></P
>
   <P>Телефон отдела насекомых: 765-43-21</P>
```

Теперь вернемся к файлу **_macro.cfg** и перепишем макрос **html** таким образом, что содержимое нижнего колонтитула на всех страницах будет появляться слева от блока содержания.

_macro.cfg

```
@html[title;body] основной код страницы (спец. колонтитул слева)
<HTML>
<HEAD><TITLE>$title</TITLE></HEAD>
<BODY>
   <P>^macro[fixed]</P>
   <TABLE BORDER=1><TR>
      <TD>Теперь на всех страницах он слева!^macro[custom]</TD>
      <TD>$body</TD>
   </TR></TABLE>
</BODY>
</HTML>
```

Обратите внимание, сколько бы ни было страниц и каковы бы ни были нижние колонтитулы, исправлять все равно пришлось бы только макрос **html**.

1.3.6. Инициализация страниц. Макрос autoexec

Макрос **autoexec** обычно используют для того, чтобы подгрузить описания макросов из других файлов (но не из **_macro.cfg**) или инициализировать переменные (см. п. 1.5). Если в шаблоне описан макрос с таким именем, он будет вызван автоматически в первую очередь (перед

вызовом макроса **main**). Результат, получаемый при вызове макроса **autoexec**, Парсером игнорируется.

Пример. Используем макрос **autoexec** для того, чтобы инициализировать переменную **title** и подгрузить описания макросов из файла **_my_macro.cfg**.

```
@main[] HTML-код страницы
   <HTML>
   <HEAD><TITLE>Пример использования макроса autoexec</TITLE></HEAD>
   <BODY>
      <H1>^var[title]</H1>
   </BODY>
   </HTML>
@autoexec[] инициализация страницы
   ^macro_use[my_macro.cfg]
   ^var[title;Используем макрос autoexec]
```

1.4. Таблицы

1.4.1. Зачем нужны таблицы?

Вернемся к примеру, рассмотренному в п. 1.3.2. Вы помните, что номер телефона, который должен был отображаться на разных страницах сайта, мы вынесли в текст макроса **my_phone**. Если теперь потребуется отображать еще адрес и время работы, нам ничто не помешает также внести эти данные в макрос. Например, так.

```
@my_contacts[text_before_phone;text_before_address;text_before_time]
<B><I>$text_before_address</I> Москва, Газетный переулок</B>
<B><I>$text_before_time</I> 10:00-19:00</B>
<B><I>$text_before_phone</I> +7 (095) 229-85-23</B>
```

Можно поместить в текст макроса сведения обо всех представительствах компании, но пользоваться им будет уже неудобно. Представьте себе, какой объем ручной правки потребуется для того, чтобы, допустим, изменить оформление всех надписей, если представительств будет штук пятьдесят. Наконец, если данные хранятся не у кого-нибудь из нас в голове, а в базе данных, не станем же мы вручную переносить их в текст макроса! Поэтому для работы с данными, представленными в виде таблиц, в Парсере предусмотрено специальное средство — *таблица*.

1.4.2. Механизм таблиц в Парсере

Механизм *таблиц* предназначен для работы с данными табличного формата. Данные, которые хранятся в текстовом файле или в базе данных, можно загрузить в таблицу — после этого к ним можно будет обращаться из шаблонов страниц. Жесткой связи между таблицей и каким-либо источником данных нет; можно использовать таблицу только для работы с данными в оперативной памяти. Таблица идентифицируется своим *именем*. Столбцы таблицы имеют номера и одновременно могут быть поименованы. Если столбцы таблицы поименованы, к ним можно обращаться как по их именам, так и по номерам. К непоименованным столбцам таблицы обращаются только по номерам.

1.4.3. Текущая таблица

Прежде чем приступить к работе с той или иной таблицей, необходимо назначить ее *текущей*. Для этого служит оператор **context**.

```
^context[имя_таблицы;код]
```

В первом аргументе передается имя таблицы, во втором — код, для которого эта таблица назначается текущей. Код обрабатывается Парсером. Если таблица с указанным именем не существует, она создается.

Замечание. Код, передаваемый оператору **context** в свою очередь может содержать вызовы этого оператора.

```
^context[table1;
         ^rem[Текущая таблица - table1]
         ^context[table2;
                  ^rem[Текущая таблица - table2]
         ]        ^rem[Текущая таблица - снова table1]
]
```

Замечание. Если ни одна таблица не назначена текущей с помощью оператора **context**, то текущей является *безымянная таблица*. Использовать ее не рекомендуется.

1.4.4. Заполнение таблицы данными

Текущую таблицу можно заполнить данными из источников следующих типов:

- Код (а именно: оператор **config** — см. п. 3.6.1).
- Текстовый файл в формате *tab-delimited*. В таких файлах соседние строки таблицы разделены символом перевода строки, а значения в соседних столбцах — символом табуляции. Для загрузки в текущую таблицу данных из файла, устроенного подобным образом, служит оператор **load** (см. п. 3.6.3).
- Результат выполнения запроса на языке SQL. Для подключения к серверу баз данных в Парсере предусмотрен оператор **server**, а для работы с запросами — **sql** (см. п. 3.5.2).

Далее мы будем активно использовать оператор **load** в примерах, поэтому сейчас рассмотрим его более подробно.

`^load[имя_файла]`	Данные из файла с указанным именем загружаются в текущую таблицу.
`^load[named;имя_файла]`	Данные из файла с указанным именем загружаются в текущую таблицу. Значение **named** указывает, что первая строка файла содержит не данные, а имена столбцов.

Замечание. Имя файла на диске должно начинаться символом подчеркивания, который не указывается в аргументе оператора.

Внимание! При загрузке данных в таблицу предыдущее содержимое этой таблицы теряется.

Пример. Загрузим в таблицу с именем **contacts** координаты нескольких представительств компании. Сначала создадим файл с именем **_contacts.dat** и наберем в нем следующий текст.

_contacts.dat

```
address >                    time >          phone
Москва, Газетный переулок > 10:00-19:00 >   229-85-23
Антарктида >                 круглосуточно > 229-88-33
```

Не забудьте, что значения в соседних столбцах (например, слова **address** и **time** или **Антарктида** и **круглосуточно**) должны быть разделены не пробелами, а одним символом табуляции. Приведенный ниже фрагмент кода загрузит данные из этого файла в таблицу с именем **contacts**. Имена столбцов таблицы будут взяты из первой строки файла.

load_contacts.html

```
^context[contacts;
         ^load[named;contacts.dat]
]
```

1.4.5. Текущая строка, навигация и поиск в таблице

В каждой таблице имеется *текущая строка*. Оператор **item** (см. п. 1.4.5) возвращает значение, расположенное на пересечении заданного столбца и текущей строки. В только что созданной таблице текущей является первая строка.

Побочным эффектом ряда операторов является смена текущей строки. Если номер новой текущей строки больше (или меньше) номера предыдущей текущей строки на N, будем говорить, что произошло *перемещение* на N строк вперед (или назад).

Для перемещения на определенное «расстояние» от текущей строки используют оператор **shift** (см. п. 3.6.9). Для перемещения к строке, у которой в определенном столбце находится заданное значение, используют оператор **locate** (см. п. 3.6.8). Для того чтобы «обойти» все строки таблицы, породив для каждой из них некоторый код, используют операторы **menu** (см. п. 3.6.6) и **table** (см. п. 3.6.7).

1.4.6. Обращение к данным в таблице

Оператор **item** для текущей таблицы возвращает значение, расположенное на пересечении текущей строки и столбца с определенным именем или номером. Этот оператор имеет несколько форматов вызова.

^item[имя_или_номер_столбца]	Для текущей таблицы возвращает значение, расположенное на пересечении указанного столбца и текущей строки.
^item[next;имя_или_номер_столбца] **^item[prev;имя_или_номер_столбца]**	Для текущей таблицы возвращает значение, расположенное на пересечении указанного столбца и строки, которая следует за текущей (или предшествует ей).
^item[имя_таблицы;имя_или_номер_столбца]	Для указанной таблицы возвращает значение, расположенное на пересечении указанного столбца и текущей строки.

Пример. Модифицируем пример, приведенный в п. 1.4.4. Выведем данные из первой строки таблицы.

show_contacts.html

```
<HTML>
<HEAD><TITLE>Пример использования оператора item</TITLE></HEAD>
<BODY>
^context[contacts;
         ^load[named;contacts.dat]
         Адрес: ^item[address]
         Время работы: ^item[time]
         Телефон: ^item[phone]
]
</BODY>
</HTML>
```

1.4.7. Перебор строк текущей таблицы

В том случае, если для каждой строки текущей таблицы требуется сформировать фрагмент HTML-кода, используют оператор **menu**.

^menu[код]	Последовательно делает текущей каждую строку текущей таблицы. Для каждой строки обрабатывает переданный код. Перебор начинается с первой строки таблицы и заканчивается последней.
^menu[код;код_разделитель]	То же самое, но вдобавок код, переданный во втором аргументе, будет вставлен между непустыми результатами обработки строк.

Пример. Модифицируем пример, приведенный в п. 1.4.6. Отобразим данные из текущей таблицы в виде HTML-таблицы.

show_contacts_as_table.html

```
<HTML>
<HEAD><TITLE>Пример использования оператора menu</TITLE></HEAD>
<BODY>
^context[contacts;
         ^load[named;contacts.dat]
            <TABLE><TR>
               <TD>Адрес</TD>
               <TD>Время работы</TD>
               <TD>Телефон</TD>
            </TR>
            ^menu[<TR>
                     <TD>^item[address]</TD>
                     <TD>^item[time]</TD>
                     <TD>^item[phone]</TD>
                  </TR>
            ]
         </TABLE>
]
</BODY></HTML>
```

1.5. Переменные

Переменные предназначены для хранения *значений*, которые используются при обработке шаблонов. В процессе обработки одного шаблона все переменные являются глобальными, т. е. доступны в любом месте кода. Это относится и к самому коду шаблона, и к макросам, описанным в других файлах, если шаблон содержит ссылки на них.

Для того чтобы завести переменную, достаточно присвоить ей то или иное значение. *Имя переменной* может состоять из латинских букв, цифр и символов подчеркивания, но не должно начинаться цифрой.

Значения переменных, вообще говоря, являются строками, однако операторы могут интерпретировать их как целые числа в десятичной, восьмеричной или шестнадцатеричной записи или как действительные числа.

Для работы с переменными используется оператор **var**. С помощью этого оператора вы можете присваивать переменным значения и считывать значения переменных.

^var[имя_переменной;значение]	Переменной присваивается указанное значение.
^var[имя_переменной;операция;операнд]	Над значением переменной и операндом выполняется операция. Результат заносится в переменную. Допустимы следующие операции: **+** (сложение), **-** (вычитание), ***** (умножение), **/** (деление) и **.** (конкатенация строк).
^var[имя_переменной;строка_формата; математическое_выражение]	Значение математического выражения (см. п. 1.7) вычисляется и преобразовывается к виду, определяемому строкой формата (см. п. 1.8). Полученный результат записывается в переменную.
^var[имя_переменной]	Возвращает значение переменной.

1.6. Уровни обработки

В коде, который обрабатывается Парсером, могут встречаться выражения на языках HTML, JavaScript и SQL. Между тем результаты, возвращаемые некоторыми операторами, а именно **item**, **form**, **cookie**, **var** и **env**, вообще говоря, могут не соответствовать синтаксису этих языков. Например, если переменная **neravenstvo** содержит значение **A<B**, то вызов **^var[neravenstvo]** выдаст результат, непригодный для вставки в HTML-код (так как браузер может «запутаться» в тегах). Поэтому в Парсере предусмотрен механизм *уровней обработки*, позволяющий при вставке в код «пройтись напильничком» по результатам этих операторов. В зависимости от установленного уровня обработки в них выполняются те или иные замены.

Существуют следующие уровни обработки:

- **html** — символы **"**, **<**, **>** и **&** заменяются строками **"**, **<**, **>**, и **&** соответственно. Символ табуляции заменяется пробелом;
- **config** — символы табуляции и перевода строки заменяются пробелом;
- **javascript** — символы с кодами **255d**, **209d** (буква **я** в кодировках windows-1251 и koi8-r соответственно) для обеспечения совместимости с Netscape Navigator 3

заменяются на **\я**. Символы **'**, **"** и **** предваряются символом ****. Символы с кодами **10d** и **13d** преобразуются в **\n** и **\r** соответственно;

- **sql** — для сервера Oracle символ **'** заменяется парой **''**, символ табуляции — пробелом. Для сервера MySQL символы **'**, **"** и **** предваряются символом ****. Символы с кодами **10d** и **13d** преобразуются в символы **\n** и **\r** соответственно. Символ табуляции заменяется пробелом;
- **none** — никакие замены не выполняются.

По умолчанию установлен уровень обработки **html**. Для переключения уровней обработки используют оператор **level**.

```
^level[уровень_обработки;код]
```

Для переданного кода устанавливается указанный уровень обработки. Если при обработке этого кода встретится один из операторов **item**, **form**, **cookie**, **var** или **env**, то его результат будет преобразован в соответствии с установленным уровнем обработки.

Замечание. Для кода, расположенного внутри операторов **config** и **sql**, соответствующий уровень обработки устанавливается автоматически.

1.7. Математические и логические выражения

Математические выражения в языке Парсера составляются по обычным правилам с использованием знаков математических операций, имен функций и круглых скобок. Операндами в математическом выражении могут быть константы, значения, взятые из таблиц, и переменные.

Некоторые операторы, например **if**, используют значение математического выражения в качестве критерия при выборе одной альтернативы из двух предложенных. В таких случаях будем говорить, что мы имеем дело не с математическим, а с логическим значением. Любое отличное от нуля число интерпретируется как логическое значение **истина**, а 0 — как логическое значение **ложь**. Если значение выражения отлично от нуля, будем называть такое выражение истинным, в противном случае — ложным.

Замечание. Можно подставлять в математическое выражение переменные без оператора **var**. Достаточно указать в качестве операнда имя переменной, например, вместо **^var[mnogo_deneg]** написать просто **mnogo_deneg**. Аналогично, обращаясь в математическом выражении к элементам таблиц, можно не использовать оператор **item**. Для того чтобы обозначить элемент таблицы, используйте такую форму записи: **имя_таблицы::имя_столбца** или **::имя_столбца** (для текущей таблицы).

Замечание. Точность вычислений зависит от платформы, на которой установлены веб-сервер и Парсер. На большинстве платформ мы сможем работать со значениями в диапазоне $\pm 10^{307}$ с пятнадцатью значащими цифрами.

Арифметические операции

Знаки операции	Операция
+	Сложение
-	Вычитание

`*`	Умножение
`/`	Деление
`%`	Остаток от деления. Для отрицательных чисел результат отличается от принятого в математике, например –11%2=–1

Побитовые операции

Знаки операции	Операция
`~`	Побитовая инверсия
`^`	Побитовая операция XOR
`&`	Побитовая операция AND
`|`	Побитовая операции OR

Логические операции

Для всех логических операций результат равен `1`, если условие выполняется, и `0` в противном случае.

Знаки операции	Операция
`<`	Меньше
`>`	Больше
`<=`	Меньше или равно
`>=`	Больше или равно
`==`	Равно
`!=`	Неравно
`&&`	Логическая операция AND
`||`	Логическая операция OR
`!`	Логическая операция NOT

Функции

Имя функции	Функция
`round()`	Округление до ближайшего целого. Например, round(–1.23)=–1, round(–1.58)=–2, round(1.58)=2
`floor()`	Операция округления до целого в меньшую сторону, например floor(1.23)=1, floor(–1.23)=–2
`ceiling()`	Операция округления до целого в большую сторону, например ceiling(1.23)=2, ceiling(–1.23)=–1

abs()	Абсолютная величина (модуль), например abs(1.23)=1.23, abs(–1.23)=1.23
sign()	Знак выражения. Например, sign(1.23)=1, sign(0)=0, sign(–1.23)=–1

1.8. Форматные строки

Форматная строка определяет форму представления значения вычисленного математического выражения. В общем случае она имеет следующий вид: **%Длина.ТочностьТип**.

Замечание для тех, кто знаком с программированием на языке Си. Форматная строка устроена так же, как в функции **printf**.

Тип — определяет способ преобразования числа в строку. Существуют следующие типы:

d — десятичное целое число со знаком;
u — десятичное целое число без знака;
o — восьмеричное целое число без знака;
x — шестнадцатеричное целое число без знака; для вывода цифр, больших **9**, используются буквы **a**, **b**, **c**, **d**, **e**, **f**;
X — шестнадцатеричное целое число без знака; для вывода цифр, больших **9**, используются буквы **A**, **B**, **C**, **D**, **E**, **F**;
f — действительное число.

Точность — точность представления дробной части, т. е. количество знаков после запятой. Если для отображения дробной части значения требуется больше знаков, то значение округляется. Обычно точность указывают в том случае, если используется тип преобразования **f**. Для других типов указывать точность не рекомендуется. Если точность не указана, то для типа преобразования **f** она по умолчанию принимается равной **6**. Если указана точность **0**, то число выводится без дробной части.

Длина — количество знаков, отводимое для значения. Может получиться так, что для отображения полученного значения требуется меньше символов, чем указано в блоке **Длина**. Например, указана длина **10**, а получено значение 123. В этом случае слева к значению будет приписано семь пробелов. Если нужно, чтобы слева приписывались не пробелы, а нули, следует в начале блока **Длина** поместить **0**, например, написать не **10**, а **010**. Блок **Длина** может отсутствовать, тогда для значения будет отведено ровно столько символов, сколько требуется для его отображения.

2. Пример сайта

2.1. О чем эта глава?

Давайте создадим простой веб-сайт и посмотрим, каким образом с помощью Парсера решаются некоторые распространенные задачи. В частности, мы научимся формировать меню разделов, средства альтернативной навигации и списки новостей. Начнем, как водится, с технического задания. Затем наметим общую схему решения поставленной задачи. После этого создадим файлы данных и шаблоны страниц, подробно рассмотрим их устройство, укажем на некоторые недостатки предложенного решения. В конце главы приводится список упражнений и даются ненавязчивые рекомендации по их выполнению.

Замечание. Перед тем как погрузиться в проектирование и создание учебного сайта, имеет смысл прочитать описания операторов, которые мы будем использовать, а именно: `context`, `eval`, `form`, `if`, `item`, `load`, `locate`, `macro`, `menu`, `truncate`, `var`.

2.2. Техническое задание

Сайт состоит из четырех разделов: «Новости», «Статьи», «Ссылки» и «Контакты». Набор разделов может меняться, поэтому сайт должен быть сделан таким образом, чтобы добавление, удаление или изменение раздела требовало как можно меньше ручной правки.

О каждом разделе известна следующая информация:

- заголовок;
- имя HTML-файла.

Каждый раздел представляет собой отдельную страницу. Все разделы устроены единообразно и состоят из следующих функциональных частей:

- меню;
- заголовок раздела;
- информационное наполнение раздела;
- средства альтернативной навигации.

Меню содержит ссылки на все разделы сайта. Название самого раздела тоже присутствует в меню, но ссылка на нем не ставится. Меню оформлено в виде таблицы, состоящей из одной строки. Каждая ссылка находится в отдельной ячейке таблицы. Названия разделов в ячейках центрированы по горизонтали. Из эстетических соображений ячейки должны быть одинаковой длины.

Заголовок раздела представляет собой строку, заключенную в теги `<H1></H1>`.

Функциональность и оформление информационного наполнения раздела зависит от того, что содержательно представляет собой этот раздел. Например, в разделе «Новости» это список новостей, оформленный определенным образом (см. ниже).

Средства альтернативной навигации обеспечивают посетителю возможность последовательно «перелистывать» все разделы сайта от первого до последнего. В каждом разделе средства альтернативной навигации состоят из двух ссылок: на предыдущий и на следующий раздел. Исключениями из этого правила являются первый и последний разделы (в них нет ссылок на предыдущий и следующий раздел соответственно). Ссылка на предыдущий раздел выглядит

так: `<<<<`, а на следующий `>>>>`. От остального пространства страницы средства альтернативной навигации отделены двумя горизонтальными линиями (`<HR>`).

Внимательный читатель обратил внимание, что мы до сих пор ни слова не сказали о взаиморасположении перечисленных выше конструктивных элементов на странице. Это не случайно. Поскольку Парсер позволяет задать типовую структуру страниц, вопрос о том, как именно они будут устроены, перестает быть ключевым для HTML-кодера. Сколько бы ни было разделов, мы сможем изменить их устройство, внеся правку в один единственный файл. Пока сделаем так: сверху поместим меню, под ним заголовок, затем информационное наполнение раздела, а внизу — средства альтернативной навигации.

Раздел «Новости» содержит ссылки на все новости, опубликованные на сайте.

О каждой новости известна следующая информация:

- заголовок;
- дата;
- текст.

Ссылка ставится на заголовке новости. Справа от ссылки ставится дата новости, выделенная курсивом. Ниже со сдвигом вправо отображается начало текста новости (возьмем первые 170 символов). Добавление, удаление, изменение новостей должно обходиться как можно «дешевле» и требовать минимума исправлений в шаблонах страниц. По ссылке из раздела новостей посетитель может перейти к соответствующей новости. Страница новости устроена очень просто: сверху заголовок, на следующей строчке — дата, а дальше — основной текст.

Проектирование остальных трех разделов предоставляется читателю в качестве упражнения.

2.3. Конструкция сайта

Всю информацию о разделах будем хранить в файле формата tab-delimited, назовем его `_sections.cfg`. При формировании HTML-кода разделов будем пользоваться этой информацией, загружая ее в таблицу. Каждому разделу присвоим уникальный числовой идентификатор, который тоже будет храниться в файле `_sections.cfg`. Раздел всегда можно будет опознать по этому номеру, что избавит нас от ручной правки шаблонов в случае изменения заголовков или переименования файлов.

В файле `_sections.cfg` предусматриваются следующие столбцы:

- `no` — уникальный числовой идентификатор раздела;
- `title` — заголовок раздела;
- `file` — имя файла раздела.

Одинаковую для всех разделов структуру страницы опишем в виде макроса в файле `_macro.cfg`. Этот макрос назовем `section_html`. Его аргументы — идентификатор раздела `section_no` и код его информационного наполнения `section_body`.

Для каждого раздела создадим отдельный шаблон, в частности, для раздела новостей создадим шаблон `news.html`. В каждом шаблоне раздела опишем макрос `main`. Этот макрос будет формировать код информационного наполнения раздела и вызывать макрос `section_html`, передавая ему номер раздела, который заведомо известен, и сформированный код.

Первым делом макрос **section_html** с помощью операторов **context** и **load** назначает текущую таблицу и загружает в нее данные из файла **_sections.cfg**.

Затем он конструирует меню разделов, проходя с помощью оператора **menu** по всей таблице. При этом для того чтобы понять, когда нужно ставить ссылку, а когда нет, номер раздела в текущей строке таблицы (значение в столбце **no**) сравнивается с номером раздела, переданным в аргументе **section_no**. Если равенство соблюдается, значит, оператор **menu** «наступил» на строку, соответствующую формируемому разделу, и ссылку ставить не нужно. Во всех остальных случаях ссылка ставится.

Однако мы не учли одно из требований технического задания: ячейки HTML-таблицы должны быть одинаковой ширины. Чтобы добиться этого, укажем в каждой ячейке HTML-таблицы ее ширину в процентах от ширины окна. Это значение вычислим так:

```
^eval[round(^item[count]/100)]
```

Конечно, эти вычисления выполняются до вызова оператора **menu**, но здесь мы пишем о них после, чтобы не заострять внимание на второстепенном.

Сформировав меню, нужно отобразить заголовок раздела. Это не так просто, поскольку макросу передается не заголовок раздела, а его идентификатор. Поэтому заголовок придется найти в таблице по столбцу **no**:

```
^locate[no;$section_no]
```

После заголовка оператор вставляет информационное наполнение раздела, которое получает «на халяву» при вызове — это значение аргумента **section_body**.

Теперь остается только сформировать ссылки на предыдущий и следующий разделы. Воспользуемся тем, что после вызова оператора **locate** текущей в таблице является строка, соответствующая формируемому разделу. Следовательно, вызвав оператор **item** с аргументом **next** или **prev**, мы можем получить имя файла следующего или предыдущего раздела соответственно. Кстати, вы не забыли, нужно ведь еще проверить, не является ли раздел первым или последним? Для этого снова используем оператор **item** с аргументом **count** (чтобы узнать номер последнего раздела, считая с **1**) и к оператору **item** с аргументом **number** (чтобы узнать номер формируемого раздела). Если номер раздела равен **1**, то мы имеем дело с первым разделом и не ставим ссылку на предыдущий, а если количеству разделов — то с последним и не ставим ссылку на следующий. Так, для формирования ссылки на предыдущий раздел имеем:

```
^if[^item[number]==1;
    &lt^;&lt^;&lt^;&lt^;;
    <A HREF="^item[prev;file]">&lt^;&lt^;&lt^;&lt^;</A>
]
```

Новости также поместим в файл формата tab-delimited, который будет называться **_news.cfg**. В этом файле будут следующие столбцы:

- **no** — уникальный числовой идентификатор новости;
- **title** — заголовок новости;

- **date** — дата новости;
- **text** — текст новости.

Поскольку новости обновляются часто, мы не станем создавать для каждой из них отдельный шаблон. Лучше подготовим единый шаблон **newsdetails.html**, принимающий в качестве параметра идентификатор той новости, которую требуется отобразить в данный момент. В разделе новостей на этот шаблон будет стоять несколько ссылок с разными параметрами:

```
<A HREF="newsdetails.html?n_no=1">Москву атакуют гигантские кролики</A>
<A HREF="newsdetails.html?n_no=2">С мая введут акцизы на интернет</A>
...
```

Для каждой новости своя ссылка. Кстати, эти ссылки и есть информационное наполнение раздела, которое макрос **main** из шаблона **news.html** передает макросу **section_html** в качестве значения аргумента **section_body**. Шаблон **newsdetails.html** будет загружать все новости в таблицу и находить в ней новость с переданным идентификатором:

```
^context[news;
         ^load[named;news.cfg]
         ^locate[no;^form[n_no]]
]
```

2.4. Реализация

2.4.1. Файлы данных

В корневом каталоге сайта создадим файлы **_sections.cfg** и **_news.cfg**. В первой строке каждого из этих файлов укажем названия свойств разделов (или новостей).

_sections.cfg

```
no > title >    file
1 >  Новости >  news.html
2 >  Статьи >   articles.html
3 >  Ссылки >   links.html
4 >  Контакты > address.html
```

_news.cfg

```
no > title >                              date >       text
1 >  Москву атакуют гигантские кролики > 05.03.2001 > Полчища...
2 >  С мая вводятся акцизы на интернет > 04.03.2001 > Каждый...
3 >  Букву ё скоро отменят >             03.03.2001 > Межведомств...
```

2.4.2. Структура разделов. Макрос section_html

Создадим файл `_macro.cfg` и опишем в нем макрос `section_html`.

_macro.cfg

```
@section_html[section_no;section_body] код раздела
<HTML>
<HEAD><TITLE>Учебный сайт</TITLE></HEAD>
<BODY>
# Назначим таблицу sections текущей
^context[sections;
# Загрузим в таблицу данные о разделах
        ^load[named;sections.cfg]
#
#       Сформируем меню разделов
        <TABLE BORDER=1 WIDTH=100% CELLSPACING=0>
#       Вычислим ширину ячейки в процентах от ширины окна
        ^var[menu_item_width;^eval[round(100/^item[count])]]
        <TR ALIGN=CENTER>
#          Пошли перебирать разделы
           ^menu[<TD WIDTH=^var[menu_item_width]%>
#                Будем ли ставить ссылку?
                 ^if[^item[no]==$section_no;
#                    Ссылку на этот раздел ставить не будем
                     ^item[title];
#                    А на другие разделы будем
                     <A HREF="^item[file]">^item[title]</A>
                 ]
                </TD>
           ]
        </TR>
        </TABLE>
#
#       Выведем заголовок раздела
#       Сначала найдем раздел по его номеру
        ^locate[no;$section_no]
#       А теперь выведем заголовок
        <H1>^item[title]</H1>
#
#       Наконец, вставим собственно содержимое раздела
        $section_body
#
#       Альтернативная навигация
        <HR>
        <CENTER>
#       Ссылка на предыдущий раздел, если он есть
        ^if[^item[number]==1;
            &lt^;&lt^;&lt^;&lt^;;
            <A HREF="^item[prev;file]">&lt^;&lt^;&lt^;&lt^;</A>
        ]
          ^; ^; ^; ^;
#       Ссылка на следующий раздел, если он есть
        ^if[^item[number]==^item[count];
            &gt^;&gt^;&gt^;&gt^;;
            <A HREF="^item[next;file]">&gt^;&gt^;&gt^;&gt^;</A>
        ]
        </CENTER>
        <HR>
]
</BODY>
</HTML>
```

2.4.3. Шаблон раздела «Новости»

Шаблон раздела «Новости» содержит единственный макрос — **main**. Он вызывает описанный в файле **_macro.cfg** макрос **section_html**, передавая ему номер раздела новостей и код, содержащий ссылки на новости. Напомним, что разделу новостей мы присвоили номер **1**.

news.html

```
@main[] раздел "Новости"
# Вызываем макрос section_html для формирования кода раздела
^macro[section_html;1;
# Номер раздела "Новости" - 1

# А теперь сформируем тело раздела, т.е. список новостей
        ^context[news;
                ^load[named;news.cfg]
                ^menu[
#                       Ссылка на новость
                        <P>
         <A HREF="newsdetails.html?n_no=^item[no]">^item[title]</A>
#                       Дата
                        <I>^item[date]</I>
                        </P>
#                       Начало текста новости
                        <P><UL>^truncate[^item[text];170]</UL></P>
                ]
        ]
]
```

2.4.4. Единый шаблон новости

Шаблон для новостей принимает в качестве параметра номер новости, которую требуется отобразить. Он загружает все новости в таблицу и с помощью оператора **locate** по столбцу **no** находит нужную новость.

newsdetails.html

```
<HTML>
<HEAD><TITLE>Учебный сайт</TITLE><HEAD>
<BODY>
^context[news;
          ^load[named;news.cfg]
          ^locate[no;^form[n_no]]
          <H1>^item[title]</H1>
          <P>^item[date]</P>
          <P>^item[text]</P>
]
</BODY>
</HTML>
```

2.5. Работа над ошибками

Теперь обратим внимание на некоторые недостатки нашего решения и посмотрим, каким образом эти недостатки можно было бы устранить.

Возможность деления на ноль. При вычислении ширины ячейки в HTML-таблице мы выполняем деление, не проверив предварительно делитель на неравенство нулю. Поэтому, если в файле **_sections.cfg** не окажется ни одной записи, Парсер выдаст сообщение

об ошибке, что некрасиво и дискредитирует разработчиков сайта в глазах посетителей. В этой ситуации предусмотрительный кодер поступил бы примерно следующим образом:

```
# Назначим таблицу sections текущей
^context[sections;
#         Загрузим в таблицу данные о разделах
          ^load[named;sections.cfg]
#         Проверим, есть ли в таблице хоть одна строка
          ^empty[Приветствуем вас на самом скучном в мире сайте;
#                 Если есть, то продолжим. Сформируем меню разделов
                  ...
          ]
          ...
]
```

В меню всегда только одна строка. А что произойдет, если разделов будет не четыре, а двадцать пять? В строке окажется двадцать пять ячеек со втиснутыми в них ссылками. Как бы ни отреагировал на это браузер, результат будет выглядеть непрезентабельно. Одно из возможных решений проблемы — зафиксировать количество столбцов в HTML-таблице и воспользоваться оператором `table` для того, чтобы расположить ссылки в нескольких строках.

```
#         Сформируем многострочное меню разделов
#
          <TABLE BORDER=1 WIDTH=100% CELLSPACING=0>
          <TR ALIGN=CENTER>
#            Пошли перебирать разделы
             ^table[<TD WIDTH=25%>
#                   Будем ли ставить ссылку?
                    ^if[^item[no]==$section_no;
#                      Ссылку на этот раздел ставить не будем
                       ^item[title];
#                      А на другие разделы будем
                       <A HREF="^item[file]">^item[title]</A>
                    ]
                    </TD>;
                    4; ^rem[не более четырех ссылок в строке]
                     ^;
             ]
          </TR>
          </TABLE>
```

Ссылки на новости выводятся в произвольном порядке. Точнее, в том порядке, в котором новости расположены в файле `_news.cfg`. Между тем ссылки на новости обычно выводят в порядке убывания дат. Сортировка таблицы с помощью оператора `sort` в данном случае не поможет, поскольку этот оператор не предусматривает сортировку по датам. Наиболее эффективное и универсальное решение в данном случае — хранить новости в базе данных.

2.6. Упражнения

1. Добавьте на все страницы сайта адрес вебмастера и копирайт. Изменение этих данных должно требовать редактирования только одного файла. В копирайте всегда должен быть указан текущий год.
2. Организуйте альтернативную навигацию по новостям. Для каждой новости выводите ее порядковый номер и количество новостей, например, `4 из 27`. Учтите, что в файле `_news.cfg` указаны уникальные идентификаторы новостей, а не их порядковые номера.

3. Сделайте в новостях меню разделов (подумайте, возможно, имеет смысл начать с вынесения меню разделов в отдельный макрос).
4. Измените компоновку разделов. Сделайте меню вертикальным и поместите его слева. Создайте (место выберите сами) блок со ссылками на три последние новости. Будем считать, что новости расположены в файле **`_news.cfg`** в порядке убывания дат.
5. Выводите текст только для тех новостей, которые посетитель еще не просмотрел (используйте оператор **`cookie`**).

3. Операторы Парсера

3.1. Переменные, выражения, вычисления

3.1.1. Файлы-счетчики. Оператор counter

Формат вызова

```
^counter[имя_файла]
```

Аргументы

имя_файла — начало имени файла, в котором хранится текущее значение счетчика. Имя файла на диске заканчивается расширением **.count**, которое не указывают при вызове оператора.

Описание

Оператор возвращает значение хранимого в файле счетчика, а затем увеличивает его на `1`. Если файл с указанным именем не существует, то он будет создан. Значение счетчика в этом случае будет установлено равным `0`.

Примеры

Следующий вызов вернет значение счетчика, хранящееся в файле **index.count**, а затем увеличит его на `1`.

```
Количество посещений с 1832 года: ^counter[index]
```

3.1.2. Математические выражения. Оператор eval

Формат вызова

```
^eval[математическое_выражение]

^eval[математическое_выражение;форматная_строка]
```

Аргументы

математическое_выражение — выражение, значение которого требуется вычислить.

форматная_строка — строка, которая задает форму представления результата.

Описание

Оператор возвращает значение математического выражения (см. п. 1.7). Значение оформляется в соответствии с форматной строкой, если она передана при вызове (см. п. 1.8).

Примеры

Вызов	Результат
`^eval[2*2]`	`4`
`^eval[quantity*goods::price]`	Произведение значения переменной **quantity** и значения, расположенного на пересечении столбца **price** и текущей строки в таблице **goods**
`^eval[0.4*100;Содержание спирта в водке %2.2f%%]`	`Содержание спирта в водке 40.00%`
`^eval[256+10;%08X]`	`0000010A`
`^eval[25.12;%08.3f]`	`0025-120`

3.1.3. Присвоение значения переменной. Оператор var с двумя аргументами

Формат вызова

```
^var[имя_переменной;значение]
```

Аргументы

имя_переменной — имя переменной, которой присваивается значение.

значение — присваиваемое значение.

Описание

Оператор присваивает переменной значение. Результатом обработки оператора всегда является пустая строка.

Примеры

```
^var[mnogo_deneg;Три рубля]
```

```
^var[today;^date[]]
```

```
^var[factorial5;^eval[1*2*3*4*5]]
```

3.1.4. Получение значения переменной. Оператор var с одним аргументом

Формат вызова

```
^var[имя_переменной]
```

Аргументы

имя_переменной — имя переменной, значение которой требуется получить.

Описание

Оператор возвращает значение переменной, имя которой ему передано. Если переменной с указанным именем ранее не было присвоено никакого значения, оператор вернет пустую строку.

Примеры

Присвоим переменной **mnogo_deneg** значение **три рубля**, а затем выведем это значение.

```
^var[mnogo_deneg;три рубля]
У меня в кармане ^var[mnogo_deneg]
```

3.1.5. Присвоение переменной значения. Оператор var с аргументом-операцией

Формат вызова

^var[имя_переменной;операция;операнд]

Аргументы

имя_переменной — имя переменной, которой присваивается новое значение.

операция — знак операции.

операнд — операнд, участвующий в операции.

Описание

Выполняет операцию над значением, которое хранится в переменной, и операндом. Полученный результат заносит в переменную.

Допустимы следующие операции:

- **+** — сложение;
- **-** — вычитание;
- ***** — умножение;
- **/** — деление;
- **.** — сцепление (конкатенация) строк.

Примеры

Обработка следующего фрагмента даст результат

У меня в кармане три рубля и еще пять копеек.

```
^var[mnogo_deneg;три рубля]
^var[mnogo_deneg;.; и еще пять копеек]
У меня в кармане ^var[mnogo_deneg]
```

Следующий фрагмент кода выводит числа от `1` до `4`:

```
^var[counter;1]
^while[counter<5;
        ^var[counter]
        ^var[counter;+;1]<BR>
]
```

3.1.6. Присвоение переменной значения. Оператор var с форматированием

Формат вызова

`^var[имя_переменной;форматная_строка;математическое_выражение]`

Аргументы

`имя_переменной` — имя переменной, которой присваивается новое значение.

`форматная_строка` — строка, которая задает форму представления значения переменной.

`математическое_выражение` — математическое выражение.

Описание

Оператор вычисляет математическое выражение (см. п. 1.7), представляет результат в соответствии с форматной строкой (см п. 1.8) и заносит полученный результат в переменную.

Примеры

Вызов `^var[pi;%05.2f;3+0.1415]` поместит в переменную `pi` значение `03-14`.

3.1.7. Получение псевдослучайного числа. Оператор random

Формат вызова

`^random[]`

`^random[ширина_диапазона]`

Аргументы

`ширина_диапазона` — ширина диапазона, в который может попадать псевдослучайное число.

Описание

Оператор возвращает псевдослучайное неотрицательное целочисленное значение, строго меньшее, чем переданное в аргументе значение. Значение аргумента по умолчанию — `1000000`.

Примеры

Вызов `^random[200]` вернет целое число в диапазоне от `0` до `199`.

3.2. Макросы

3.2.1. Вызов макроса. Оператор macro

Формат вызова

```
^macro[макрос;аргумент1;аргумент2;....;аргументN]
```

Аргументы

`макрос` — имя вызываемого макроса.

`аргумент1`, … **`аргументN`** — значения аргументов, передаваемые макросу при вызове.

Описание

Оператор вызывает макрос и передает ему указанные значения аргументов. Результат вызова — обработанный Парсером текст макроса (см. п. 1.3).

Внимание! Обработка текста макроса выполняется в следующем порядке. Сначала имена аргументов, предваряемые символом `$`, заменяются их значениями, переданными при вызове. Важно, что значения аргументов передаются как есть, встречающиеся в них вызовы операторов не обрабатываются. Затем полученный текст обрабатывается Парсером. Приведенный ниже пример демонстрирует, что пренебрежение этим обстоятельством может приводить к ошибкам. Предположим, в файле **`_macro.cfg`** описан макрос **`repeat_text`**.

```
@repeat_text[snipet;times]
^var[i;0]
^while[i<$times;
       $snipet
       ^var[i;+;1]
]
```

В шаблоне страницы мы вызываем этот макрос в следующем контексте:

```
^var[i;9]
^macro[repeat_text;There is no beer;^var[i]]
```

Подстановка переданного значения аргумента **`times`** в текст макроса даст такой код:

```
^var[i;0]
^while[i<^var[i];
       $snipet
       ^var[i;+;1]
]
```

Легко видеть, что условие выполнения цикла всегда неверно, поэтому цикл не выполнится ни разу.

Примеры

Рассмотрим следующий фрагмент кода:

```
@section_header[title;date]
   <H1>$title</H1>
   <I>date</I>
@main
   <HTML>
   <HEAD><TITLE>Учебный сайт</TITLE></HEAD>
   <BODY>
      ^macro[section_header;Новости;^date[]]
   </BODY>
   </HTML>
```

Его обработка даст следующий результат:

```
<HTML>
<HEAD><TITLE>Учебный сайт</TITLE></HEAD>
<BODY>
<H1>Новости</H1> <I>05.03.2001</I>
</BODY>
</HTML>
```

3.2.2. Динамическое описание макроса. Оператор macro_new

Формат вызова

```
^macro_new[имя_макроса;текст_макроса]
```

```
^macro_new[имя_макроса[список_параметров];текст_макроса]
```

Аргументы

имя_макроса — имя определяемого макроса.

список_параметров — список параметров определяемого макроса.

описание_макроса — текст макроса.

Описание

Оператор определяет новый макрос. Если определяемый макрос не имеет аргументов, используется первая форма вызова, если имеет — то вторая.

Примеры

Опишем макрос `link_to_contents` без аргументов.

```
^macro_new[link_to_contents;<A HREF="contents.html">Содержание</A>]
```

Опишем макрос `page_title` с двумя аргументами и вызовем его.

```
^macro_new[page_title[title;tag];
           <$tag>$title</$tag>
]
^macro[page_title;Новости;H1]
```

Результатом такого вызова будет следующий код: `<H1>Новости</H1>`.

3.2.3. Загрузка макросов из файла. Оператор macro_use

Формат вызова

```
^macro_use[путь]
```

Аргументы

путь — путь к файлу или каталогу с описаниями макросов.

Описание

Оператор загружает описания макросов из указанного файла. Имя файла на диске должно начинаться символом подчеркивания, который не указывается при передаче значения аргумента. Если передана строка **parser.cfg**, загружаются описания макросов изо всех файлов с таким именем, начиная с корня сервера. Если передан путь к каталогу, загружаются описания макросов изо всех файлов, расположенных в этом каталоге.

Примеры

```
^macro_use[/conf/issues.cfg]
```

3.2.4. Проверка существования макроса. Оператор macro_exists

Формат вызова

```
^macro_exists[имя_макроса]
```

Аргументы

имя_макроса — имя макроса, существование которого требуется проверить.

Описание

Оператор возвращает значение **yes**, если макрос с указанным именем существует. В противном случае оператор возвращает пустую строку.

Примеры

```
^eq[^macro_exists[show_copyright];yes;
    ^macro[show_copyright];
    Копируй, кому не лень
]
```

3.3. Ветвление кода

3.3.1. Выбор одного варианта из двух. Оператор if

Формат вызова

```
^if[логическое_выражение;код_если_выражение_истинно]

^if[логическое_выражение;код_если_выражение_истинно;
                         код_если_выражение_ложно
]
```

Аргументы

логическое_выражение — логическое выражение, результат которого служит критерием выбора вариантов.

код_если_выражение_истинно — код, выполняемый, если логическое выражение истинно.

код_если_выражение_ложно — код, выполняемый, если логическое выражение ложно.

Описание

Оператор вычисляет значение логического выражения (см. п. 1.7). Если логическое выражение истинно, то оператор возвращает значение аргумента **код_если_выражение_истинно**. Если логическое выражение ложно, то оператор возвращает значение аргумента **код_если_выражение_ложно** или, если оно не было передано, пустую строку.

Замечание. Аргумент, значение которого не требуется для вычисления значения оператора, не обрабатывается Парсером. Это следует иметь в виду, если у операторов, входящих в этот аргумент, имеются побочные эффекты, и не рассчитывать на них.

Примеры

```
^if[1<2;Один и правда меньше двух;Математика сбоит]
```

3.3.2. Выбор одного варианта из нескольких. Оператор switch

Формат вызова

```
^switch[выражение;
        ^case[значение1;вариант1]
        ...
```

```
        ^case[значениеN;вариантN]
        ^case[DEFAULT;вариант_по_умолчанию]
]
```

Аргументы

выражение — выражение, значение которого служит критерием при выборе варианта.

значение1...значениеN — значения, по которым выбираются варианты.

вариант1...вариантN — варианты кода, сопоставленные значениям с соответствующими номерами.

вариант_по_умолчанию — код, не сопоставленный ни одному из значений.

Описание

Оператор **switch**, в отличие от оператора **if**, позволяет выбирать между несколькими, а не только двумя вариантами кода. Критерием выбора является значение заданного выражения. Каждому значению из некоторого набора сопоставляется один вариант. Кроме того, можно задать код по умолчанию, который будет выбран в том случае, если значение выражения окажется вне заданного набора. Сопоставление некоторому значению варианта кода выполняется с помощью оператора **case**. В первом его аргументе передается значение выражения, а во втором — сопоставленный ему вариант кода. Для того чтобы задать код по умолчанию, следует в качестве первого аргумента оператора **case** указать слово **DEFAULT**.

Примеры

```
^switch[^var[sex];
        ^case[М;Мое почтенье, сударь!]
        ^case[Ж;Мое почтенье, сударыня!]
        ^case[DEFAULT;Ой, кто это?!]
]
```

3.3.3. Проверка строки на пустоту. Оператор default

Формат вызова

```
^default[строка;код_если_пусто]
```

Аргументы

строка — проверяемая строка; возвращается оператором, если она не пуста.

код_если_пусто — строка, возвращаемая оператором, если значение аргумента **строка** пусто.

Описание

Оператор возвращает значение аргумента **строка**, если оно не является пустой строкой. В противном случае возвращает значение аргумента **код_если_пусто**.

Примеры

```
^default[^form[name];Аноним]
```

3.3.4. Проверка строки на пустоту. Оператор ifdef

Формат вызова

```
^ifdef[строка;код_если_не_пусто]

^ifdef[строка;код_если_не_пусто;код_если_пусто]
```

Аргументы

строка — строка, о которой нужно узнать, пустая она или нет.

код_если_не_пусто — код, выполняемый, если строка не пуста.

код_если_пусто — код, выполняемый, если строка пуста.

Описание

Если строка не пуста, оператор возвращает значение аргумента **код_если_не_пусто**. В противном случае возвращается значение аргумента **код_если_пусто**. Если строка пуста, а значение аргумента **код_если_пусто** не указано, то возвращается пустая строка.

Примеры

```
^ifdef[^var[title];<H1>^var[title]</H1>;Без названия!]
```

3.3.5. Проверка значений на равенство. Оператор eq

Формат вызова

```
^eq[строка1;строка2;код_если_равно]

^eq[строка1;строка2;код_если_равно;код_если_неравно]
```

Аргументы

строка1, **строка2** — значения, проверяемые на равенство.

код_если_равно — возвращается, если значения аргументов **строка1** и **строка2** равны.

код_если_неравно — возвращается, если значения аргументов **строка1** и **строка2** не равны.

Описание

Проверит значения аргументов **строка1** и **строка2** на равенство. Если значения равны, возвращается значение аргумента **код_если_равно**, иначе —

код_если_неравно. Если равенство не соблюдается, а значение аргумента **код_если_неравно** не указано, возвращается пустая строка.

Примеры

```
^eq[1;2;никогда не будет возвращаться;всегда будет возвращаться]
```

3.3.6. Сравнение целочисленных значений. Оператор cmp

Формат вызова

```
^cmp[целое_число1;целое_число2;код_если_меньше;
код_если_равно;код_если_больше]
```

Аргументы

целое_число1;целое_число2 — целочисленные значения, которые требуется сравнить.

код_если_меньше — возвращается, **если целое_число1** меньше, чем **целое_число2**.

код_если_равно — возвращается, **если целое_число1** и **целое_число2** равны.

код_если_больше — возвращается, **если целое_число1** больше, чем **целое_число2**.

Описание

Оператор сравнивает значения аргументов **целое_число1** и **целое_число2**. Если **целое_число1** меньше, чем **целое_число2**, возвращается аргумент **код_если_меньше**, если больше — **код_если_больше**, а в случае равенства возвращается **код_если_равно**.

Примеры

```
^cmp[^var[speed];100;медленно!;хм, неплохо -
 100 км/час;спешишь на тот свет?]
```

3.3.7. Сравнение строки с началом URI страницы. Оператор start

Формат вызова

```
^start[начало_URI;код_если_совпадает;код_если_не_совпадает]
```

Аргументы

начало_URI — строка, сравниваемая с началом URI страницы.

код_если_совпадает — код, возвращаемый, если строка совпадает с началом URI текущей страницы.

код_не_совпадает — код, возвращаемый, если строка не совпадает с началом URI текущей страницы.

Описание

Если URI текущей страницы начинается значением аргумента **начало_URI**, то оператор возвращает значение аргумента **код_если_совпадает**, иначе — значение аргумента **код_если_не_совпадает**.

Примеры

```
^start[/admin;
       <BODY BGCOLOR=BLACK>;
       <BODY BGCOLOR=WHITE BACKGROUND=/i/very_beautiful.gif>
]
```

3.3.8. Вывод «полосатых» HTML-таблиц. Оператор color

Формат вызова

```
^color[hex-код_первого_цвета;hex-код_второго_цвета]
```

Аргументы

hex-код_первого_цвета — код «нечетного» цвета

hex-код_второго_цвета — код «четного» цвета.

Описание

Оператор используется в основном для оформления «полосатых» таблиц. Если номер вызова оператора в контексте текущей таблицы нечетный, оператор возвращает hex-код первого цвета, а если четный — hex-код второго цвета. Под hex-кодом в данном случае понимается последовательность из шести шестнадцатеричных цифр (три группы по две цифры), задающих цвет в палитре RGB.

Примеры

```
^context[cats;
         ^config[Черная кошка
                 Белый кот
                 Черная кошка
                 Белый кот]
         <TABLE>
         ^menu[<TR BGCOLOR=^color[#FFFFFF;#505050]>
                  <TD><FONT COLOR=#0F0F0F>^item[0]</FONT></TD>
               </TR>
         ]
         </TABLE>
]
```

3.4. Циклы

3.4.1. Цикл с условием. Оператор while

Формат вызова

```
^while[логическое_выражение;повторяющийся_код]
```

Аргументы

логическое_выражение — условие выполнения цикла.

повторяющийся_код — тело цикла.

Описание

Оператор тиражирует тело цикла до тех пор, пока значение логического выражения остается истинным (см. п. 1.7).

Замечание. После 10 000 повторений цикл в любом случае будет прерван. При этом будет возвращено значение **[endless loop detected]**.

Примеры

Рассмотрим такой фрагмент кода:

```
^var[i;4]
^while[i>=0;
       До взрыва осталось ^var[i] сек.<BR>
       ^var[i;-;1]
]
```

Его обработка даст следующий HTML-код:

```
До взрыва осталось 4 сек.<BR>3 сек.<BR>2 сек.<BR>1 сек.<BR>0 сек.<BR>
```

3.4.2. Цикл с параметром. Оператор for

Формат вызова

```
^for[имя_переменной_цикла;начало;конец;повторяющийся_код]

^for[имя_переменной_цикла;начало;конец;
     повторяющийся_код;разделитель
]
```

Аргументы

имя_переменной_цикла — имя переменной, которая является счетчиком цикла.

начало — математическое выражение, начальное значение счетчика цикла.

конец — математическое выражение, конечное значение счетчика цикла.

повторяющийся_код — тело цикла.

разделитель — фрагмент разделительного кода (указывать необязательно).

Описание

Присваивает счетчику цикла начальное значение и тиражирует тело цикла до тех пор, пока значение счетчика не превысит конечное значение. На каждом шаге счетчик цикла увеличивается на `1`. Начальное и конечное значение — математические выражения (см. п. 1.7). Разделительный код вставляется между непустыми шагами цикла.

Примеры

Рассмотрим такой фрагмент кода:

```
Шаги: ^for[i;1;2*2;
           номер ^var[i];,
].
```

Его обработка даст следующий HTML-код:

```
Шаги: номер 1, номер 2, номер 3, номер 4.
```

3.5. Базы данных

3.5.1. Подключение к SQL-серверу. Оператор server

Формат вызова

```
^server[параметры_подключения_к_серверу;код]
```

Аргументы

параметры_подключения_к_серверу — данные, необходимые для подключения к SQL-серверу.

код — код, обрабатываемый Парсером при установленном подключении.

Описание

Оператор устанавливает подключение к SQL-серверу. Переданный во втором аргументе код обрабатывается Парсером, причем для обращения к базе данных используется установленное подключение. В настоящей версии Парсера поддерживается подключение к MySQL Server, Oracle или источникам данных ODBC.

Замечание. Возможность работы с тем или иным сервером баз данных зависит от того, какой вариант Парсера вы используете. Из этого следует, что один и тот же шаблон не может обращаться, например, и к Oracle, и к MySQL.

Параметры подключения к MySQL:

user:password@host:port/database, где

- **user** — имя пользователя;
- **password** — пароль;
- **host** — имя или адрес компьютера в сети;
- **port** — номер порта (указывать не обязательно);
- **database** — имя базы данных.

Параметры подключения к серверу Oracle:

user:password@oracle_connection_string, где

- **user** — имя пользователя;
- **password** — пароль;
- **oracle_connection_string** — строка параметров для подключения к серверу Oracle.

Параметры для подключения к источникам данных ODBC:

user:password@odbc_datasource, где

- **user** — имя пользователя;
- **password** — пароль;

- **odbc_datasource** — имя источника данных.

Замечание. При работе с Oracle все SQL-запросы внутри оператора **server** выполняются в рамках одной транзакции. После успешного выполнения всех SQL-запросов транзакция фиксируется. При первой ошибке происходит откат транзакции, а последующие SQL-запросы игнорируются.

Замечание. Данные в базе данных MySQL должны быть представлены в кодировке koi8-r, а в Oracle и ODBC — в кодировке Windows-1251.

Замечание. В некоторых случаях бывает нужно обращаться из Windows-приложений к базе данных, управляемой сервером MySQL. При этом база данных может использоваться не только Парсером. В таком случае для работы с базой данных в кодировке koi8-r в настройках ODBC-драйвера следует задать соответствующее преобразование кодировок:

```
SQL command on connect: set CHARACTER SET cp1251_koi8.
```

Примеры

Подключение к серверу баз данных MySQL и вывод списка всех таблиц базы данных.

```
^server[site_user:JH12tyKNP@data/newspaperdb;
        ^sql[show tables]
        ^menu[^item[0]<BR>]
]
```

3.5.2. Выполнение SQL-запроса. Оператор sql

Формат вызова

```
^sql[запрос]

^sql[запрос;количество_записей]

^sql[запрос;количество_записей;пропустить]
```

Аргументы

запрос — запрос на языке SQL.

количество_записей — количество записей, которое будет взято из результатов запроса.

пропустить — количество пропускаемых записей.

Описание

Оператор отправляет запрос SQL-серверу и записывает полученные результаты в текущую таблицу. Столбцам таблицы присваиваются имена полей запроса. Если указано количество записей, допустим, равное **12**, то в текущую таблицу будут загружены только первые двенадцать записей. Если несколько первых

записей требуется пропустить, укажите их количество в качестве значения третьего аргумента.

Внимание! Вне зависимости от содержания SQL-запроса, при обработке оператора `sql` содержимое текущей таблицы теряется.

Замечание. Текст запроса обрабатывается на уровне обработки `sql` (см. п. 1.6).

Замечание. Не рекомендуется использовать запросы вида `SELECT *`, поскольку они могут привести к загрузке большого объема ненужных для работы данных. Кроме того, в большинстве случаев, формулируя SQL-запрос, кодер должен знать, какие данные он получит в ответ: он ведь собирается с ними дальше работать.

Примеры

Приведенный ниже фрагмент кода загружает в текущую таблицу первые 50 новостей, а затем выводит ссылки на них.

```
^context[news;
         ^server[site_user:JH12tyKNP@data/newspaperdb;
                 ^sql[SELECT id,title FROM news WHERE active=1;50]
                 ^menu[
<A HREF="details.html?id=^item[id]">^item[title]</A><BR>
                 ]
         ]
]
```

3.6. Таблицы

3.6.1. Назначение текущей таблицы. Оператор context

Формат вызова

```
^context[имя_таблицы;код]
```

Аргументы

имя_таблицы — имя таблицы, назначаемой текущей.

код — код, обрабатываемый Парсером.

Описание

Оператор назначает текущей указанную таблицу. Код обрабатывается Парсером.

Примеры

```
^context[data;
        ^menu[<A HREF=^item[id].html>^item[title]</A>]
]
```

3.6.2. Заполнение таблицы данными из кода. Оператор config

Формат вызова

```
^config[данные]

^config[команда1...командаN;данные]

^config[команда1...командаN]
```

Аргументы

данные — значения, загружаемые в таблицу.

команда1...командаN — режимы формирования таблицы.

Описание

Оператор заполняет текущую таблицу переданными значениями, которые должны располагаться так же, как в файле формата tab-delimited. А именно: значения, принадлежащие одной строке таблицы, отделяются друг от друга символом табуляции, а строки таблицы — символом перевода строки.

Аргумент **команда** задает режим заполнения таблицы. Этот аргумент может принимать следующие значения:

- **named** — первая строка значений содержит имена столбцов таблицы;
- **append** — данные будут добавлены в текущую таблицу последней строкой;

- **insert** — данные будут вставлены перед текущей строкой таблицы. Новая строка сделается текущей;
- **insert;номер_строки** — данные будут вставлены перед строкой с указанным номером;
- **remove** — удаление текущей строки таблицы. Данные можно не передавать;
- **remove;номер_строки** — удаление из таблицы строки с указанным номером. Данные можно не передавать;
- **remove;номер_строки;количество_строк** — удаление указанного количества строк, начиная со строки с указанным номером. Данные можно не передавать;
- **join;имя_таблицы** — добавить к текущей таблице строки из таблицы с указанным именем. Таблицы обязательно должны иметь одинаковую структуру. Данные можно не передавать.

Внимание! Закрывающую квадратную скобку следует ставить сразу после значения, расположенного на пересечении последней строки и последнего столбца. Если перенести ее на следующую строчку, то в контекст будет добавлена лишняя строка.

Замечание. Данные обрабатываются на уровне **config** (см. п. 1.6).

Примеры

Следующий фрагмент кода формирует таблицу из трех строк и трех столбцов.

```
^config[named;
last_name > date_of_birth > e_mail
Иванов >    20.10.65 >      ivanov@nechto.ru
Петров >    12.03.72 >      petrov@nechto.ru
Сидоров >   04.09.81 >      sidoroff@nechto.ru]
```

Столбцы таблицы получат имена **last_name**, **date_of_birth** и **e_mail**. Как имена столбцов, так и значения в столбцах должны быть отделены друг от друга символом табуляции.

3.6.3. Загрузка данных в таблицу. Оператор load

Формат вызова

```
^load[полное_имя_файла]
```

```
^load[named;полное_имя_файла]
```

Аргументы

полное_имя_файла — имя файла данных.

named — значение, которое указывает на то, что первая строка файла содержит имена столбцов.

Описание

Оператор загружает данные из файла в формате tab-delimited в текущую таблицу. Имя файла на диске должно начинаться символом подчеркивания, который не указывается в аргументе оператора. Если указан первый аргумент **named**, то первая строка файла интерпретируется как список имен столбцов.

Внимание! При загрузке данных в таблицу ее содержимое теряется.

Замечание. Пустые строки в файле данных игнорируются.

Замечание. Для того чтобы поместить в файл данных комментарий, можно поставить в первой позиции строки символ **#** («диез»), а за ним написать текст комментария. Строки, начинающиеся символом **#**, Парсер пропускает.

Примеры

Следующий фрагмент кода назначает текущей таблицу **data** и загружает в нее данные из файла **_data.cfg**.

```
^context[data;
         ^load[named;data.cfg]
]
```

3.6.4. Проверка таблицы на пустоту. Оператор empty

Формат вызова

```
^empty[код_если_нет_данных;код_если_есть_данные]
```

Аргументы

код_если_нет_данных — код, возвращаемый оператором при отсутствии данных в таблице.

код_если_есть_данные — код, возвращаемый оператором при наличии данных в таблице.

Описание

Оператор возвращает первый аргумент, если текущая таблицы пуста, и второй — в противном случае.

Примеры

Рассмотрим следующий фрагмент кода.

```
^context[data;
         ^config[]
         ^empty[^sendmail[to:admin@domain.ru
subject: Таблица data не содержит записей

Дорогой администратор, твоя таблица совсем                не содержит
данных, пользователи мучаются,                          почини, пожалуйста.
С уважением, твой навеки Робот
         ];
                  Данные успешно загружены
         ]
]
```

Обработка этого фрагмента приведет к отправке администратору сообщения по электронной почте. Если заполнить таблицу **data** данными, то сообщение отправлено не будет. Дело в том, что Парсер обрабатывает только те аргументы, значения которых необходимы для вычисления значения оператора.

3.6.5. Обращение к элементам таблицы. Оператор item

Формат вызова

```
^item[столбец]

^item[next;столбец]

^item[prev;столбец]

^item[имя_таблицы;столбец]
```

Аргументы

столбец — столбец, значение из которого требуется получить.

next, **prev** — указывает, что значение следует брать не из текущей строки таблицы, а из следующей или предыдущей.

имя_таблицы — имя таблицы, в том случае, если требуется получить значение не из текущей таблицы.

Описание

Оператор возвращает элемент, стоящий на пересечении указанного столбца и текущей строки таблицы. Если таблица не указана, то возвращаются данные из текущей таблицы. Для того чтобы получить значение не из текущей строки, а из предыдущей или следующей, укажите в качестве первого аргумента значения **prev** или **next** соответственно.

Замечание. При использовании аргументов **prev** и **next** считается, что таблица «закольцована». Вызов **^item[next;столбец]** для последней строки вернет значение из нулевой строки. И наоборот, вызов **^item[prev;столбец]** для нулевой строки вернет значение из последней.

Специальные форматы вызова оператора **item** используются для получения сведений о текущей таблице.

`^item[count]` — количество строк в текущей таблице.

`^item[current]` — номер текущей строки текущей таблицы (начиная с `0`).

`^item[number]` — номер текущей строки текущей таблицы, начиная с `1`.

Примеры

```
^item[title]
```

```
^item[prev;title]
```

```
^item[people;name]
```

```
^item[0]
```

3.6.6. Перебор строк таблицы. Оператор menu

Формат вызова

```
^menu[код]
```

```
^menu[код;разделитель]
```

Аргументы

`код` — код, выполняемый для каждой строки таблицы.

`разделитель` — код, размещаемый между результатами обработки строк.

Описание

Оператор выполняет переданный код последовательно для каждой строки текущей таблицы. Разделительный код вставляется между всеми непустыми результатами выполнения кода.

Замечание. Оператор `menu` не перемещает текущую строку таблицы, т. е. после его вызова текущей будет та же строка, что и до вызова.

Примеры

Следующий фрагмент кода создает таблицу и выводит ее содержимое в виде HTML-таблицы.

```
         ^config[named;
last_name > date_of_birth >                                        e_mail
Иванов >    20.10.1965 >      ivanov@nechto.ru
Петров >    12.03.1972 >      petrov@nechto.ru
Сидоров >   04.09.1981 >      sidoroff@nechto.ru]
         <TABLE BORDER=1>
         ^menu[<TR>
               <TD>^item[last_name]</TD>
               <TD>^item[date_of_birth]</TD>
               <TD>^item[e_mail]</TD>
               </TR>
         ]
         </TABLE>
]
```

3.6.7. Вывод данных в табличной форме. Оператор table

Формат вызова

```
^table[количество_столбцов;код]

^table[количество_столбцов;код;заполнитель]
```

Аргументы

количество_столбцов — количество столбцов в HTML-таблице.

код — код, формируемый для каждой строки таблицы.

заполнитель — данные, размещаемые в незаполненном до конца столбце HTML-таблицы.

Описание

Оператор располагает строки текущей таблицы в HTML-таблице с указанным количеством столбцов. Если в результате последние строки последнего столбца HTML-таблицы остаются пустыми, в них записывается заполнитель.

Примеры

Рассмотрим следующий код:

```
^context[data;
         ^config[named;id >  title
1 >   Новости
2 >   Погода
3 >   Публикации
4 >   Ссылки
5 >   Контакты]
         <TABLE>
         <TR>
         ^table[2;
                <TD><A HREF="^item[id].html">^item[title]</A></TD>;
                 ^;
         ]
         </TR></TABLE>
]
```

Обработка приведенного выше кода даст следующий результат:

```
<TABLE>
<TR>
<TD><A HREF="1.html">Новости</A></TD>
<TD><A HREF="3.html">Ссылки</A></TD>
</TR><TR>
<TD><A HREF="2.html">Погода</A></TD>
<TD><A HREF="4.html">Контакты</A></TD>
</TR><TR>
<TD><A HREF="5.html">Публикации</A></TD>
<TD> </TD></TR></TABLE>
```

3.6.8. Поиск в таблице. Оператор locate

Формат вызова

```
^locate[имя_столбца;образец]
```

Аргументы

имя_столбца — имя столбца, в котором выполняется поиск.

образец — искомое значение.

Описание

Оператор ищет в указанном столбце текущей таблицы значение, равное искомому значению. Строка таблицы, в которой обнаруживается такое значение, делается текущей. Если искомое значение найти не удается, текущей делается первая строка таблицы.

Примеры

```
^context[data;
         ^locate[last_name;Иванов]
         ^eq[^item[last_name];Иванов;
             Запись найдена;
             Запись не найдена
         ]
]
```

3.6.9. Перемещение по таблице. Оператор shift

Формат вызова

```
^shift[количество_строк]
```

Аргументы

количество_строк — «расстояние», на которое надо переместиться по таблице.

Описание

Оператор перемещает указатель текущей строки текущей таблицы на заданное количество строк вниз. Если аргумент отрицательный, то указатель перемещается вверх. Указатель перемещается по таблице циклически, т. е., достигнув последней строки, перемещается к первой и наоборот.

Примеры

```
^shift[3]
```

```
^shift[-4]
```

3.6.10. Поворот текущей строки таблицы. Оператор flip

Формат вызова

```
^flip[имя_таблицы]
```

Аргументы

имя_таблицы — имя таблицы, над которой выполняется операция.

Описание

Оператор преобразует текущую строку таблицы в столбец. В результате образуется таблица, состоящая из одного столбца. Предыдущее содержимое таблицы теряется.

Примеры

Рассмотрим следующий фрагмент кода:

```
^context[data;
         ^config[1 > 2 > 3
4 > 5 > 6]
^flip[data]
^menu[^item[0]]
]
```

Обработка этого фрагмента даст такой результат: `123`.

3.6.11. Сортировка таблицы. Оператор sort

Формат вызова

```
^sort[столбец_сортировки;параметры_сортировки]
```

Аргументы

столбец_сортировки — столбец, по которому требуется отсортировать таблицу.

параметры_сортировки — параметры, задающие способ сортировки.

Описание

Оператор сортирует текущую таблицу по указанному столбцу. Способ сортировки определяется параметрами. Параметры всегда состоят из двух символов: первый обозначает направление сортировки, а второй — способ сравнения значений.

Первый символ:

`+` — по возрастанию ключа;
`-` — по убыванию ключа.

Второй символ:

`A` — лексикографический порядок;
`N` — значения сравниваются как действительные числа;
`D` — значения сравниваются как целые числа.

Примеры

```
^sort[^item[0];+A]
```

3.6.12. Запись данных из таблицы в файл. Оператор save

Формат вызова

```
^save[имя_файла]

^save[имя файла;текст]
```

Аргументы

`имя_файла` — имя файла, в который производится запись.

`текст` — текст, записываемый в файл.

Описание

Если используется первый формат вызова, то оператор записывает данные из текущей таблицы в файл с указанным именем. Если используется второй формат вызова, то в файл записывается переданный текст. Имя файла на диске должно начинаться символом подчеркивания, который не указывается в аргументе оператора.

Внимание! Используя оператор **`save`**, следует иметь в виду, что несколько посетителей сайта могут инициировать одновременную запись данных в один и тот же файл. Последствия этого непредсказуемы. Поэтому, если речь идет не о простейших случаях, мы рекомендуем отказаться от использования файлов в формате tab-delimited и работать с базами данных.

Примеры

```
^context[data;^save[data.cfg]]

^context[data;^save[data.cfg;Жил-был у бабушки серенький козлик]]
```

3.7. Строки

3.7.1. Получение длины строки. Оператор length

Формат вызова

```
^length[строка]
```

Аргументы

строка — строка, длину которой нужно определить.

Описание

Оператор возвращает длину строки.

Примеры

Вызов `^length[12345678]` вернет `8`.

3.7.2. Выделение подстроки слева. Оператор left

Формат вызова

```
^left[строка;длина_подстроки]
```

Аргументы

строка — строка, левую часть которой нужно получить.

длина_подстроки — количество символов в выделяемой левой части.

Описание

Оператор возвращает указанное количество первых символов строки. Если длина строки меньше, возвращается вся исходная строка.

Примеры

Вызов
```
^left[И он к устам моим приник и вырвал грешный мой язык;33]
```

вернет
```
И он к устам моим приник и вырвал
```

3.7.3. Выделение подстроки справа. Оператор right

Формат вызова

```
^right[строка;длина_подстроки]
```

Аргументы

строка — строка, правую часть которой нужно получить.

длина_подстроки — количество символов в выделяемой правой части.

Описание

Оператор возвращает указанное количество последних символов строки. Если длина строки меньше, возвращается вся исходная строка.

Примеры

Вызов `^left[перестройка;7]` вернет **стройка**.

3.7.4. Выделение подстроки в середине. Оператор mid

Формат вызова

```
^mid[строка;p;n]
```

Аргументы

строка — строка.

n — позиция первого символа выделяемой подстроки.

p — длина выделяемой подстроки.

Описание

Оператор возвращает **N** символов строки, начиная с позиции **P** (позиции отсчитываются с нуля). Если **P** меньше **0**, то будет возвращено **N+P** первых символов строки. Если **P+N** больше длины строки, то будут возвращены все символы строки, начиная с позиции **P**.

Примеры

Вызов `^mid[перестройка;4;5]` вернет **строй**.

Вызов `^mid[перестройка;-2;11]` вернет **перестрой**.

3.7.5. Поиск подстроки в строке. Оператор findstr

Формат вызова

```
^findstr[подстрока;строка]
```

Аргументы

строка — строка, в которой проводится поиск.

подстрока — подстрока, которую нужно найти в строке.

Описание

Оператор возвращает позицию первого символа подстроки в строке. Позиции отсчитываются с нуля. Если подстрока не найдена, оператор возвращает `-1`.

Примеры

Вызов `^findstr[арест;Бухарест]` вернет `3`.

Вызов `^findstr[иголка;стог сена]` вернет `-1`.

3.7.6. Получение символа по ASCII-коду. Оператор char

Формат вызова

```
^char[ASCII-код]
```

Аргументы

`ASCII-код` — десятичный код символа.

Описание

Оператор возвращает символ с указанным десятичным ASCII-кодом.

Примеры

Вызов `^char[49]` вернет `1`.

3.7.7. Получение ASCII-кода символа. Оператор ascii

Формат вызова

```
^ascii[символ]
```

Аргументы

`символ` — символ, ASCII-код которого нужно получить.

Описание

Оператор возвращает десятичный ASCII-код символа. Если значение аргумента — строка длиной более одного символа, то возвращается ASCII-код первого символа.

Примеры

Вызов `^char[1]` вернет `49`.

3.7.8. Разбиение строки на подстроки слева направо. Оператор lsplit

Формат вызова

```
^lsplit[строка;разделитель]
```

Аргументы

строка — строка, подстроки которой нужно записать в таблицу.

разделитель — символ, разделяющий подстроки.

Описание

Оператор выделяет в переданной строке подстроки, отделенные друг от друга разделителем. Полученные подстроки записываются в столбцы текущей таблицы слева направо. Предыдущее содержимое таблицы теряется.

Замечание. Аргументы обрабатываются на уровне **config** (см. п. 1.6).

Примеры

```
^lsplit[Иванов/12.02.70/ivanov@domain.com;/]
Фамилия: ^item[0]<BR>
День рождения: ^item[1]<BR>
E-mail: ^item[2]
```

3.7.9. Разбиение строки на подстроки справа налево. Оператор rsplit

Формат вызова

```
^rsplit[строка;разделитель]
```

Аргументы

строка — строка, подстроки которой нужно записать в таблицу.

разделитель — символ, разделяющий подстроки.

Описание

Оператор выделяет в переданной строке подстроки, отделенные друг от друга разделителем. Полученные подстроки записываются в столбцы текущей таблицы справа налево. Предыдущее содержимое таблицы теряется.

Замечание. Аргументы обрабатываются на уровне **config** (см. п. 1.6).

Примеры

```
^lsplit[rabinovich@domain.com/03.07.70/Рабинович;/]
Фамилия: ^item[0]<BR>
День рождения: ^item[1]<BR>
E-mail: ^item[2]
```

3.7.10. Преобразование к верхнему регистру. Оператор toupper

Формат вызова

```
^toupper[строка]
```

Аргументы

строка — строка, которую нужно преобразовать к верхнему регистру.

Описание

Оператор преобразует все буквы указанной строки в прописные и возвращает полученную строку.

Примеры

Вызов `^toupper[Всем привет!]` вернет **ВСЕМ ПРИВЕТ!**

3.7.11. Преобразование к нижнему регистру. Оператор tolower

Формат вызова

```
^tolower[строка]
```

Аргументы

строка — строка, которую нужно преобразовать к нижнему регистру.

Описание

Оператор преобразует все буквы указанной строки в строчные и возвращает полученную строку.

Примеры

Вызов `^tolower[Всем привет!]` вернет **всем привет!**

3.7.12. Обрезка строки. Оператор truncate

Формат вызова

```
^truncate[текст;длина]
```

Аргументы

строка — строка, которую нужно обрезать.

длина — требуемая длина строки.

Описание

Оператор укорачивает переданную строку, если количество символов в ней превышает указанную длину. В конце укороченной строки добавляются три точки, которые при вычислении длины строки не учитываются.

Примеры

Вызов

```
^truncate[Вхожу я в контору, а там меня поджидает мешок с деньгами;24]
```

вернет строку

```
Вхожу я в контору, а там...
```

3.7.13. Преобразование служебных символов в теги. Оператор unescape_br

Формат вызова

```
^unescape_br[текст]
```

Аргументы

текст — текст, в котором нужно преобразовать служебные символы в теги HTML.

Описание

Оператор заменяет в тексте одинарные символы перевода строки тегом **
, а двойные — тегом **<P>.

Примеры

Вызов

```
^unescape_br[первая строка
вторая строка

третья строка
]
```

вернет: **первая строка
вторая строка<P>третья строка**.

3.8. Формы, запросы, переменные окружения

3.8.1. Значения в полях формы. Оператор form

Формат вызова

```
^form[имя_поля]
```

Аргументы

имя_поля — имя поля формы или параметра, значение которого нужно получить.

Описание

Оператор возвращает значение указанного поля формы или параметра, переданного через URL в формате **?name=value**. Данный оператор используется на страницах, которые вызываются в качестве скриптов. Оператор корректно работает при использовании обоих методов передачи данных, GET и POST.

Внимание! Для получения целочисленных значений категорически рекомендуется использовать оператор **number** (см. п. 3.8.2). В противном случае вы рискуете получить строку там, где ожидается число.

Примеры

На странице **form.html** пользователь может ввести в поля формы те или иные значения. После нажатия на кнопку **OK** будет загружена страница **showvalues.html**, отображающая введенные значения.

form.html

```
<HTML>
<HEAD><TITLE>Форма</TITLE></HEAD>
<BODY>
<FORM ACTION="showvalues.hrml">
   <P>Имя:      <INPUT NAME="first_name"></P>
   <P>Фамилия: <INPUT NAME="last_name"></P>
   <P><INPUT TYPE=SUBMIT VALUE="OK"></P>
</FORM>
</BODY>
</HTML>
```

showvalues.html

```
<HTML>
<HEAD><TITLE>Значения полей формы</TITLE></HEAD>
<BODY>
<P>Имя:      ^form[first_name]</P>
<P>Фамилия: ^form[last_name]</P>
</BODY>
</HTML>
```

Для вызова страницы **showvalues.html** можно также использовать URL с параметрами.

```
<HTML>
<HEAD><TITLE>URL с параметрами</TITLE></HEAD>
<BODY>
<A HREF="showvalues.html?first_name=John&last_name=Doe">John Doe</A>
</BODY>
</HTML>
```

3.8.2. Численные значения в полях формы. Оператор number

Формат вызова

```
^number[имя_поля]
```

Аргументы

имя_поля — имя поля, значение которого нужно получить.

Описание

Оператор делает то же, что и оператор **form**, но для целочисленных значений. Если значение, введенное в поле формы, не является числовым, оператор возвращает значение 0.

Примеры

```
^number[age]
```

3.8.3. Значение переменной QUERY_STRING. Оператор query

Формат вызова

```
^query[]
```

Аргументы

Отсутствуют.

Описание

Оператор возвращает строку запроса (значение переменной окружения QUERY_STRING), передаваемую в URI.

Замечание. Если значения параметров, которые требуется передать вызываемому скрипту, содержат символы, запрещенные протоколом HTTP, следует преобразовать эти значения с помощью оператора **mangle** (см. п. 3.8.8).

Примеры

На странице **query.html** посетитель может нажать на ссылку. После этого будет загружена страница **showstring.html**, отображающая переданную ей строку запроса.

query.html

```
<HTML>
<HEAD><TITLE>Страница со строкой запроса</TITLE></HEAD>
<BODY>
<A HREF="showstring.html?the_string">Показать строку запроса</A>
</BODY>
</HTML>
```

showstring.html

```
<HTML>
<HEAD><TITLE>Строка запроса</TITLE></HEAD>
<BODY>
Строка запроса: ^query[]
</BODY>
</HTML>
```

3.8.4. Переменные окружения веб-сервера. Оператор env

Формат вызова

```
^env[имя_переменной_окружения]
```

Аргументы

имя_переменной_окружения — имя переменной окружения, задаваемой веб-сервером, значение которой требуется получить.

Описание

Оператор возвращает значение указанной переменной окружения, задаваемой веб-сервером.

Замечание. Со списком стандартных переменных окружения можно ознакомиться по адресу http://www.w3c.org/cgi. Веб-сервер Apache задает значения ряда дополнительных переменных.

Примеры

Вызов `^env[SERVER_NAME]` вернет доменное имя веб-сервера, например `www.design.ru`.

3.8.5. URI страницы. Оператор uri

Формат вызова

```
^uri[]
```

Аргументы

Отсутствуют.

Описание

Оператор возвращает URI текущей страницы.

Примеры

Предположим, адрес страницы **http://www.server.ru/abc/def.html?ghi**. Тогда помещенный в ее код вызов **^uri[]** вернет значение **/abc/def.html?ghi**.

3.8.6. Формирование множества значений поля. Оператор combine

Формат вызова

```
^combine[имя_поля_формы]

^combine[имя_поля_формы;символ_разделитель]
```

Аргументы

имя_поля_формы — имя поля формы, из значений которого требуется составить множество.

символ_разделитель — символ, которым будут разделены элементы множества.

Описание

Оператор возвращает значения указанного поля формы, разделенные заданным символом. По умолчанию в качестве символа-разделителя используется | (вертикальная черта).

Примеры

После нажатия на кнопку **OK** на странице будет отображено значение **1|2**.

```
<HTML>
<HEAD><TITLE>Составление множества значений поля</TITLE></HEAD>
<BODY>
<FORM>
   <P><INPUT TYPE=CHECKBOX NAME=EXP VALUE=1></P>
   <P><INPUT TYPE=CHECKBOX NAME=EXP VALUE=2></P>
   <P><INPUT TYPE=SUBMIT VALUE="OK"></P>
</FORM>
Значения поля exp: ^combine[exp]
</BODY>
</HTML>
```

3.8.7. Определение типа и версии браузера. Оператор browser

Формат вызова

```
^browser[тип_браузера;номер_версии]
```

Аргументы

тип_браузера — предполагаемый тип браузера.

номер_версии — предполагаемая минимальная версия браузера.

Описание

Оператор возвращает номер версии браузера, если одновременно выполняются следующие условия:

- тип браузера, в который загружается страница, совпадает с типом, указанным в первом аргументе;
- номер версии, в который загружается страница, не превышает указанного во втором аргументе.

В остальных случаях выдается пустая строка.

Допустимые значения типа браузера: **ie** (для Microsoft Internet Explorer) и **nn** (для Netscape Navigator).

Примеры

Предположим, страница загружается в Internet Explorer 4.0. В этом случае вызов **^browser[ie;4]** вернет значение **4**, а вызовы **^browser[nn;4]** или **^browser[ie;5]** — пустую строку.

3.8.8. Кодирование строки для URL. Оператор mangle

Формат вызова

```
^mangle[строка]
```

Аргументы

строка — кодируемая строка.

Описание

Оператор кодирует строку в соответствии с требованиями протокола HTTP для использования в качестве части URL.

Замечание. С русской версией веб-сервера Apache оператор работает корректно.

Примеры

В коде страницы имеется вызов **^mangle[Москва]**. Тогда, если посетитель сайта использует компьютер Macintosh, такой вызов вернет значение **%8C%EE%F1%EA%E2%E0**. Если же посетитель работает в Windows, оператор вернет **%CC%EE%F1%EA%E2%E0**.

3.8.9. Формирование HTTP-заголовка ответа. Оператор header

Формат вызова

```
^header[имя_атрибута;значение]
```

Аргументы

имя_атрибута — имя формируемого HTTP-заголовка.

значение — значение формируемого HTTP-заголовка.

Описание

Оператор выдает программе-клиенту HTTP-заголовок с указанным значением, а также позволяет изменять заголовок **Content-type**.

Примеры

Вызов `^header[refresh;5^;url=/]` вернет заголовок **refresh** со значением **5;url=/**. Обратите внимание: во втором аргументе встречается точка с запятой, предваряемая «птичкой».

3.9. Сервисы и протоколы интернета

3.9.1. Запись и чтение cookie. Оператор cookie

Формат вызова

```
^cookie[имя_cookie]

^cookie[имя_cookie;значение]

^cookie[имя_cookie;значение;срок_хранения]
```

Аргументы

имя_cookie — имя cookie.

значение — значение, присваиваемое cookie.

срок_хранения — срок хранения cookie.

Описание

Оператор **cookie** с одним аргументом возвращает значение cookie с переданным именем. Оператор **cookie** с двумя аргументами сохраняет в cookie с указанным именем указанное значение. Третий аргумент — срок хранения записанного cookie (по умолчанию устанавливается срок, равный 90 дням).

Если в качестве третьего аргумента введена строка **session**, то cookie хранится в течение одной сессии. Если задано целочисленное значение, то cookie хранится указанное количество дней.

Примеры

`^cookie[test;5]` — записать в cookie **test** значение 5 на 90 дней.

`^cookie[test;5;7]` — записать в cookie **test** значение 5 на 7 дней.

`^cookie[test;5;session]` — записать в cookie **test** значение 5 на время сессии.

`^cookie[test]` — прочитать значение cookie **test** и выдать в качестве результата.

3.9.2. Запуск CGI-скриптов. Оператор exec

Формат вызова

```
^exec[URI]

^exec[URI;QUERY_STRING]

^exec[URI;QUERY_STRING;PATH_INFO]
```

Аргументы

URI — URI запускаемого CGI-скрипта.

QUERY_STRING — задаваемая переменная окружения **QUERY_STRING**.

PATH_INFO — задаваемая переменная окружения **PATH_INFO**.

Описание

Оператор вызывает CGI-скрипт с указанным URI. Переменным окружения сервера **QUERY_STRING** и **PATH_INFO** для вызываемого скрипта присваиваются указанные значения.

Примеры

```
^exec[/cgi-bin/test.pl;var1=1&var2=2]
```

3.9.3. Отправка сообщений по электронной почте. Оператор sendmail

Формат вызова

```
^sendmail[сообщение]
```

Аргументы

сообщение — заголовок и текст передаваемого сообщения.

Описание

Оператор отправляет по электронной почте указанное сообщение. В среде Unix в зависимости от версии ОС вызывается команда **/usr/lib/sendmail -t** или **/usr/sbin/sendmail -t**. Сообщение передается на **stdin** в формате, ожидаемом программой **sendmail**. В среде Win32 должен быть предварительно определен макрос **SMTP_SERVER**, содержащий адрес SMTP-сервера. На этот сервер сообщение будет отправлено по протоколу SMTP.

Замечание. Параметр обрабатывается на уровне обработки **config** (см. п. 1.6).

Замечание. Текст сообщения перед передачей всегда перекодируется в кодировку koi8-r. Кодировку сообщения рекомендуется указывать в его заголовке.

Примеры

```
^sendmail[from: support@parser.ru
to: webmaster@parser.ru
subject: Я сделал это!
content-type: text/plain^; charset="koi8-r"

Теперь я могу отправлять почту!
]
```

3.9.4. Вложение данных в электронное сообщение. Оператор uuencode

Формат вызова

```
^uuencode[данные;имя_файла]
```

Аргументы

данные — данные, которые нужно представить в формате uuencode.

имя_файла — имя файла, которое будет указано в заголовке последовательности uuencode.

Описание

Оператор представляет текст в формате uuencode с указанным именем файла в заголовке.

Замечание. Текст обрабатывается на уровне обработки **config** (см. п. 1.6).

Примеры

```
^sendmail[`from: sender@here.ru
to: receiver@there.ru
subject: test text attachment
content-type: multipart/mixed;
boundary="----=_NextPart_000_0005_01C0AB8C.159C2660"

This is a multi-part message in MIME format.

------=_NextPart_000_0005_01C0AB8C.159C2660
Content-Type: text/plain; charset="koi8-r"

Прилагаю тот HTML, о котором мы говорили...

------=_NextPart_000_0005_01C0AB8C.159C2660
Content-Type: text/html; name="chapter01.html"
Content-Disposition: attachment; filename="chapter01.html"
Content-Transfer-Encoding: uuencode

^uuencode[<TITLE>Глава 1</TITLE>...;chapter01.html]

------=_NextPart_000_0005_01C0AB8C.159C2660-
`]
```

3.10. Файлы

3.10.1. Выделение имени файла из пути. Оператор name

Формат вызова

```
^name[полное_имя_файла]
```

Аргументы

полное_имя_файла — путь, из которого требуется выделить имя файла.

Описание

Оператор возвращает имя файла без пути и расширения.

Примеры

Вызов `^name[/dir/subdir/]` вернет значение **index**.

3.10.2. Проверка существования файла на диске. Оператор -f

Формат вызова

```
^-f[имя_файла]
```

Аргументы

имя_файла — имя файла, который нужно найти.

Описание

Оператор возвращает путь, если файл с указанным именем существует. В противном случае возвращает пустую строку.

Примеры

```
^var[src;^-f[image.gif]]
^ifdef[^var[src];<IMG SRC=^var[src]>;]
```

3.10.3. Получение информации о файле. Оператор stat

Формат вызова

```
^stat[что_вернуть;путь]

^stat[fsize;путь;обозначение_байта;обозначене_Кб;обозначение_Мб]
```

Аргументы

что_вернуть — тип возвращаемого значения.

путь — путь к файлу.

обозначение_байта — обозначение единицы измерения «байт».

обозначене_Кб — обозначение единицы измерения «килобайт».

обозначение_Мб — обозначение единицы измерения «мегабайт».

Описание

Оператор возвращает информацию о файле.

Предусмотрены следующие типы информации:

size — размер файла в байтах.

fsize — форматированный размер файла. Если запрашивается этот тип информации, то оператор имеет не два аргумента, а пять. Результатом вызова является размер файла в мегабайтах, килобайтах или байтах. Полученное значение округляется до целого. Для обозначения единицы измерения используется указанное обозначение.

atime — время последнего обращения.

mtime — время последнего изменения.

ctime — время создания.

Примеры

```
Файл: <A HREF="big.zip">big.zip</A><BR>
Размер: ^stat[fsize;big.zip;bytes;Kb;Mb]<BR>
Последнее обновление: ^stat[mtime;big.zip]
```

3.10.4. Поиск файла. Оператор find

Формат вызова

```
^find[имя_файла]
```

```
^find[имя_файла;сообщение]
```

Аргументы

имя_файла — имя искомого файла.

сообщение — текст, который вернет оператор в том случае, если файл не будет найден.

Описание

Оператор возвращает относительное имя файла при условии, что он существует и находится на текущем или более высоком уровне. В противном случае оператор возвращает сообщение, задаваемое вторым аргументом.

Примеры

```
<IMG SRC="^find[section.gif;default.gif]">
```

Файл **section.gif** будет найден даже в том случае, если сам шаблон находится в каталоге, расположенном «глубже», чем графический файл.

3.10.5. Загрузка в таблицу оглавления каталога. Оператор index

Формат вызова

```
^index[путь]

^index[путь;шаблон]
```

Аргументы

путь — путь к каталогу.

шаблон — шаблон для отбора файлов и каталогов.

Описание

Оператор загружает в текущую таблицу список файлов и каталогов, находящихся в указанном каталоге. Если указан шаблон, то в список попадут только те файлы и каталоги, имена которых соответствуют шаблону. Список файлов и каталогов сортируется в лексикографическом порядке.

Замечание. Шаблон — это *регулярное выражение*, соответствующее стандарту POSIX 1003.2, раздел 2.8, частичный перевод которого приведен в Приложении C. В дополнение к возможностям, которые устанавливает этот стандарт, в Парсере предусмотрены следующие обозначения:

**** — обратная косая черта («бэк-слэш»);
\t — символ табуляции;
\d — любая цифра от **0** до **9**;
\s — пробел, символ табуляции, символ перевода строки.

Примеры

```
^context[dir;
         ^index[pictures;\.(gif|jpg)$]
         Картинки в каталоге pictures:<BR>
         ^menu[^item[0]<BR>]
]
```

3.10.6. Выгрузка файла с сервера. Оператор download

Формат вызова

```
^download[полное_имя_файла]

^download[полное_имя_файла;локальное_имя_файла]

^download[полное_имя_файла;локальное_имя_файла;mime-type]
```

Аргументы

полное_имя_файла — полное имя выгружаемого файла.

локальное_имя_файла — имя файла, под которым посетителю предлагается записать файл на локальный диск.

mime-type — тип данных.

Описание

Оператор выдает программе-клиенту файл с указанным полным именем. Если не указано локальное имя файла, то используется имя выгружаемого файла без пути. Тип данных, если он не указан, автоматически определяется Парсером по расширению файла. Таблицу соответствий типов расширениям располагают в макросе **MIME_TYPES**. Таблица имеет следующий формат.

```
расширение_имени_файла_1 mime-type_1
...
расширение_имени_файла_N mime-type_N
```

Если тип данных не удается определить по таблице, используется тип **application/octet-stream**. Ниже приведен типичный текст макроса **MIME_TYPES**.

```
@MIME_TYPES[] определения mime-type для ^^download
zip             application/zip
doc             application/msword
xls             application/vnd.ms-excel
pdf             application/pdf
ppt             application/powerpoint
rtf             application/rtf
gif             image/gif
jpg             image/jpeg
png             image/png
tif             image/tiff
html            text/html
htm             text/html
txt             text/plain
mts             application/metastream
mid             audio/midi
midi            audio/midi
p3              audio/mpeg
ram             audio/x-pn-realaudio
rpm             audio/x-pn-realaudio-plugin
ra              audio/x-realaudio
wav             audio/x-wav
au              audio/basic
mpg             video/mpeg
avi             video/x-msvideo
mov             video/quicktime
swf             application/x-shockwave-flash
```

Примеры

```
^download[/restricted/file.zip;special_user]
```

3.10.7. Загрузка файла на сервер. Оператор upload

Формат вызова

```
^upload[поле_формы;что_вернуть]
```

```
^upload[поле формы;имя_файла]
```

Аргументы

поле_формы — имя поля формы типа **FILE**.

что_вернуть — тип возвращаемого значения.

имя_файла — имя копируемого файла.

Описание

Если аргумент **что_вернуть** имеет одно из значений **name**, **size** или **value**, то оператор вернет соответственно имя, размер или содержимое файла, «введенного» пользователем в это поле. В противном случае значение второго аргумента будет интерпретироваться как имя файла. Под этим именем «введенный» в поле файл будет загружен на сервер.

Примеры

```
<FORM METHOD=POST ENCTYPE="MULTIPART/FORM-DATA">
      <INPUT TYPE=FILE NAME=IMAGE>
      <INPUT TYPE=SUBMIT>
</FORM>
^ifdef[^form[image];
       ^upload[image;/upload/^upload[image;name]
]
Записан файл ^upload[image;name] объемом ^upload[image;size] байт.
;]
```

3.10.8. Вложение файла в формате uuencode. Оператор upload

Формат вызова

```
^upload[поле_формы;uuencode;имя_файла]
```

Аргументы

поле_формы — имя поля формы типа **FILE**.

uuencode — обозначает, что используется именно эта форма оператора upload.

имя_файла — имя файла, указываемое в заголовке последовательности uuencode.

Описание

Оператор преобразует файл, «введенный» в поле формы, в формат uuencode и возвращает полученную последовательность. В заголовке последовательности указывается имя файла, переданное в аргументе **имя_файла**.

Примеры

```
^sendmail[to: user@domain.ru
from: mymail@domain.ru
subject: Веселые картинки
Content-Type: multipart/mixed^;
    boundary="----=_NextPart_000_009C_01BEA7AE.47C51900"

This is a multi-part message in MIME format.

------=_NextPart_000_009C_01BEA7AE.47C51900
Content-Type: text/plain^; charset=koi8-r

Вот картинка, которую ты так давно просил.

------=_NextPart_000_009C_01BEA7AE.47C51900
Content-Type: text/html^; name="^upload[picture;name]"
Content-Disposition: attachment^; filename="^upload[picture;name]"
Content-Transfer-Encoding: uuencode

^upload[picture;uuencode;^upload[picture;name]]

------=_NextPart_000_009C_01BEA7AE.47C51900--
]
```

3.10.9. Удаление файла. Оператор unlink

Формат вызова

```
^unlink[имя_файла]
```

Аргументы

имя_файла — имя удаляемого файла.

Описание

Оператор удаляет файл с указанным именем.

Примеры

```
^unlink[delme.txt]
```

3.11. Поиск и замена с использованием регулярных выражений

3.11.1. Поиск подстроки по шаблону. Оператор match

Формат вызова

```
^match[шаблон;настройки;строка]
```

Аргументы

шаблон — шаблон для поиска.

настройки — символы, задающие режим поиска.

строка — строка, в которой проводится поиск.

Описание

Оператор ищет в строке подстроку, соответствующую шаблону. Шаблон — это *регулярное выражение*, соответствующее стандарту POSIX 1003.2, раздел 2.8, частичный перевод которого приведен в Приложении C. В дополнение к возможностям, которые устанавливает этот стандарт, в Парсере предусмотрены следующие обозначения:

`\\` — обратная косая черта («бэк-слэш»);
`\t` — символ табуляции;
`\d` — любая цифра от `0` до `9`;
`\s` — пробел, символ табуляции, символ перевода строки.

Предусмотрены следующие режимы поиска:

i — не учитывать регистр;
n — исключить конец строки из заданного в шаблоне поиска «любого символа».

Поскольку символы ^ и $ используются в Парсере, в шаблоне вместо символа ^ используется строка ^^ (см. п. 1.2), а вместо символа $ — строка $$ (см. п. 1.3).

Если в шаблоне не используются круглые скобки, то при обнаружении строки, соответствующей шаблону, оператор возвращает значение **yes**. В противном случае оператор возвращает пустую строку.

Если в шаблоне используются круглые скобки, то содержимое текущей таблицы заменяется результатами поиска, а оператор возвращает пустую строку. При этом подстрока, отвечающая части шаблона, заключенной в круглые скобки, заносится в столбец с именем вида **\позиция**, где **позиция** — это номер открывающей круглой скобки. Например, вызов **^match[(.(.));;ab]** запишет в столбец **\1** текущей таблицы значение **ab**, а в столбец **\2** — **b**.

Начало строки до подстроки, отвечающей шаблону, заносится в столбец **prematch**. Например, вызов **^match[b.;;abcd]** запишет в столбец **prematch** значение **a**.

Подстрока, отвечающая шаблону, заносится в столбец **match**. Например, вызов **^match[b.;;abcd]** запишет в столбец **match** значение **bc**.

Конец строки после подстроки, отвечающей шаблону, заносится в столбец **postmatch**. Например, вызов **^match[b.;;abcd]** запишет в столбец **postmatch** значение **d**.

Примеры

Вызов **^match[^^apple.*;;apple169]** вернет значение **yes**.

```
^context[temp;
^match[([a-z]+)([^^a-z]+)$$;;apple169]
^item[\2]
]
```

Обработка приведенного выше фрагмента вернет **169**.

3.11.2. Замена подстроки, отвечающей шаблону. Оператор match

Формат вызова

^match[шаблон;настройки;замена;строка]

Аргументы

шаблон — шаблон для поиска.

настройки — символы, задающие режим поиска.

замена — подстрока, на которую нужно заменить подстроку, отвечающую шаблону.

строка — строка, в которой проводится поиск и замена.

Описание

Оператор ищет в строке подстроку, отвечающую шаблону, заменяет ее другой подстрокой и возвращает полученный результат. Шаблон поиска устроен так же, как у оператора **match** с тремя аргументами (см. п. 3.11.1). В дополнение к режимам поиска, предусмотренным в операторе **match** с тремя аргументами, добавляется режим **g**, подразумевающий замену всех найденных вхождений.

Примеры

Вызов `^match[apple;g;orange;I like apples]` вернет **I like oranges**.

3.12. Множества

3.12.1. Проверка вхождения элемента во множество. Оператор optionset

Формат вызова

```
^optionset[множество;элемент]
```

Аргументы

множество — множество, принадлежность элемента которому проверяется.

элемент — элемент, о котором требуется узнать, принадлежит ли он множеству.

Описание

Оператор возвращает значение **yes**, если указанный элемент принадлежит множеству. Если элемент не принадлежит множеству, возвращается пустая строка. Множество представляется в виде списка, элементы которого разделены символом | (вертикальная черта).

Примеры

Вызов `^ifdef[^optionset[1|2|3;3];Нашли;Не нашли]` вернет значение **Нашли**.

3.12.2. Удаление элемента из множества. Оператор optionclear

Формат вызова

```
^optionclear[множество;элемент]
```

Аргументы

множество — множество, из которого нужно удалить элемент.

элемент — удаляемый элемент.

Описание

Оператор удаляет из множества указанный элемент и возвращает полученную строку. Если элемент не принадлежит множеству, то результат равен исходному множеству.

Примеры

Вызов `^optionclear[1|2|3|4;3]` вернет `1|2|4`.

3.12.3. Пересечение множеств. Оператор optionand

Формат вызова

```
^optionand[множество1;множество2]
```

Аргументы

множество1, **множество2** — множества, пересечение которых нужно построить.

Описание

Оператор возвращает пересечение указанных множеств.

Примеры

Вызов `^optionand[a|b|c;b|c|d]` вернет `b|c`.

3.12.4. Объединение множеств. Оператор optionor

Формат вызова

```
^optionor[множество1;множество2]
```

Аргументы

множество1, **множество2** — множества, объединение которых нужно построить.

Описание

Оператор возвращает объединение указанных множеств.

Примеры

Вызов `^optionor[a|b;b|c]` вернет `a|b|c`.

3.12.5. Подсчет элементов множества. Оператор optioncount

Формат вызова

```
^optioncount[множество]
```

Аргументы

множество — множество, количество элементов которого нужно подсчитать.

Описание

Оператор возвращает количество элементов множества.

Примеры

Вызов `^optioncount[a|b|c]` вернет `3`.

3.13. Календарь

3.13.1. Получение текущей даты. Оператор date

Формат вызова

```
^date[форматная_строка]
```

Аргументы

форматная_строка — строка, определяющая форму представления даты.

Описание

Оператор возвращает текущую дату и/или текущее время. Способ представления даты и времени определяется форматной строкой.

В форматной строке специальные обозначения, предваряемые символом % (знак процента), заменяются значениями даты или времени. Остальной текст оставляется без изменений. Если необходимо использовать символ `%`, он удваивается.

Специальные обозначения:

- `%a`, `%A` — английское название дня недели, сокращенное или полное соответственно.
- `%b`, `%B` — английское название месяца, сокращенное или полное соответственно.
- `%c` — дата и время.
- `%C` — дата и время в длинном формате представления времени и даты.
- `%d` — день месяца (`01-31`).
- `%H` — час (`00-23`).
- `%I` — час (`00-12`).
- `%j` — номер текущего дня года (`001-366`).
- `%m` — номер месяца (`01-12`).
- `%M` — минута (`00-59`).
- `%p` — указатель `AM` или `PM` при двенадцатичасовом обозначении.
- `%S` — секунда (`00-59`).
- `%w` — номер дня недели (`0-6`); нулевой — воскресенье, шестой — суббота.
- `%y` — две последние цифры номера года (`00-99`).

- **%Y** — год (например, **2001**).

Примеры

Вызов **^date[100%%-точное время. Дата: %d.%m.%Y. Время: %H:%M:%S]** даст примерно следующий результат:

```
100%-точное время. Дата: 03.03.2001. Время: 14:31:19
```

3.13.2. Загрузка в таблицу даты и времени. Оператор loadtimestamp

Формат вызова

```
^loadtimestamp[]

^loadtimestamp[дата_время]
```

Аргументы

дата_время — дата и время в формате **year.month.day hour:min**.

Описание

Оператор загружает в текущую таблицу указанные дату и время. Столбцам таблицы присваиваются имена **year**, **month**, **day**, **hour** и **min**. Если аргумент не указан, загружаются текущие дата и время.

Примеры

```
^loadtimestamp[2000.10.28 16:45]
```

3.13.3. Загрузка в таблицу календаря на месяц. Оператор calendar

Формат вызова

```
^calendar[тип_календаря;год;месяц]
```

Аргумент

тип_календаря — тип календаря («английский» или «русский»).

год, **месяц** — эти аргументы задают месяц, календарь которого нужно загрузить в таблицу.

Описание

Оператор загружает в текущую таблицу календарь указанного месяца. Календарь представляется в виде семи столбцов, соответствующих дням недели. Если аргумент **тип_календаря** имеет значение **month**, то первым днем недели считается понедельник, а если **montheng** — воскресенье.

Примеры

Вызов `^calendar[month;2001;03]` загрузит в текущую таблицу следующие данные.

```
          01 02 03 04
05 06 07 08 09 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

На пересечении первой строки и первых трех столбцов, а также на пересечении последней строки и последнего столбца — пустые значения.

Следующий фрагмент кода выводит календарь со ссылками на новости по датам за текущий месяц.

```
^context[month;

^calendar[month;^date[%Y];^date[%m]]
<TT>
^menu[^for[i;0;6;
            ^ifdef[^item[^var[i]];
                ^var[params;y=^date[%Y]&m=^date[%m]&d=^item[^var[i]]]
                 <A HREF="newsdetails.html?^var[params]">^item[^var[i]
]</A>;
                  ^; ^;] ^;
      ];
      <BR>
]
</TT>

]
```

3.13.4. Загрузка в таблицу календаря на неделю. Оператор calendar

Формат вызова

```
^calendar[тип_календаря;год;месяц;день]
```

Аргументы

тип_календаря — тип календаря («английский» или «русский»).

год, **месяц**, **число** — эти аргументы задают неделю, календарь которой нужно загрузить в таблицу.

Описание

Оператора загружает в текущую таблицу календарь на неделю, представленный в виде трех столбцов. Имена столбцов: **year**, **month** и **day**. Неделя, для которой генерируется календарь, определяется указанными годом, месяцем и числом. Если аргумент **тип_календаря** имеет значение **week**, то первым днем недели считается понедельник, а если **weekeng** — воскресенье.

Примеры

Вызов `^calendar[week;2000;08;3]` загрузит в текущую таблицу следующие данные.

```
2000    07      31
2000    08      01
2000    08      02
2000    08      03
2000    08      04
2000    08      05
2000    08      06
```

3.13.5. Сдвиг даты на один день, месяц или год. Оператор dateoffset

Формат вызова

```
^dateoffset[сдвиг_год;сдвиг_месяц;сдвиг_день]
```

Аргументы

`сдвиг_год` — количество прибавляемых или вычитаемых лет.

`сдвиг_месяц` — количество прибавляемых или вычитаемых месяцев.

`сдвиг_день` — количество прибавляемых или вычитаемых дней.

Описание

Оператор изменяет дату в формате оператора `loadtimestamp` в текущей таблице на один год, месяц или день, в зависимости от того, какой из аргументов указан. Из трех возможных может присутствовать только один аргумент, он может быть равен только `1` или `-1`.

Примеры

```
^loadtimestamp[2000.07.31]
^dateoffset[;;1]
^item[year].^item[month].^item[day]
```

Обработка приведенного выше фрагмента даст результат `2000-08-01`.

3.14. Графические файлы

3.14.1. Получение размеров изображения. Оператор imgsize

Формат вызова

```
^imgsize[полное_имя_файла]
```

Аргументы

полное_имя_файла — полный путь к графическому файлу.

Описание

Оператор загружает в текущую таблицу размеры изображения, которое находится в графическом файле формата GIF или JPEG. В нулевой столбец записывается ширина, а в первый — высота изображения.

Примеры

```
^context[image_size;
         ^imgsize[image.gif]
         <IMG SRC="image.gif" WIDTH=^item[0] HEIGHT=^item[1]>
]
```

3.14.2. Формирование изображения. Оператор gif_template

Формат вызова

```
^gif_template[имя_файла;инструкции]

^gif_template[имя_файла;инструкции;имя_файла_результата]
```

Аргументы

имя_файла — имя файла с исходным изображением.

инструкции — действия, которые нужно выполнить с изображением.

имя_файла_результата — имя файла, в который будет записано созданное изображение.

Описание

Оператор загружает изображение в формате GIF и выполняет с ним указанные действия. Полученное изображение выдается пользователю или записывается в файл.

Последовательность инструкций, управляющих формированием изображения, представляет собой таблицу в формате tab-delimited (как в операторе **config** — см. п. 3.6.1).

Инструкции

Инструкция	Формат	Описание
Вывод надписи	**ширина_символа>x>y>текст_надписи**	Выводит надпись, левый верхний угол которой располагается в точке с координатами **x** и **y**. Если не указана ширина символа, используется пропорциональный шрифт.
Смена цвета	**replace_color>hex-цвет1>hex-цвет2>вершины_многоугольника**	Внутри многоугольника заменяет первый цвет вторым.

Шрифт, который будет использоваться для вывода надписей, задается макросом **font**, который необходимо описать следующим образом:

```
^macro_new[font;имя_файла_шрифта.gif
ширина_символа>высота_символа
набор_букв
]
```

Файл шрифта представляет собой двуцветный GIF-файл, в котором буквы расположены столбцом. Значения высоты и ширины символа отделяются друг от друга табуляцией. Набор букв — это строка, которая показывает, какие буквы и в какой последовательности размещены в файле шрифта. Дело в том, что часто для вывода надписи не нужны все буквы алфавита. В этом случае в файл шрифта помещают только их. Поскольку Парсер «не знает», какие буквы и в каком порядке мы поместили в файл шрифта, необходимо явно указать на это. Например, если файл шрифта содержит буквы **к**, **г** и **б**, то в качестве набора букв следует передать строку **кгб**.

Примеры

Приведенный ниже фрагмент кода загружает изображение из файла **metro.gif** и наносит на него надпись **схема линий московского метрополитена**. Затем черные точки, расположенные внутри квадрата с указанными координатами вершин, перекрашиваются в голубой цвет. Полученное изображение записывается в файл **metro_new.gif**.

```
^macro_new[font;metro_font_rus.gif
30>30
АБВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдежзиклмнопрстуфхцчшщъыьэюя
]
^gif_template[metro.gif;
20>20>схема линий московского метрополитена
replace_color>000000>00FF00>100,100,200,100,200,200,100,200
;metro_new.gif]
```

3.15. Код и режимы его обработки

3.15.1. Вставка комментария. Оператор rem

Формат вызова

```
^rem[комментарий]
```

Аргументы

комментарий — текст комментария.

Описание

Оператор предназначен для вставки в код комментариев. В качестве аргумента этому оператору передается текст комментария. Обработка аргумента не выполняется, комментарии в результирующий код не вставляются.

Замечание. Строки, у которых в первой колонке находится символ **#** («диез»), считаются комментариями и не обрабатываются Парсером.

Примеры

```
# Выведем HTML-таблицу
^context[data; ^rem[назначаем текущую таблицу]
        <TABLE BORDER=1>
        ^menu[<TR> ^rem[обходим строки таблицы]
                  <TD>^item[title]</TD><TD>^item[size]</TD>
              </TR>
        ]         </TABLE>
]
# Вот и вывели, а ты говорил, не получится!
```

3.15.2. Обработка кода. Оператор process

Формат вызова

```
^process[код]
```

Аргументы

код — код, подлежащий обработке.

Описание

Оператор обрабатывает переданный ему код. Оператор применяется, если, например, часть шаблона страницы хранится в базе данных.

Замечание. Оператор вычисляется при уровне обработке **none**.

Примеры

Вызов `^process[^^date[%Y]]` вернет номер текущего года.

3.15.3. Установка уровня обработки. Оператор level

Формат вызова

```
^level[уровень_обработки;код]
```

Аргументы

уровень_обработки — устанавливаемый уровень обработки.

код — код, обрабатываемый на установленном уровне.

Описание

Оператор устанавливает для переданного кода указанный уровень обработки. Механизм уровней обработки подробно описан в п. 1.6.

Примеры

```
^context[^load[common.js]
         ^level[javascript;
                <SCRIPT LANGUAGE=javascript>
                   ^menu[item[0]]
                </SCRIPT>
         ]
]
```

3.15.4. Оптимизация HTML-кода. Оператор optimize

Формат вызова

```
^optimize[режим_оптимизации]
```

Аргументы

режим_оптимизации — может принимать значение **yes** или **no**, если оптимизацию нужно включить или отключить соответственно.

Описание

Оператор задает режим оптимизации HTML-кода перед выдачей в браузер. Оптимизация кода заключается в том, что из нескольких подряд идущих символов, являющихся по правилам языка HTML разделителями (пробелов, табуляций, концов строк разного вида) оставляется первый.

Замечание. По умолчанию оптимизация включена. Отключать оптимизацию необходимо в некоторых специфических случаях, например, для поля формы **TEXTAREA**.

Примеры

```
<FORM>
^optimize[no]
<TEXTAREA>
   Я достаю из широких штанин

         дубликатом бесценного груза
</TEXTAREA>
</FORM>
```

4. Рекомендации по использованию языка

4.1. Как сделать код «комильфо»?

Разрабатывая сайт, рекомендуется соблюдать определенные правила составления и оформления кода. В совокупности эти правила образуют *стиль кодирования*. Пренебрежение стилем само по себе не сделает код неработоспособным, скорее работоспособность утратит сам кодер, так как через некоторое время перестанет ориентироваться в собственном произведении. Стиль кодирования становится тем более важным, если сайт создается командой специалистов. Не договорившись о стиле, члены команды рискуют принять участие в проекте по созданию вавилонского сайта. Каждый коллектив может придерживаться того стиля, который ближе и привычнее для большинства разработчиков, однако хороший стиль непременно обладает следующими качествами:

- Наличие подробных и внятных комментариев. Каждое неочевидное действие должно сопровождаться пояснениями.
- Внятность имен. Все имена должны быть понятными, иметь схожую структуру и использовать однотипную мнемонику.
- Единообразие оформления. Одни и те же конструкции должны оформляться одним и тем же способом.
- Обозримость кода. Выражение всякой «мысли» не должно занимать более одного экрана.
- Универсальность технических приемов. Одни и те же задачи всегда решаются одним и тем же способом, если нет особых причин применить другой способ.

4.2. Комментируйте код

Старайтесь комментировать все действия, имеющие самостоятельный смысл. Обязательно помещайте комментарии в заголовках макросов. Участки кода, имеющие существенно разный смысл, можно отделять друг от друга пустыми строками — Парсер их оптимизирует при обработке. Комментируя схожие по характеру действия, старайтесь использовать схожую лексику и схожие грамматические конструкции.

Напоминаем предусмотренные в Парсере способы комментирования кода.

- Оператор **`rem`** (см. п. 3.15.1).
- Любая строка, имеющая в первой колонке символ **`#`**, является комментарием.
- Комментарий можно поместить после закрывающей квадратной скобки в заголовке макроса с явным указанием параметров (см. п. 1.3.3).

4.3. Внимательно относитесь к именам

Имена должны быть информативными. Они обязаны соответствовать назначению тех сущностей, которым присвоены. Не стоит давать макросу имя любимой кошки (девушки, рок-звезды и т. д.), вспомните лучше, какую задачу он решает. Избегайте абстрактных и многозначных существительных. Имена вроде **`parameter`**, **`x2378`** или **`string`** ни о чем не говорят. Стремитесь к внятности имен. Хороший способ формирования осмысленных имен — придерживаться принципа «глагол_существительное», так, макрос, сохраняющий настройки, можно назвать **`save_options`**.

Не используйте аллитераций и непонятных аббревиатур. Старайтесь, чтобы имена были легко произносимы и однозначно воспринимались на слух. Каково будет объяснять коллеге по телефону, что в код необходимо добавить вызов макроса `_RaCS_zajyka_moya`? Старайтесь сделать код понятным, а не забавным. Возможно, конструкции `^if[$en;loran]` или `^config[vam]` выглядят смешно, но все-таки код — не место для каламбуров.

Имена `i`, `j`, `k` позволительно давать только счетчикам циклов.

Парсер различает регистр букв в именах, однако категорически не рекомендуется использовать имена, отличающиеся только регистром. Не следует использовать похожие имена, а также имена, одно из которых является началом другого (например, `extrapolation` и `extra`).

Не используйте имен, совпадающих с именами операторов (этим вы создадите трудности не столько Парсеру, сколько себе и коллегам). Нежелательно присваивать таблицам имена, совпадающие с возможными зарезервированными аргументами операторов (такими, как `next`, `prev`, `DEFAULT`). Не пользуйтесь безымянной таблицей. Это допустимо разве что на тестовых или очень простых страницах.

Помните, что имена переменных доступны во всем коде, вовлеченном в процесс обработки одного шаблона. Если вы подгружаете макросы с помощью оператора `macro_use`, убедитесь, что они не «испортят» нужные вам переменные.

4.4. Макросы должны быть обозримыми

Стремитесь к тому, чтобы макрос умещался на одном экране при стандартном разрешении монитора. Постарайтесь уложиться в 30 — 40 строк. Человек слаб, поэтому, разбирая текст макроса из двух тысяч строк, может впасть в грех злословья. Помните об этом и не подводите коллег.

4.5. Правильно структурируйте код

Если оператор занимает несколько строк, располагайте закрывающую квадратную скобку на одном уровне с «птичкой». Используйте имеющуюся во многих текстовых редакторах функцию поиска парных скобок и подсветки текущего блока.

Не ставьте из эстетических соображений лишних пробелов перед значениями аргументов, поскольку они будут восприняты Парсером как неотъемлемая часть этих значений.

Не злоупотребляйте функцией копирования блоков в текстовом редакторе — это рассадник ошибок, причем трудных для поиска. Схожую логику следует выносить в макросы, а не бездумно тиражировать.

4.6. Не используйте оператор form вместо number

Не используйте оператор `form` для получения численных значений, подставляемых в другие операторы. Вместо него используйте оператор `number`, например:

```
^sql[DELETE FROM people WHERE id=^number[id]]
```

Иначе любой подрастающий хакер сотрет все ваши данные раньше, чем вы успеете нажать на Reset.

Приложение A.
Установка Парсера на веб-сервер под Apache

1. Поместите файл с исполняемым кодом Парсера (в текущей версии **parser2.cgi**) в каталог для CGI-скриптов.
2. Для русской версии Apache добавьте в конфигурационный файл веб-сервера (обычно **httpd.conf**) строку: **CharsetRecodeMultipartForms Off**.
3. Добавьте в файл **.htaccess** (или в **httpd.conf** в секцию **<virtualhost>**) блоки:

```
#назначение обработчиком .html страниц
AddType parsed-html .html
Action parsed-html /cgi-bin/parser2.cgi
#запрет на доступ к .cfg файлам. основной: _macro.cfg
<Files ~ "\.cfg$">
Order allow,deny
Deny from all
</Files>
```

Приложение B.
Установка Парсера на веб-сервер под IIS 4+

1. Запустите Management Console, нажмите на правую кнопку мыши на названии вашего веб-сервера и выберите **Properties**.
2. Перейдите на вкладку **Home directory** и в разделе **Application settings** нажмите на кнопку **Configuration...** В появившемся окне нажмите на кнопку **Add**.
3. В поле **Executable** введите полный путь к файлу **parser2.exe**.
4. В поле **Extension** введите строку **.html**.
5. Включите опцию **Check that file exists**.
6. Нажмите на кнопку **OK**.

Замечание. Парсер можно установить на IIS 2 и Personal Web Server, назначив обработчиком HTML-файлов CGI-скрипт **parser2.exe**.

Приложение C.
Регулярные выражения в стандарте POSIX 1003.2

Внимание! Нижеследующий текст является переводом man-страницы, размещенной по адресу http://www.gsp.com/cgi-bin/man.cgi?section=7&topic=re_format.

Описание

Регулярные выражения (РВ), описанные в стандарте POSIX 1003.2, имеют два различных синтаксиса: современные РВ (примерно такие как *egrep*, называемые в стандарте 1003.2 «расширенными» РВ) и устаревшие РВ (примерно такие как *ed*, называемые в стандарте 1003.2 «базовыми» РВ). Устаревшие РВ существуют главным образом для обеспечения обратной совместимости в некоторых старых программах. Эти выражения будут описаны в самом конце данного документа. Стандарт 1003.2 оставляет открытыми многие аспекты синтаксиса и семантики РВ, вследствие чего некоторые решения, построенные на данном стандарте, могут быть не полностью совместимы с другими реализациями на его же основе. Значок «§» отмечает такие потенциально частично несовместимые решения.

Современные РВ представляют собой одну или несколько непустых секций (*branches*), разделенных знаком «|». Соответствие определяется по любой секции такого выражения.

Секция состоит из одной или нескольких соединенных частей (*pieces*). Соответствие определяется сначала для первой части, потом для второй и т. д.

Часть представляет собой элементарное выражение (*atom*), вслед за которым может следовать один из символов «*», «+», «?» или же ограничитель (*bound*). Элементарное выражение и «*» допускают 0 или более соответствий. Элементарное выражение и «+» допускают одно или более соответствий, а элементарное выражение и «?» допускают 0 или 1 соответствие.

Ограничитель представляет собой конструкцию, начинающуюся с открывающей скобки «{», за которой следует беззнаковое десятичное целое число, возможно с последующей запятой «,» и еще одним беззнаковым десятичным целым числом. Вся эта конструкция обязательно закрывается правой скобкой «}». Значения целых чисел могут лежать в интервале от 0 до RE_DUP_MAX (255) включительно, и в случае использования двух значений первое не должно превышать второе.

Элементарное выражение, за которым следует ограничитель с одним целым числом *i* и без запятой, допускает ровно *i* соответствий выражения. Элементарное выражение и целое число *i* вместе с запятой допускает *i* и более соответствий, а элементарное выражение и два разделенных запятой целых числа допускают число соответствий от i до j включительно.

Элементарное выражение может быть заключенным в круглые скобки «()» регулярным выражением (соответствие определяется по регулярному выражению), пустыми круглыми скобками (соответствует пустой строке), ограниченным выражением — *bracket expression* (см. далее), точкой «.» (соответствует одному любому символу), знаком «^» (соответствует пустой подстроке в начале строки), знаком «$» (соответствует пустой подстроке в конце строки); сочетанием «\» и одного из следующих символов «^.[$()|*+?{\» (соответствует одному обычному символу), сочетанием «\» и любого другого символа (соответствует одному обычному символу, как при отсутствии «\»), а также единственным неслужебным символом (соответствует этому символу).

Символ «{» вместе с последующим обычным символом, не являющимся цифрой, не считается началом ограничителя, а символ «\» не может служить концом РВ.

Ограниченное выражение представляет собой перечень заключенных в квадратные «[]» скобки символов. При этом соответствие обычно определяется по какому-либо одному из символов в списке (исключение см. далее). Если перечень символов начинается со знака «^», соответствие определяется по любому символу с начала списка (исключение см. далее). Если два символа в списке разделены знаком «-», — это означает непрерывное подмножество полного диапазона символов, расположенное между двумя указанными символами включительно, т. е. «[0-9]» в ASCII-кодировке соответствует любой десятичной цифре. При использовании двойного диапазона конечная граница оказывается неопределенной, например в случае «a-c-e». Применение диапазонов символов сильно зависит от последовательности сравнения и поэтому настоятельно не рекомендуется при разработке переносимых программ.

При включении в список знака «]» в качестве сравниваемого символа он должен быть первым (следуя возможно за «^»). В этом же случае знак «-» должен быть первым или последним символом или же второй границей диапазона. Для использования при сравнении знака «-» в качестве первой границы диапазона его необходимо заключить в квадратные скобки «[» и «]» (см. далее). За исключением вышеописанных случаев и некоторых комбинаций с использованием «[» (см. следующий параграф), любые другие специальные символы, включая «\», утрачивают свое служебное значение внутри ограниченного выражения.

Внутри ограниченного выражения элемент сопоставления (символ; последовательность нескольких символов, рассматриваемая как единственный символ; или имя сравниваемой последовательности для одного из предыдущих случаев), заключенный в квадратные скобки «[» и «]», образует сравниваемую последовательность из этих символов. Эта последовательность представляет из себя единственный элемент в списке ограниченного выражения. Таким образом, ограниченное выражение, содержащее сравниваемую последовательность из нескольких символов, может соответствовать более чем одному символу: например, если сравниваемая последовательность включает элемент «ch», то РВ «[[.ch.]]*c» будет соответствовать первым пяти символам в последовательности «chchcc».

Внутри ограниченного выражения заключенный в «[=» и «=]» элемент сопоставления рассматривается как класс эквивалентности, образующий последовательности сравниваемых символов, которые точно соответствуют этому элементу, включая его самого. (При отсутствии другого эквивалентного элемента сопоставления обработка последовательности символов аналогична случаю закрывающих скобок «[» и «]».) Например, если «o» и «^» входят в класс эквивалентности, то тогда сочетания «[[=o=]]», «[[=^=]]» и «[o^]» являются синонимами. Класс эквивалентности может служить границей диапазона.

Внутри ограниченного выражения имя класса символов, заключенное в «[:» и «:]», образует перечень всех символов, принадлежащих данному классу. Существуют следующие стандартные классы символов:

alnum	alpha	blank	cntrl
digit	graph	lower	print
punct	space	upper	xdigit

Они соответствуют символьным классам, определенным в `ctype`. В зависимости от системы классы могут различаться. Символьный класс не может использоваться в качестве границы диапазона.

Существуют два особых вида ограниченных выражений: ограниченные выражения «[[:<:]]» и «[[:>:]]» соответствуют нулевой строке в начале и конце слова. Слово, определяется как ограниченная последовательность символов. Используемые в слове символы относятся к классу *alnum* (определяется через `ctype`), а также могут являться символом подчеркивания. Этот синтаксис не описан в стандарте POSIX 1003.2, однако является совместимым с ним и может аккуратно использоваться при создании переносимого программного обеспечения.

В случае если РВ имеет несколько отвечающих условию соответствия подстрок в обрабатываемой строке, то учитывается соответствие для той подстроки, которая начинается раньше. Если, начиная с определенной позиции, существует несколько соответствий РВ, то соответствие определяется по наиболее длинной строке. Подвыражения соответствуют наиболее длинным возможным подстрокам исходя из принципа, что полное соответствие должно быть максимально длинным. При этом также соблюдается правило приоритетности первой подстроки по сравнению с теми, которые начинаются за ней. Обратите внимание, что подвыражения более высокого уровня имеют приоритет над низкоуровневыми составляющими подвыражений.

Длина соответствия определяется по числу символов, а не по сравниваемым элементам. Нулевая строка считается длиннее строки, которая вообще не имеет соответствий. Например, «bb*» соответствует трем символам в середине «abbbc», а «(wee|week)(knights|nights)» соответствует десяти символам в «weeknights». При сравнении «(.*).*» с «abc» заключенное в скобки подвыражение соответствует всем трем символам, а при сравнении «(a*)*» с «bc» все регулярное выражение, а также его заключенная в скобки часть соответствуют нулевой строке.

Независимое от регистра сравнение не учитывает различия при использовании больших и малых букв. Если тот или иной независимый от регистра символ появляется за пределами ограниченного выражения, он легко преобразуется в ограниченное выражение, содержащее оба его регистра, т. е. «x» превращается «[xX]». Аналогично, при появлении такого символа внутри ограниченного выражения происходит сопряжение регистров, т. е. «[x]» становится «[xX]», а «[[invalid operator x]» — «]».

На длину РВ не накладывается никаких ограничений, поэтому для сохранения POSIX-совместимости в переносимых программах нельзя использовать РВ длиннее 256 байт.

Устаревшие («базовые») регулярные выражения имеют несколько отличий. В «базовых» РВ «|» рассматривается как обычный символ и не имеет никакого функционального эквивалента. «+» и «?» также являются обычными символами, а их функциональность задается с помощью соответствующих ограничителей {1,} и {0,1}. Обратите внимание на то, что «x+» в современных регулярных выражениях эквивалентно «xx*». В «базовых» РВ ограничителями являются «\{» и «\}», в то время как «{» и «}» сами по себе рассматриваются как обычные символы. Скобки для вложенных подвыражений задаются с помощью «\(» и «\)», а простые скобки «(» и «)» являются обычными символами. Знак «^» считается обычным символом, за исключением случая, когда он находится в начале РВ, а также в начале заключенного в скобки подвыражения. Знак «$» считается обычным символом, за исключением случая, когда он находится в конце РВ, а также в конце заключенного в скобки подвыражения. Знак «*» считается обычным символом, за исключением случая, когда он находится в начале РВ, а также в начале заключенного в скобки подвыражения (возможно после символа «^»). Наконец, в современных РВ существует новый тип элементарного выражения — *back reference*, которое представляет из себя последовательность из знака «\» и ненулевого десятичного числа. Данное выражение соответствует d-му, заключенному в скобки подвыражению (нумерация подвыражений задается по порядку их следования справа налево), т. е., например «\([bc]\)\1» соответствует «bb» или «cc», но не «bc».

Ошибки

Существует два вида приводящих к ошибкам РВ.

В существующей спецификации «)» рассматривается как обычный символ при отсутствии закрывающей «)». Это приводит к неожиданным ошибкам разбиения на слова. Средство очевидно — избегайте этого.

Использование выражений типа *back reference* может иметь катастрофические ошибки даже в эффективных решениях, поскольку такие выражения бывают неоднозначными (разве «a\(\(b\)*\2\)*d» соответствует «abbbd»?). Избегайте этого.

Спецификация 1003.2 на независимое от регистра соответствие также расплывчата. У разработчиков в данный момент принято такое определение: «Один регистр подразумевает оба регистра».

Синтаксис ограничителей слов просто ужасен.

Часто задаваемые вопросы

Что такое PARSER и кто его применяет в работе?
PARSER — это средство скриптования сайтов с помощью простого языка. Он немного сложнее HTML, но совершенно не требует от кодера умения программировать.
PARSER начал создаваться в Студии Лебедева в 1997 году. Сегодня все сайты в Студии Лебедева создаются с его помощью.

А зачем он вообще?

Для того, чтобы по многу раз не переписывать одно и то же. С помощью макросов можно описать конструкцию сайта таким образом, что для изменения внешнего вида всего сайта, потребуется изменить только 1-2 файла.

Если есть PHP, то для чего PARSER?

PHP — это язык программирования, требующий соответствующей подготовки. На парсере могут работать те, кто называются «html-кодеры», то есть, далеко не программисты. PARSER именно тем и хорош, что позволяет простым смертным создавать и поддерживать динамические сайты без больших затрат времени.

Почему PARSER распространяется бесплатно?

Мы в Студии Лебедева абсолютно убеждены в том, что ни одна закрытая технология не может долгое время оставаться современной и актуальной. Всегда будет появляться что-то новое, что будет превосходить технологии предыдущего поколения. Делая PARSER доступным каждому, мы не хотим ни с кем конкурировать в области инструментов разработки сайтов. Широкое использования PARSER поможет технологии оставаться современной.

Продажа технологии сужает круг пользователей и замедляет развитие новых проектов. Мы не продаем инструмент, а даем его бесплатно любому желающему, рассчитывая на то, что этот шаг поможет развитию российского интернета.

Как насчет открытых кодов?

PARSER будет распространятся в исходниках в 2001 году. Скорее всего, это произойдет весной.

Как подключить PARSER к веб-серверу?

Для этого в Apache нужно в файлах .htaccess или httpd.conf (секция `<virtualhost>`) следующие строки (подразумевается, что `/cgi-bin` указывает на каталог, прописаный в `ScriptAlias` в `httpd.conf`):

```
AddType parsed-html .html

Action parsed-html /cgi-bin/parser
```

Скопировать файл parser в каталог `/cgi-bin`.
В каталоге сайта, на который указывает `localhost`, создать файл `index.html`.

Можно ли использовать в коде спецсимволы «;» и «^»?

Конечно. Их нужно либо предварять символом «^», либо брать целиком весь параметр в обратные кавычки «`». Например:

```
^eq[abc^;def;ghijkl;...;...]

^eq[`abc;def`;ghijkl;...;...]
```

Как перейти на работу с PARSER в своих разработках?

Можно постепенно заменять конструкциями парсера файлы уже готового сайта, например описать заголовки и таблицы html-документа макросами, использовать переменные окружения через парсер, упростить обработку данных форм. Или изначально продумать удобную конструкцию сайта, и писать код, исходя из возможностей парсера. Парсер не подменяет собой html-код, он генерирует его.

Поэтому изменять исходные тексты очень легко, и всегда сразу можно увидеть, что получилось в результате той или иной команды.

Обязательно ли ставить русскую версию Apache для работы PARSER?

Нет. PARSER работает с английским и русским Apache, а также с IIS. И вообще — любой сервер, позволяющий обрабатывать документы ДО их выдачи пользователю, сможет работать с парсером.

Как запустить PARSER под IIS?

Для этого в программе конфигурации IIS кликаете правой клавишей мыши на веб-сайт и выбираете `Properties`. Открывается окно `Web Site Properties`. Затем в странице `Home Directory` нажимаете кнопку `Configuration`. В появившемся окне нажимаете `Add`, и прописываете путь к `PARSER`. В строке `Extension` пишете `html`, и жмете OK. Затем в окне `Web Site Properties` выбираете страницу `Documents`, и добавляете `index.html` в список файлов, открываемых по умолчанию. Перезагружаете сервер, 3 раза "ку", крутите фонарики и радуетесь полчаса.

Как запускать скрипты из-под парсера, и не получать ошибку №6 под win32?

Для этого нужно воспользоваться оператором `^exec[script_to_run]`. Путь к скрипту нужно указывать относительно ТЕКУЩЕГО html-документа. Например, если путь к текущему документу `c:\apache\htdocs\index.html`, а скрипт находится в директории `c:\apache\cgi-bin`, то выполнять оператор `^exec` нужно следующим образом:

```
^exec[/../cgi-bin/script_to_run]
```

Что такое контекст?

Контекст — это двухмерный массив данных. Для обращения к элементу текущей строки текущего контекста используется оператор `^item[n]`, где n — номер элемента или его название (в случае именованного контекста). Чтобы обратиться к элементу другого контекста, следует использовать конструкцию `^item[m;n]`, где m — название желаемого контекста, а n — номер или название элемента.

Для перемещения по строкам контекста используются операторы `^menu` и `^shift`.

Для поиска элемента в контексте используется оператор ^locate.

Чтобы создать именованный контекст, используются конструкции ^config[named;...] и ^load [named;...].

Почему в конструкции

```
^macro[server;
  ^load[data.txt]
    ^menu[
      ^sql[insert into table values ('^item[0]', '^item[1]')]
    ]
```

] вставляется всего одна запись?

Это происходит потому, что оператор ^sql[] меняет текущий контекс, независимо от команды sql. Для правильного внесения записей в базу, следует использовать подобную конструкцию:

```
^macro[server;
  ^context[data;
    ^load[data.txt]
    ^menu[
      ^context[sql;
        ^sql[insert into table values('^item[data;0]',
'^item[data;1]')]
      ]
    ]
  ]
]
```