Язык шаблонов Communiware версия 0.93

В.Б. Вагнер

11 мая 2004 г.

Оглавление

Глава 1

Неформальное введение в язык шаблонов

Язык шаблонов Communiware отличается от большинства широкоизвестных языков программирования, таких как C, Perl или Python, тем что он предназначен не для реализации алгоритмов, а для описания способа визуализации данных, хранящихся в базе данных Communiware. Родственников ему надо искать скорее среди разнообразных языков описания отчетов, таких как оператор формата в Perl или синтаксис строки формата в C-шной функции `printf`.

Шаблон Communiware представляет собой фрагмент HTML-страницы (или целую страницу), в которую местами вставлены указания для Communiware изобразить в этом месте некоторую информацию из базы данных. Эти указания называются *динамическими элементами* в противоположность статическим элементам HTML-оформления, не меняющимся вне зависимости от того, для показа каких данных используется шаблон.

Один динамический элемент может быть ответственным как за такую маленькую часть страницы, как отдельное числовое значение, так и за достаточно большую, но содержательно единую часть, такую например, как дискуссию по данному материалу или навигационное меню сайта. В последнем случае он, скорее всего будет содержать ссылку на шаблон фрагмента, ответственный за оформление этой части.

Имена динамических элементов в Communiware case-sensitive, То есть, большие и маленькие буквы считаются различными и название

динамического элемента следует писать так как это описано в документации, и никак иначе.

То же самое касается и большинства остальных имен, упоминаемых в тексте шаблона — идентификаторов итемов, имен типов, атрибутов и др. Если явно не указано противное, **большие и маленькие буквы считаются различными**

Communiware — система, предназначенная для показа web-страниц, поэтому обработка любого шаблона приводит к формированию некоей HTML-страницы[1].

Независимо от того, каким образом производится обработка шаблона — в результате http-запроса, при почтовой рассылке и так далее, на входе всегда есть некий содержательный итем и шаблон, по которому его надо показать, а на выходе — сформированная HTML-страница.

1.1 Роль языка шаблонов при разработке Коммунивер-сайта

Основным содержимым Communiware-сайта являются не шаблоны, а контент — набор содержательных итемов (материалов, рубрик и др) и система связей между ними. Шаблоны это только способ описать особенности визуализации, придать сайту неповторимый внешний вид.

Если сайт был спроектирован правильно, т.е. все зависимости между содержательными итемами непосредственно отражаются посредством связей, то одни и те же содержательные данные можно показывать по разным шаблонам, при условии что эти шаблоны предназначены для отображения той же *онтологии*, то есть набора типов итемов и связей.

При разработке шаблонов Communiware не нужно стремиться запрограммировать какую-то сложную логику на языке шаблонов. Логика взаимоотношений между итемами должна быть непосредственно отражена в базе с помощью системы связей, а шаблоны должны визуализировать только простейшие взаимоотношения текущего итема с соседними. Например, в шаблоне рубрики обычно присутствует список «подрубрик текущей рубрики» и список «материалов, входящих в данную рубрику».

С другой стороны, пути навигации пользователя по сайту, набор

[1]Возможно формирование страниц в любой другой XML или SGML DTD, например WML, но мы пока это не рассматриваем

информации, отображаемый на страницах соответствующих итемов, и даже те возможности по созданию итемвов и связыванию их между собой, которыми располагают пользователи сайта, определяются именно шаблонами.

Поэтому шаблоны являются важнейшей частью Communiware-сайта, хотя разработка сайта должна начинаться не с них, а с планирования структуры сайта, выбора типов используемых итемов и способов их связывания. Но именно шаблоны являются тем мясом, которое необходимо нарастить на этот скелет, чтобы получить живой функционирующий сайт.

1.2 Соотношение между шаблонами и другими итемами

Когда пользователь обращается к Коммунивер-сайту, его обычно интересует какой-то содержательный элемент, например «такая-то статья» или «главная страница сайта». При этом ему совершенно не важно что в отображении показанной ему страницы принимали участие еще и какие-то шаблоны.

Связывание шаблонов с содержательными итемами — это дело разработчика сайта.

Каждому содержательному итему в Communiware соответствует *шаблон по умолчанию*, используемый для отображения этого итема в нормальных условиях. Этот шаблон назначается при создании итема, и в дальнейшем может быть изменен с помощью интерфейса вебмастера или какого-нибудь другого инструмента, позволяющего редактировать все свойства итема.

Поэтому для тех содержательных итемов, которые образуют основной „скелет” сайта, разработчик может назначить шаблоны сам.

Но большую часть страниц сайта составляют содержательные итемы, которые создаются людьми, ответственными за содержание, уже после того, как разработчик окончил работу над сайтом. При этом редактора или корреспондента, публикующего статью, меньше всего интересует по какому шаблону она будет показана. Он совершенно обоснованно ожидает, что эта статья будет показана так же, как и остальные статьи на данном сайте.

Поэтому в Communiware существует механизм *стандартных шаблонов* который позволяет разработчику управлять тем, какие шаблоны

будут назначены вновь создаваемым итемам определенного типа. Данный механизм позволяет задать шаблоны для каждого типа итема как на уровне сайта в целом, так и на уровне отдельной рубрики.

Стандартные шаблоны действуют в том случае, если при создании итема ему не был явно указан шаблон, или при редактировании шаблон был сброшен в значение «Стандартный для типа».

Если в момент создания итема ему указана рубрика, и этой рубрике (или одной из вышележащих) соответствует стандартный шаблон для данного типа, то итему будет назначен этот шаблон. Если рубрика не была указана, или ни одной из вышележащих рубрик не соответствует стандартный шаблон, то будет использован шаблон, стандартный для сайта.

Если для данного типа итема не назначен стандартный шаблон на уровне сайта, то создать итем такого типа через интерфейс вебмастера просто нельзя — в меню «Создать новый итем» этот тип не появится. Тем не менее, через интерфейсы постинга такой итем создать можно. В этом случае, ему будет присвоен шаблон объявленный шаблоном по умолчанию для типа. Обычно, это шаблон принадлежащий служебному сайту Communiware (`/default`).

Не следует забывать, что шаблон сам по себе является итемом, и хранится в базе данных Communiware точно так же, как и содежательные материалы. Как правило все шаблоны имеют шаблон по умолчанию `template_t`, который показывает исходный текст шаблона, и его взаимосвязи с другими шаблонами.

Более того, существует возможность устанавливать различные связи между шаблонами и содержательными итемами, что позволяет, например, реализовать систему альтернативных видов некоторого итема.

Как правило, на типичном сайте существуют итемы, которым соответствует несколько разных страниц. Даже для такой маленькой части сайта, как статья, существует возможность либо показать сам текст статьи, либо дискуссию по ней.

Для корня сайта существует гораздо больший набор видов — здесь и карта сайта (рубрикатор) и список вновь опубликованных материалов, и список авторов и др.

Для того, чтобы увидеть итем не по его „родному" шаблону, а по некоторому другому, нужно указать URL вида `/префикс сайта/итем/шаблон`.

Все динамические элементы Communiware, формирующие гиперссылки на итемы, позволяют указать шаблон, по которому этот итем

нужно показать.

1.3 Понятие контекста

Шаблон Communiware никогда не выполняется сам по себе, он всегда применяется к какому-то итему. Поэтому в шаблоне можно обращаться к данным, хранящимся в этом итеме. Эти данные представляют собой набор именованных значений, называемых *атрибутами* итема. Эти атрибуты могут быть строками, числами, датами и текстами с форматированием (RICHTEXT). Кроме атрибутов текущего итема в шаблоне доступен ряд параметров, характеризующих текущий обрабатываемый запрос, таких как имя (идентификатор) пользователя, тип браузера для которого формируется страница и так далее. Совокупность всех значений доступных в шаблоне, называется *контекстом*.

Поскольку на одной и той же странице могут встречаться участки, посвященные разным итемам, (например на странице рубрики может быть список статей, каждый из элементов которого соответствует итему статьи), в процессе выполнения шаблона контекст может меняться.

Основым способом изменения контекста является формирование *списка* итемов которые нужно показать. В процессе показа списка будут сформированы контексты для каждого итема, входящего в этот список.

Естественно, что при смене контекста меняются только атрибуты итема. Атрибуты запроса остаются неизменными в процессе всего формирования страницы.

Поскольку по окончании обработки списка мы возвращаемся в исходный контекст, этот контекст сохраняется и в процессе обработки списка. Он называется *объемлющим* контекстом. Существует возможность обращения к атрибутам объемлющих контекстов, используя специальный синтаксис (см. раздел 4.1).

Язык шаблонов позволяет определять новые атрибуты в самом шаблоне. Эти атрибуты рассматриваются как атрибуты запроса, и не исчезают при переходе в контекст другого итема.

Поскольку Communiware не является императивным языком программирования, атрибуты контекста, даже созданные в самом шаблоне, нельзя рассматривать как переменные, значения которых можно менять по усмотрению программиста. Результат двухкратного определения одного и того же атрибута в шаблоне не определен.

1.4 Что такое динамический элемент

Динамический элемент представляет собой конструкцию вида

```
<:Имя параметр параметр ...:>
```

которая может появиться в тексте шаблона в любом месте, где может появиться HTML тэг. Результат использования динамического элемента внутри тэга, например

```
<a href="mailto:<:Attr EMAIL:>">послать письмо</a>
```

не определен. В ситуации когда необходимо подставить значение нескольких параметров обычного HTML-тэга правильнее воспользоваться динамическим элементом Subst и сформировать тэг целиком.

```
<:Subst "<a href=\"mailto:@EMAIL\">послать письмо</a>":>
```

Некоторые динамические элементы имеют блочную структуру, например

```
<:Loop filter:>
произвольный фрагмент шаблона,
содержащий в том числе и <EM>тэги</EM>,
и <:Subst "динамические элементы":>.
<:EndLoop:>
```

Динамический элемент может занимать несколько строк, но каждый параметр должен умещаться на одной строке.

Параметры динамического элемента разделяются между собой пробелами. Внутри параметров производится подстановка значений атрибутов, т.е. конструкция вида `@ИМЯ_АТРИБУТА` заменяется на значение этого атрибута.

Формальные правила разбора параметров и подстановки описаны в разделе 4.1.

1.5 Простейший шаблон

Рассмотрим пример использования динамических элементов для создания простейшей страницы.

```
<html><head>
<title>Простейший шаблон для итема <:Attr ITEM_TYPE:>
</title></head><body>
<H1><:Attr TITLE:></H1>
<blockquote>
<:Attr ABSTRACT:>
</blockquote>
<p align=right>
Дата написания <:Attr WRITTEN "%d-%m-%Y":>
</p>
<:Text:>
</body>
</html>
```

В первой строке этого шаблона формируется заголовок HTML-документа. Как мы видим, между заголовком и телом документа с точки зрения шаблона нет никакой разницы. И то и другое — части HTML-документа, в которых используются определенные тэги, и можно использовать динамические элементы для вывода информации об итеме.

Атрибут контекста выводится в HTML с помощью динамического элемента Attr. Этот динамический элемент получает в качестве первого параметра имя атрибута — не значение, подставленное с помощью символа `@`, а именно имя, и выводит его. В качестве второго, необязательного, аргумента Attr получает спецификацию формата, с помощью которой форматируется выводимое значение. Подробнее о том, какие спецификации формата допустимы для каких типов атрибутов рассказано в разделе 6.3.5.

Кроме Attr, для вывода значений атрибута можно использовать динамический элемент Counter (см. раздел 7.1.2). Он предназначен исключительно для вывода числовых атрибутов, и позволяет сопроводить число счетным словом в корректной форме (21 статья, 2 статьи, 20 статей).

Основное содержимое итема — его текст, выводится с помощью специального динамического элемента Text. Дело в том, что в текущей версии Communiware текст не является полноправным атрибутом итема — его нельзя подставлять с помощью `@`, нельзя использовать в качестве параметра фильтров и так далее. Для вывода текста применяются специальные динамические элементы Text и Source.

1.6 Включаемые шаблоны. Виртуальные страницы

1.6.1 Включение стандартного фрагмента

HTML-страница, сгенерированная нашим первым шаблоном выглядит некрасиво. В ней нет никаких параметров оформления, не задан даже фоновый цвет.

Но задавать параметры оформления в каждом шаблоне — непроизводительно. Ведь все страницы сайта должны иметь общий стиль оформления. Хотелось бы иметь общий шаблон заголовка HTML, который бы использовался во всех шаблонах сайта. Communiware предоставляет такую возможность. Динамический элемент Include позволяет включить в текст генерируемой страницы результат обработки указанного шаблона. Даже больше — указанный шаблон будет обработан в текущем контексте, и все изменения контекста, сделанные во включаемом шаблоне, будут доступны в нашем шаблоне.

Модифицируем наш шаблон так, чтобы он использовал шапку и из стандартного набора шаблонов:

```
<HTML><TITLE><:Attr TITLE:><BODY>
<:Include def_page_title_e:>
<blockquote>
<:Attr ABSTRACT:>
</blockquote>
<p align=right>
Дата написания <:Attr WRITTEN "%d-%m-%Y":>
</p>
<:Text:>
</body></html>
```

Как мы видим, наш шаблон даже несколько укоротился, а на результирующей странице помимо стилевого оформления появились еще и стандартные навигационные меню.

1.6.2 Включение стандартного обрамления

К сожалению, мы так и не смогли изменить атрибуты тэга body. Проблема в том, что тэг всегда должен быть закрыт в том же шаблоне, что

и открыт. (иначе слишком велика вероятность ошибки, из-за которой страница может быть неправильно показана в браузере).

Поэтому нужно средство включения стандартных фрагментов, состоящих из открывающей и закрывающей части. Такое средство называется Surround (см. раздел 7.4.5.

Так что наш шаблон следует переписать так:

```
<:Surround def_decor_s:>
<:Include def_page_title_e:>
<blockquote>
<:Attr ABSTRACT:>
</blockquote>
<p align=right>
Дата написания <:Attr WRITTEN "%d-%m-%Y":>
</p>
<:Text:>
<:EndSurround:>
```

Обратим внимание на то, что если шаблон большой, и в нем имеется несколько динамических элементов Surround, то без специальной поддержки синтаксиса Communiware в редакторе[2], разобраться к какому из Surround относится данный EndSurround непросто.

Но у закрывающего элемента EndSurround (как и у всех остальных закрывающих частей блоковых динамических элементов Communiware) можно указывать параметры.

```
<:EndSurround def_decor_s:>
```

Эти параметры не имеют никакого значения для обработчика шаблонов, но человеку, читающему шаблон, могут помочь разобраться в его структуре.

1.6.3 Виртуальные страницы

Впрочем, если мы посмотрим в текст любого стандартного шаблона, например `def_article_t` то мы не увидим там ссылки на шаблон `def_decor_s`. Вместо нее там присутствует конструкция

```
<:Surround *decor:>
```

[2] такая поддержка сейчас существует только для vim 6.0

Это так называемыя ссылка на *виртуальные страницы*. Ее следует читать как «включить сюда шаблон, выполняющий функцию стандартного заголовка на данном сайте». Какой конкретно это будет шаблон, решает разработчик сайта, назначая шаблон на роль `decor` с помощью интерфейса виртуальных страниц, доступного через соответствующий пункт системного меню.

Такой подход позволяет использовать одни и те же шаблоны на разных сайтах, поскольку включение уже стандартного заголовка и подвала придает странице правильный стиль оформления, а кроме заголовка и подвала виртуализованы и некоторые другие часто встречающиеся элементы страниц.

Виртуальная страница совершенно не обязана быть шаблоном. Например, существует виртуальная страница `help`, представляющая собой итем типа «Книга».

Среди виртуальных страниц следует выделить `pictures`. Это итем, в который помещаются элементы оформления, общие для многих шаблонов сайта — не только картинки, но и css и библиотеки функций JavaScript.

Динамические элементы `Image` и `Lib`, работающие с этими объектами, по умолчанию работают именно с этим итемом.

1.6.4 Типы шаблонов

В данном разделе мы столкнулись уже с тремя разными шаблонами. Один, тот, который мы писали, генерирует целую HTML-страницу, другой, который включается с помощью динамического элемента `Include`, генерирует последовательность законченных HTML-тэгов, третий - обрамление вокруг некоторого содержимого.

Это так называемые *типы шаблонов*.

В Communiware различаются следующие типы шаблонов:

PAGE шаблон полной страницы. Должен генерировать полный HTML документ (или документ в другой XML/SGML DTD).

ELEMENT Генерирует фрагмент страницы, состоящий из последовательности тэгов (все открытые в таком шаблоне тэги, в нем же должны быть закрыты.

Шаблоны этого типа включаются в шаблон станицы с помощью динамических элементов `Include` и `List`.

CONTAINER Шаблон, который позволяет создать тэг, окаймляющий, кусок включаемого шаблона. Используется с помощью динамического элемента `Surround`.

MAIL Шаблон почтовой рассылки. Так же, как и шаблон типа PAGE генерирует цельный документ. Разница заключается в том, что этот шаблон содержит информацию, позволяющую демону рассылки решить, нужно ли в текущий момент рассылать письмо, сгенерированное по данному шаблону или нет.

1.7 Использование списков

Шаблон, рассмотренный в предыдущем разделе — вполне работоспособный шаблон для итема, основной смысл которого в его тексте. Теперь попробуем разработать шаблон для итема, основной смысл которого в связях с соседними итемами. Типичным примером такого итема является рубрика.

Рубрика имеет заголовок и аннотацию, как правило, не имеет текста, и на ее странице должен показываться список итемов, связанных с ней связью типа TOPIC. Естественно, заголовки статей и подрубрик в этом списке должны быть гиперссылками на страницы самих статей.

```
<:Surround *decor:>
<H1><:Attr TITLE:></H1>
<blockquote>
<:Attr ABSTRACT:>
</blockquote>
<:Loop {TOPIC<-}:>
<HR>
<:Attr ITEM_TYPE:><:ItemLink ITEM_ID @TITLE:><BR>
<:Attr ABSTRACT:>
<:EndLoop:>
<:EndSurrount *decorm:>
```

Этот шаблон отличается от предыдущего только тем, что находится между `<:Loop:>` и `<:EndLoop:>`, то есть собственно списком связанных итемов, появившимся там, где в предыдущем шаблоне был текст.

Динамический элемент Loop формирует список итем с помощью *фильтра*, указанного первым аргументом, и отображает каждый из

этих итем с помощью фрагмента шаблона, указанного между Loop и EndLoop.

Как правило, на сайте существует много списков, которые должны быть показаны единообразно. Например, список статей может появиться и на странице рубрики, и на странице хронологического списка статей, и на странице новых поступлений. Поэтому существует динамический элемент List, который отображает каждый элемент списка с помощью шаблона, указанного по имени.

Например, воспользовавшись стандартным шаблоном `def_material_e` мы могли бы представить наш список следующим образом:

```
<dl>
<:List {TOPIC<-} article_e:>
</dl>
```

(шаблон `article_e` отображает каждый элемент списка как элемент HTML-ного definition list, поэтому мы поставили тэг DL вокруг списка).

1.7.1 Фильтры

Наиболее важным новым понятием в этом шаблоне является *фильтр*

Рассмотрим его поподробнее. Фильтр это способ отбора итемов.

Простейший фильтр представляет собой список явно перечисленных ITEM_ID в круглых скобках.

Остальные фильтры представляют собой список условий, которым должен удовлетворять итем.

Условия могут накладываться либо на атрибуты итема, либо на его связи.

Именованные фильтры

Очень часто полезно использовать одно и то же сложное условие отбора в нескольких местах. Communiware позволяет сохранить фильтр в базе данных и обращаться к нему по имени.

Именованные фильтры помимо собственно условия отбора содержат описание параметров. Параметры это значения, которые необходимо подставить в условия запроса, чтобы получить интересующий нас результат.

В большинстве случаев, это значения некоторых атрибутов контекста. Так, например, использованный нами фильтр PassiveLinked использует значение ITEM_ID текущего контекста, чтобы получить список итем, связанных *с текущим*. Но в некоторых случаях удобно использовать один и тот же фильтр, явно указывая ему какой-либо параметр в шаблоне.

В списке параметров фильтра на формальные параметры ссылаются как на атрибуты контекста с именами ARG1, ARG2 и тд.

Именованные фильтры можно сопровождать дополнительными условиями.

Например, если у нас есть фильтр MyFilter, которому в качестве параметра передается имя связи, мы можем ограничить его результаты только итемами типа ARTICLE вот таким способом:

```
{MyFilter(TOPIC) | TYPE_ID = ARTICLE}
```

SQL-фильтры

В некоторых случаях выразительной мощности языка фильтров Communiware не хватает для описания требуемого набора условий. Поэтому Communiware позволяет использовать в качестве фильтров непосредственно SQL-запросы к нижележащей базе данных.

SQL-запросы довольно многословны, к тому же их синтаксис зачастую зависит от используемой базы данных. Поэтому непосредственно в шаблонах Communiware SQL-запросы не используются. SQL-фильтры могут быть только именованными. Кроме того, их нельзя сопровождать дополнительными условиями.

Зато результатом работы SQL-фильтра не обязательно должен быть список итемов. Это может, например, быть список допустимых значений атрибута, или список атрибутов.

Кроме того, язык SQL позволяет возвратить в каждом элементе результирующего списка несколько значений (колонок таблицы). Все эти значения, возвращенные запросом SQL-филтьтра доступны в шаблоне как атрибуты элемента списка.

Это позволяет, например, тем же запросом, который формирует список рубрик, посчитать количество статей в рубрике.

Perl-фильтры

Существуют ситуации, в которых возможностей языка SQL тоже не хватает, например при необходимости получить список иллюстраций данного итема, так как иллюстрации хранятся не в базе данных, а просто на диске, или при необходимости взаимодействия с системой полнотекстового поиска.

Для этой цели в Communiware предусмотрена возможность использовать в качестве фильтра объект на языке Perl. Разработка таких фильтров выходит за рамки деятельности обычного Communiware-разработыика, и является задачей программиста. Поэтому в распоряжении разработчика имеются только те perl-фильтры, которые входят в стандартный комплект (см. раздел 8.1) или в состав Communiware-пакетов, установленных на текущем сервере. Синтаксически perl-фильтр отличается от любого другого именованного фильтра тем, что его имя всегда начинается с двух двоеточий.

1.7.2 Сортировка списков

По умолчанию, Loop и List выводят итемы списка в том порядке, в каком они возвращаются фильтром. Это далеко не всегда удобно. Очень часто нужно вывести итемы, например по алфавиту, или в порядке публикации.

Поэтому в динамических элементах Loop и List предусмотрен параметр, задающий критерий сортировки. Критерий сортировки обычно представляет собой список атрибутов через запятую, каждый из которых сопровождается знаком + или -, задающим прямой или обратный порядок сортировки.

Особым случаем сортировки является «ручная» сортировка, т.е. предоставление модератору возможности указания порядка итемов вручную.

Эта сортировка задается посредством указания критерия сортировки `+ORDNUM`. ORDNUM не является атрибутом итема (одна и та же статья может входить в две рубрики, и иметь в них разное положение), а является атрибутом связи между итемами. Поэтому использовать его для сортировки можно только в том случае если в условие фильтра входит отбор итемов, связанных с некоторым итемом определенной связью.

Еще один специфический случай — это *древовидная сортировка* (см раздел 7.6.1. Необходимость в ней возникает потому, что достаточно часто требуется отобразить не только непосредственных потомков

данного итем по данной связи, но и потомков этих потомков, и при этом отразить иерархические взаимоотношения между ними.

При древовидной сортировке задаются два атрибута, обеспечивающие связь между предками и потомками, первый из которых — уникальный идентификатор текущего объекта (обычно ITEM_ID) и второй, задающий непосредственного предка, по используемой иерархии (в стандартных фильтрах он называется PARENT_ID) и критерий сортировки „братьев и сестер" — итемов на одном уровне иерархии. Поскольку древовидная сортировка всегда предполагает отбор по определенной связи, в качестве критерия сортировки на одном уровне можно использовать ORDNUM (при этом предполагается, что нас интересует порядковый номер связи итема с тем итемом, имя которого попало в PARENT_ID.

Необходимо отметить, что далеко не всякий атрибут, значение которого доступно в контексте, может быть использован в качестве критерия сортировки.

В случае обычного фильтра доступны только стандартные и расширенные атрибуты данного итема. В случае SQL-фильтра доступны все колонки, непосредственно возвращаемые фильтром, а если фильтр имеет флаг "переписываемый", то и стандартные и расширенные атрибуты.

1.7.3 Разрезание длинных списков на страницы

Веб-страница обычно должна иметь приемлемые размеры. Слишком большие странцы долго грузятся по медленным каналам, а при наличии табличного дизайна — долго форматируются. В то же время списки однородных итемов (например, статей на определенную тему) могут состоять из нескольких сотен элементов. Конечно, хороший модератор не допустит того, чтобы рубрика выросла до необозримых размеров. Но бывают списки, которые формируются автоматически, например, результаты поисковых запросов, или, во всяком случае, не допускающие содержательного разбиения на части, например, активная дискуссия. Страницы такого типа принято делить на части по какому-то формальному критерию, и указывать в каждой части ссылку на продолжения. Многие системы формируют список страниц со ссылками на каждую из них. Возможности Communiware в этом плане несколько более ограничены. Можно сформировать ссылку только на следующую страницу и на самое начало списка.

Другой часто употребляемый случай — список из которого нас интересуют только несколько первых элементов, например список пяти

самых свежих статей на сайте можно представить как список всех статей в порядке их возраста, от которого мы показываем только пять первых элементов.

Обрезанием списков управляет пятый параметр динамического элемента List (соответственно, четвертый у Loop) который может быть просто числом или конструкцией число восклицательный знак.

Если используется просто число, то это фрагмент разрезанного на страницы списка. Показываются n элементов, а то, с какого начинать, зависит от наличия в контексте запроса параметра `NEXT_ITEM`. Первым будет показан (если он есть в списке) итем, идентификатор которого содержится в этом параметре.

Ссылку на следующую страницу формирует динамический элемент Continuation. Он должен располагаться после списка (иначе неизвестно, какие элементы были показаны на этой странице) в том же контексте итема, что и сам список.

Ссылку на начало списка тоже формирует Continuation.

Вот пример шаблона, использующего разрезаемый список:

```
<:Include *page_top:>
<:Loop All_Items(ARTICLE) -PUBLISHED "" 10:>
<H1><:Attr TITLE:></H1>
<BLOCKQUOTE>
<:Attr ABSTRACT:>
</BLOCKQUOTE>
<:EndLoop:>
<:Continuation start "Начало списка":>
<:Continuation next "Следующая страница:>
<:Include *page_bottom:>
```

На странице может быть только один список, для которого возможны продолжения. Если рядом должны располагаться какие-то другие списки, показывающие только несколько первых элементов используйте для их обрезания нотацию с восклицательным знаком.

```
<H2>Новости сайта</H2>
<:List All_Items(NEWS) news_e -PUBLSHED "" 5!:>
```

Списки, обрезанные таким образом, не обращают внимания на параметр `NEXT_ITEM` и не влияют на соседствующие Continuation.

1.8 Групповые атрибуты

При выводе списков (а иногда и вместо него) очень полезно бывает знать некоторые их общие свойства. Например, для того, чтобы решить, показывать или нет список подрубрик данной рубрики, полезно выяснить, есть ли у нее подрубрики (поскольку кроме собственно элементов-подрубрик, список может содержать некоторое внешнее оформление, которое должно располагаться вне динамического элемента List.

Для этой цели существуют групповые атрибуты, или атрибуты с параметрами. Эти атрибуты принимают значение некоего свойства не одного итема, а некоторой группы итемов, связанных с текущим. То, как они связаны, задается с помощью параметров этих атрибутов. При этом можно задать тип связи, типы итемов, входящих в группу, набор статусов и диапазон дат написания или последнего изменения.

Например, формируя список рубрик мы можем указать количество итемов (статей, внешних ссылок и книг в них).

```
<!-- Шаблон для элемента списка рубрик-->
<H1><:Attr TITLE:></H1>
Содержит
<:Counter CHILDREN(TOPIC-,[ARTICLE,BOOK,EXTERNAL])
материал(|а|ов):>
На эти материалы поступило
<:Counter CHILDREN(BELONGS,REPLIC) реплик(а|и|):>,
последняя <:Attr LAST_PUBLISHED(BELONGS,REPLIC):>
```

Заметьте, что посчитывая число итемов, непосредственно связанных с данным связью TOPIC мы употребили конструкцию `TOPIC-` — в случае иерархической связи или метасвязи, она ограничивает группу только непосредственными соседями, а подсчитывая число и дату последних изменений реплик, связанных через цепочку итемов метасвязью `BELONGS`, обошлись без минуса, объявив таким образом, что нас интересует поиск на всю глубину.

1.9 Условные операции

Шаблоны, определенные в предыдущем разделе плохи тем, что в случае, если в подрубрике нет никаких реплик, будут формировать текст, подобный

Условные операции

Содержит 5 материалов. На них поступило 0 реплик, последняя

Для того, чтобы подобные конструкции выглядели по-настоящему красиво, требуется умение пропускать определенные части шаблона, в зависимости от выполнения некоторых условий.

Для этой цели предназначен динамический элемент <:If:>

Выглядит он следующим образом:

```
<:If условие:>
фрагмент шаблона, который покажется,
если условие выполняется
<:Else:>
фрагмент шаблона, который покажется,
если условие не выполняется
<:EndIf:>
```

Часть <:Else:> можно опустить. Тогда код, выполняемый в случае истинности условия будет заканчиваться непосредственно EndIf.

Возможные варианты условий описаны в разделе 4.3. Сейчас мы остановимся на простейшем из них — просто указании имени атрибута. Это выражение истино, если атрибут существует и имеет значение, отличное от 0 или пустой строки.

Поэтому улучшенный вариант шаблона из предыдущего раздела может выглядеть так:

```
<!-- Шаблон для элемента списка рубрик-->
<H1><:Attr TITLE:></H1>
Содержит
<:Counter CHILDREN(TOPIC-,[ARTICLE,BOOK,EXTERNAL])
материал(|а|ов):>
<:If CHILDREN(BELONGS,REPLIC):>
На эти материалы поступило
<:Counter CHILDREN(BELONGS,REPLIC) реплик(а|и|):>,
последняя <:Attr LAST_PUBLISHED(BELONGS,REPLIC):>
<:EndIf:>
```

1.10 Вверх по стеку контекстов

В некоторых случаях внутри элемента списка, или в другом вложенном контексте, полезно знать, а что за свойства имеет итем уровнем выше.

Рассмотрим, например, шаблон главы книги. Для того чтобы внутри этого шаблона сформировать оглавление книги, нужно найти итем книги, потом в его контексте построить список всех потомков по связи PART.

Все элементы этого списка, должны быть ссылками на соответствующие итемы. Но если этот элемент списка соответствует текущей странице, то формировать ссылку бессмыслено, а лучше бы выделить этот элемент визуально, чтобы пользователь сразу видел в каком месте книги он находится.

Т.е. обрабатывая элемент списка глав нам нужно посмотреть на два контекста выше и сравнить текущий `ITEM_ID` с `ITEM_ID` объемлющего контекста.

Это можно сделать посредством следующего фрагмента шаблона:

```
<!-- Здесь контекст главы книги-->
<!-- найдем самого дальнего предка по связи PART - саму книгу-
<:Loop AllActiveLinked(PART) -DISTANCE "" 1!:>
<!-- Здесь контекст книги --->
<UL>
<:Loop LinkedTree(PART)
t:ITEM_ID,PARENT_ID,ORDNUM LEVEL:tag(UL):>
<!-- Здесь контекст текущего элемента оглавления-->
<LI><:If ITEM_ID = @ITEM_ID#2:><B><:Attr TITLE:></B>
<:Else:><:ItemLink ITEM_ID @TITLE:><:EndIf:>
<:EndLoop:>
<!-- Контекст книги-->
<:EndLoop:>
<!-- Контекст главы-->
```

Как мы видим, для того, чтобы сослаться на контекст несколькими уровнями выше, нужно всего лишь в конце имени атрибута написать символ `#` и число контекстов, на которое нужно подняться.

1.11 Создаем собственные атрибуты

В некоторых случаях, имеет смысл „запомнить” некоторое значение в процессе обработки шаблонов, и потом его использовать наравне с прочими атрибутами контекста.

Так, например, если включаемый шаблон производит какое-то действие, которое при включении другого шаблона стоило бы производить в основном шаблоне, можно во включаемом шаблоне выставить флаг, который проверять во включающем.

Для этих целей применяется динамический элемент <:Define:> (см. раздел 7.5.1).

Рассмотрим такой пример: Допустим, у нас есть два шаблона верхней шапки, один из которых содержит левое меню, а второй нет. Соответственно, ширина ячейки таблицы, в которой должно поместиться основное содержимое страницы, в первом случае будет 560 пикселов, а во втором — 780.

И есть третий включаемый шаблон, который создает таблицу-заголовок, которая должна иметь ширину основного содержимого.

Шаблоны будут выглядеть так:

Первый шаблон шапки:

```
..........
<table width=760>
<tr valign=top>
<td><!-- Левое меню-->
,..
</td>
<td width=560>
<:Define TABLE_WIDTH 560 NUMBER:>
```

Второй шаблон шапки:

```
......
<table width=760>
<tr><td>
<:Define TABLE_WIDTH 760 NUMBER:>
```

Шаблон, включаемый в основное тело страницы:

```
<:Subst "<table width=@TABLE_WIDTH>":>
....
```

Как мы видим, атрибуты, созданные Define ничем не отличаются от любых атрибутов HTTP-запроса.

Другое применение динамического элемента Define — установка значений по умолчанию для пользовательских настроек, вводимых через формы ввода или хранящихся в куках (см раздел 2.2.2). В этих случаях необходимо чтобы Define создавал значение атрибута только в том случае, если этого атрибута еще нет в контексте, и определял тип атрибута в любом случае.

Если указать Define параметр ifabsent, он будет вести себя таким способом. Например:

```
<:Define PRINT_VIEW 0 NUMBER ifabsent:>
```

Существует еще конструкция <:Define ... ifempty:> которая переопределяет атрибут не только если он не существует вообще, но и если он существует, но имеет пустое значение.

Если надо указать атрибут контекста, который будет действовать не на текущей странице, а после прохода по определенной ссылке, то этот атрибут указывается непосредственно при формировании ссылки.

Например, ссылку на текущий итем, с созданием в его контексте атрибута `PRINT_VIEW` можно сформировать следующим образом:

```
<:ItemLink ITEM_ID "Вид для печати" "" ""
PRINT_VIEW=1:>
```

Аналогичным образом задаются параметры и в других динамических элементах, формирующих гиперссылки (Script (7.2.3), Continuation (7.6.3)).

В некоторых случаях возникает задача использования пользовательских настроек, введенных через форму, на нескольких страницах. Иногда этого можно добиться при помощи кук, но в некоторых ситуациях, когда настройки должны действовать в области, не являющейся с точки зрения браузера поддеревом URL, например в рубрике, настройки приходится пробрасывать явно.

Это можно сделать указывая все атрибуты, которые необходимо пробросить, в пятом параметре ItemLink. Если там указать параметр, не сопроводив его значением, будет использовано значение из текущего контекста.

Но такой подход часто неудобен, так как требует указания параметров настройки во всех элементах генерирующих ссылки.

Динамический элемент PassParams (7.5.2) позволяет указать список параметров, которые будут пробрасываться по умолчанию.

Глава 2

Формы ввода в Communiware

Формы ввода являются основным способом взаимодействия пользователя с web-сайтом, если это взаимодействие выходит за пределы выбора из нескольких предложенных вариантов, которое легко реализовать в виде списка ссылок.

Форма ввода в HTML представляет собой тег form, внутри которого присутствуют тэги input, select и textarea, соответствующие отдельным полям ввода. После того, как пользователь введет какие-то значения в эти поля и нажмет кнопку сабмита формы (`<input type=submit`), содержимое этих полей будет отправлено на сервер по URL, указанной в атрибуте action у тэга form. Если action не указан, то результаты заполнения формы будут отправлены на ту же URL, с которой пришла форма.

Стандарт HTTP предусматривает два метода отправки результатов заполнения форм — GET и POST. Метод GET ничем не отличается от непосредственного указания соответствующих полям ввода значений в URL, и поэтому имеет весьма жесткие ограничения на объем вводимой информации. Метод POST позволяет отправлять большие (до нескольких мегабайт) объемы информации.

При обработке результатов ввода в формы Communiware не делает различий между GET и POST, поэтому в случае любых сомнений используйте POST.

На Communiware-сайте могут встретиться следующие типы форм:

1. Формы, добавляющие введенную пользователем информацию в контекст, как значения некоторых атрибутов — так устроены, например, формы поиска и настроек.

2. Формы, аналогичные предыдущим, но содержащие специальные динамические элементы, каким-то образом интерпретирующие ввод — формы подписки, регистрации и отправки итем по почте.

3. Формы, результат ввода в которые обрабатывается каким-то из интерфейсных скриптов Communiware. Они выполняют в общем-то те же функции, что и эти скрипты, но предоставляют разработчику сайта существенно больше контроля над внешним видом формы.

4. Формы постинга, результатом обработки которых является создание или изменение итем в базе данных. Работа с формами постинга достаточно объемный раздел, поэтому мы рассмотрим его в отдельной главе 3.

2.1 Простые формы

Простые формы создаются просто прописыванием в HTML тэга `<form>`. В таком виде результат заполнения формы будет отправлен на ту же URL, с которой была получена форма. Просто в контексте появятся появятся атрибуты, соответствующие полям формы.

Если результат заполнения формы нужно отправить на другой шаблон (или другой итем), открывающий тэг form нужно сгенерировать при помощи динамического элемента Subst.

Например, типичная форма поиска может выглядеть так:

```
<:Subst "<form action=@URL_PREFIX/@ITEM_ID/search_t":>
<input type=text name=SEARCHFOR>
<input type=submit name=OK value="Найти">
</form>
```

В результате ввода поисковой строки в эту форму и нажатия кнопки «Найти» будет показан текущий итем по шаблону `search_t`, при этом в контексте появятся два дополнительных атрибута: SEARCHFOR со значением, равным введенной строке поиска, и OK со значением „Найти”.

Шаблон `search_t` должен уметь интерпретировать атрибут SEARCHFOR. (что он и делает, поскольку этот атрибут используется фильтром TitleSearch, применяемом в этом шаблоне).

Для создания элементов ввода рекомендуется использовать динамический элемент Input (см. раздел 7.7.2), а не HTML-тэг input, поскольку он автоматически использует значения атрибута в контексте в качестве значения по умолчанию в поле ввода, и кроме того, помещает указанное значение по умолчанию в контекст, если форма показана в первый раз и введенного пользователем значения по умолчанию в контексте еще нет.

Кроме того динамический элемент Input позволяет конструировать выпадающие меню и списки чекбоксов и радио-кнопок с помощью фильтров.

Таким образом, вышеприведенный фрагмент шаблона лучше переписать так:

```
<:Subst "<form action=@URL_PREFIX/@ITEM_ID/search_t":>
<:Input text SEARCHFOR:>
<:Input submit OK "Найти":>
</form>
```

Если этот фрагмент шаблона включен, например в шапку страницы, и появится в шаблоне `search_t`, то на странице результатов поиска пользователь увидит в форме введенную им строку запроса.

Еще одной полезной особенностью динамического элемента Input является возможность форматированного ввода дат. Данные типа «дата» в Communiware хранятся в формате YYYY.MM.DD HH.MM.SS. Этот формат имеет много преимуществ с точки зрения внутренней обработки, например данные в таком формате можно сортировать лексикографически, как строки, но крайне непривычен для пользователя. Особенно, если учесть что даты почти всегда выводятся пользователю в отформатированном виде. Поэтому для ввода дат нужно пользоваться конструкцией

```
<:Input date ИМЯ значение формат:>
```

которая позволяет пользователю редактировать дату, представленную почти по любому формату функции `strftime`. Недопустимы только форматы с буквенным представлением названий месяцев.

2.2 Формы со специальной обработкой

Кроме обычных элементов ввода в Communiware есть элементы ввода, специальным образом обрабатывающе введенные значения. В принципе, таким элементом ввода является рассмотренный в предыдущем разделе `<:Input date:>`, так как в контекст попадает не то значение, которое было введено пользователем, а результат его переформатирования.

Рассмотрим и другие динамические элементы, специальным образом обрабатывающие ввод.

Следует сразу обратить внимание, что динамический элемент может так или иначе обработать ввод только в том случае, если результат ввода будет отправлен на ту же URL, с которой получена форма. Т.е. если обработка ввода производится динамическим элементом, она производится в момент генерации страницы, показываемой пользователю в ответ на нажатие кнопки сабмит. Это не касается `<:Input date:>`, поскольку форматирование введенных дат производится ядром Communiware до начала обработки шаблона.

2.2.1 Регистрация пользователя

Динамический элемент Login создает поле для ввода пароля В одной форме с ним должен присутствовать элемент ввода с именем `AUTHOR_ID`, в который вводится имя пользователя.

Если в начале обработки страницы ядро Communiware обнаруживает, что значение `AUTHOR_ID` указано не в куке, (или не только в куке) а среди параметров формы, то оно проверяет наличие пароля, введенного через динамический элемент Login, и в случае совпадения пароля, соглашается изменить `AUTHOR_ID` на указанный, выдавая пользователю новую куку.

Если пароль неправильный, то сообщение об ошибке помещается в атрибут контекста ERROR, а значение AUTHOR_ID не изменяется.

Динамический элемент Login работает только в том случае, если форма, в которой он расположен, использует метод POST, так как в противном случае пароль будет виден в URL.

Пример:

```
<form method=POST>
<:If ERROR:>
<:Attr ERROR:>
```

```
<:EndIf:>
Username: <:Input text AUTHOR_ID @AUTHOR_ID:><br>
Password: <:Login:><br>
<:Input submit DOIT "Login":>
</form>
```

2.2.2 Запоминание параметров в куках

Очень удобным способом хранения индивидуальных настроек являются куки (Cookies). Куки это пары имя=значение, которые запоминаются браузером клиента и передаются им на сервер, если имя сервера и URL соответствуют области действия данной куки.

Для того чтобы создать куку, сервер должен выдать клиенту HTTP-заголовок Set-Cookie, содержащий имя куки, ее значение и информацию об области и времени действия.

Поскольку куки хранятся в браузере клиента, на сервере может не быть никакой информации о данном пользователе, т.е. настройки через куки доступны даже для анонимного пользователя.

Communiware помещает все присланные пользователем куки в контекст в виде атрибутов, поэтому разработчик имеет возможность хранить в куках произвольные настройки. Явным образом указанный в URL или введенный пользователем в форме атрибут с тем же именем имеет приоритет над значением куки.

Запоминание атрибутов контекста (например, введенных пользователем в форме) в куках — функция динамического элемента `<:Input submit:>` поскольку он обладает возможностью отличить, генерируется ли данная страница в результате сабмита формы или нет.

Динамический элемент `<:Input submit:>` имеет два параметра, отвечающих за установку кук: список атрибутов контекста, которые надо запомнить, и область действия кук. Область действия принимает следующие значения: server — физический сервер Communiware — все виртуальные сайты с тем же hostname, что и текущий и каталог скриптов, site — текущий виртуальный сайт (кроме скриптов), item — текущий итем, независимо от того по какому шаблону он показан, exact — текущий итем, показанный по текущему шаблону. По умолчанию используется site,

Срок действия кук всегда устанавливается достаточно большим.

Следует учесть, что стандартные скрипты Communiware, такие как userprefs, используют область действия server, а при наличии в браузере

нескольких кук с одинаковым именем используется та, которая имеет наиболее ограниченную область действия. Поэтому, если вы переопределите один из стандартных параметров настройки, не указав область действия server, вернуть его на место через стандартный скрипт будет невозможно.

Пример формы, устанавливающий атрибуты FOO и BAR в виде куки:

```
<form>
Введите значение FOO для запоминания
<:Input text FOO:><br>
Выберите значение BAR
<:Input radio BAR 1:>1
<:Input radio BAR 2:>2
<:Input radio BAR 3:>3
<:Input submit SET Установить "" FOO,BAR server:>
</form>
```

Пользоваться значениями запоминаемых атрибутов в шаблоне можно как до, так и после `<:Input submit:>`, так как он не изменяет их значений, а просто инструктирует браузер пользователя запомнить эти значения.

Communiware позволяет установить куки не только в результате сабмита формы, но и просто при показе пользователю страницы. Для этого можно воспользоваться динамическим элементом Header (см. раздел 7.3.1), который позволяет выдать пользователю любой HTTP-заголовок, в том числе и заголовок Set-Cookie.

2.2.3 Управление подпиской

Механизм подписки на обновления в Communiware использует специальные структуры в базе данных, которые не являются ни итемами, ни связями.

Запись о подписке чем-то напоминает связь, но в отличие от связи, связывает не два итема а три — пользователя, которому посылаются обновленные итемы, итем, индицирующий раздел сайта, обновления в котором интересуют данного пользователя, и шаблон подписки определяющий, какую информацию следует слать. Понятие раздела сайта не определено жестко, реально его определяет фильтр подписки, указанный в атрибуте PARAMS шаблона подписки.

Кроме того подписка характеризуется периодичностью, с которой проверяется наличие итем, подлежащих отсылке, и форматом, который может быть либо plain text, либо HTML. Шаблон подписки всегда генерирует HTML-документ, но если пользователь предпочитает получать обычный текст, этот HTML перед отправкой конвертируется в текст при помощи программы `w3m`.

Шаблон подписки похож на шаблон страницы, но обладает рядом дополнительных свойств. Во-первых, страница, которую генерирует этот шаблон будет просматриваться пользователем при чтении почты, возможно off-line, поэтому рекомендуется избегать графических элементов оформления, или даже подавлять иллюстрации в текстах, если в шаблон рассылки включается полный текст итем. Во вторых, страница, сгенерированная шаблоном рассылки может быть сконвертирована в текст еще на сервере (причем решение об этом принимает пользователь, и у разработчика нет никаких средств запретить это). Поэтому важные гиперссылки должны быть представлены таким образом, чтобы URL была видна в тексте. Т.е. осмыслено применять конструкции вида:

```
<H1><:Attr TITLE:><H1>
<:ItemLink ITEM_ID "<TT>@URL_PREFIX/@ITEM_ID</TT>":>
<br>
```

вместо применяемой обычно в шаблонах страниц

```
<H1><:ItemLink ITEM_ID @TITLE:></H1>
```

В третьих, контекст, в котором генерируются почтовые рассылки, обладает рядом особенностей. Самой важной из них является наличие в контексте атрибутов PERIOD_START и PERIOD_END, которые задают временной интервал, обновления за который должны посылаться.

В качестве фильтра, отбирающего итем для отсылки, всегда используется фильтр использующий эти атрибуты.

Как уже упоминалось, этот фильтр не вписывается в текст шаблона, а хранится в атрибуте PARAMS, что позволяет демону почтовой рассылки — программе, выполняющей рассылку, проверить наличие итем и не выполнять обработку шаблона совсем, если итем, подлежащих рассылке, нет.

В тексте шаблона для формирования основного списка следует использовать фильтр `::Mail`, который получает от демона рассылки результаты выполнения фильтра, указанного в атрибуте PARAMS без повторного выполнения этого фильтра. Кроме того, результаты выдавае-

мые фильтром `::Mail` отфильтрованы на предмет итем, недоступных для чтения.

Обычные почтовые рассылки Communiware рассчитаны на общедоступные сайты. В них генерируется минимальное число писем — всем подписавшимся на один и тот же раздел с одинаковым периодом посылается одно и то же письмо и все письма генерируются как для анонимного пользователя — атрибут AUTHOR_ID в контексте отсутствует. Ни один итем имеющий ограничения на чтение в такие рассылки не включается.

Для сайтов с разграничением прав доступа существует так называемая персонализированная рассылка. В ней для каждого адресата генерируется отдельное письмо, в которое включаются все итемы, читать которые имеет право данный пользователь.

Форма управления подпиской должна позволять для каждой пары итем-шаблон подписки выбрать требуемый период рассылки (включая значение „не посылать”) и формат.
Эту функциональность обеспечивает динамический элемент Subscribe.

Ему указывается итем и шаблон. Если не указан период и формат, то для них создаются меню (HTML тэги select).

Шаблон формы подписки на новые реплики в текущей статьи может выглядеть так:

```
<form>
Присылать мне свежие реплики в этой дискуссии:
<:Subscribe @ITEM_ID discussion_m:>
<:Input submit OK Подписатья:>
</form>
```

Если период и формат указаны явным образом, то для них создаются скрытые поля, т.е. сам факт нажатия кнопки приводит к подписке данным способом.

Можно также указать список адресатов (при помощи фильтра).

Такая подписка используется обычно в формах постинга. Например, в системе управления проектами осмысленно подписать пользователя назначенного менеджером некоторого проекта на все обновления в данном проекте непосредственно в момент назначения.

Запросы на подписку, сформированные динамическим элементом Subscribe обрабатываются ядром Communiware до начала обработки шаблона.

2.2.4 Отправка электронной почты

Communiware позволяет отправлять электронную почту только по адресам зарегистрированных пользователей, чтобы избежать превращения Communiware-сервера в спам-машину. Разработчикам сайтов рекомендуется защищать шаблоны, содержащие форму отправки почты с помощью динамического элемента Authentify (7.3.2), чтобы отправителем почты мог быть тоже только зарегистрированный пользователь.

Динамический элемент MailTo (7.7.5) позволяет отправить текущий итем по почте. При этом требуется указать шаблон, по которому будет отформатирован данный итем при формировании письма, и список адресатов.

Шаблон по которому отправляется почта, в отличие от шаблона подписки, не имеет специального типа, а является обычным шаблоном страницы. Тем не менее, на него распространяются те же ограничения в области дизайна, что и на шаблоны подписки. Как и в шаблонах подписки, все заголовки, созданные динамическим элементом Header, рассматриваются не как HTTP-заголовки, а как MIME-заголовки, помещаемые в письмо. В частности, с помощью конструкции

```
<:Header To some\@address:>
```

можно заменить список адресатов письма на любой текст (не обязательно корректный E-Mail адрес). Следует помнить что желание скрыть реальный список адресатов может создать проблемы для тех пользователей, почтовые системы которых получают почту сразу для нескольких пользователей из одного почтового ящика у провайдера (конечно, это свидетельствует о плохой настройке почтовой системы, но иногда случается).

Список адресатов должен представлять собой спецификацию фильтра, возвращающего требуемых пользователей. (поскольку список идентификаторов итем через запятую в круглых скобках является корректной спецификацией фильтра, адресатов можно указать и явно).

В принципе возможно послать почту по списку адресов, не связанных жестко с итемами типа «Персона». Если фильтр, используемый для формирования списка адресатов, сразу возвращает атрибут EMAIL, то будет использован адрес, содержащийся в этом атрибуте.

При использовании MailTo в обычной форме, текущим считается тот итем, в контексте которого формировалась форма, а при использовании его в форме постинга, тот итем, который изменяется или создается в

соответствующем фрагменте формы постинга. Поэтому если попытаться использовать MailTo в форме множественного постинга (см. раздел 3.4) за пределами контейнеров Item, то никакая почта послана не будет, поскольку не определён итем, который надо послать.

Не будет послана почта и в том случае, если произошла ошибка при сохранении, или контейнер пропущен в результате невыполнения условия или по действию SKIP (см раздел 3.5).

2.3 Формы, обрабатываемые интерфейсными скриптам

Скрипты Communiware это большие унифицированные блоки, реализующие какие-либо операции взаимодействия пользователя с базой данных. Как правило, функции скрипта могут быть выполнены специальным шаблоном, содержащим ту или иную форму ввода, но скрипты обладают большей гибкостью и универсальностью (правда, меньшей настраиваемостью внешнего вида) и уже включены в комплект.

Типичным примером скрипта является интерфейс вебмастера.

Скрипты отличаются тем, что разработчик сайта практически не имеет возможности повлиять на их оформление. Максимум, что позволяют некоторые интерфейсные скрипты, это включить верхнее и нижнее оформление, заданное как виртуальные итем scripts_top и scripts_bottom.

Поэтому иногда у разрабочтика сайта возникает желание создать свою собственную форму, у которой в качестве action будет указан один из стандартных скриптов Communiware. Такая возможность поддерживается, но никогда не считалась штатной. Поэтому в документации на скрипты как правило не описаны имена полей ввода, которые требуется создать, чтобы скрипт мог правильно выполнять свои функции. Эти имена приходится устанавливать путем изучения HTML-кода формы, сгенерированной скриптом.

Как правило, критически важно присутствие поля с непустым значением, имя которого соответствует имени кнопки сабмита оригинальной формы. Поэтому если вы хотите заменит кнопку скажем на `<input type=image`, необходимо создать скрытое поле с именем соответствующим имени кнопки в оригинальной форме, и значением, скажем 1.

При вызове штатным способом, через динамический элемент Script (формирующий гиперссылку на скрипт), скрипт получает ряд обязатель-

ных параметров (см. раздел 7.2.3), обеспечивающих корректное функционирование скрипта и возврат на вызывающую страницу. Для создания в форме скрытых полей, передающих эти значения, воспользуйтесь динамическим элементом Referer (раздел 7.7.1).

При обращении к скриптам (например в атрибуте action тэга form) не следует задавать префикс (часть URL перед именем скрипта) в явном виде, иначе форма может перестать работать при переносе на другой физический сервер, системный администратор которого назначил другой префикс для скриптов.

Воспользуйтесь атрибутом контекста SCRIPT_PREFIX.

Например:

```
<:Subst "<form action=\"@SCRIPT_PREFIX/login\""
 . "method=POST>":>
<:Referer:>
Имя пользователя <:Input text AUTHOR @AUTHOR_ID:><br>
Пароль: <input type=password name=PASSWD><br>
<input type=submit name=LOGIN
value="Зарегистрироваться">
</form>
```

Глава 3

Формы постинга

Формы постинга отличаются от всех прочих форм тем, что они создают или изменяют итемы в базе данных Communiware.

Одним из следствий этого является то, что эти формы должны поддерживать загрузку или редактирование больших объемов текста — собственно текстов итем. Кроме того, они должны поддерживать работу с двоичными данными, например, картинками. Поэтому они всегда используют метод POST и enctype multipart/form-data.

Кроме того, они отличаются тем, что внутри формы постинга используется контекст редактируемого или создаваемого итема. В случае создания итема этот контекст (в отличие от практически всех остальных контекстов Communiware) не содержит ITEM_ID, по той простой причине, что в момент генерации формы итем еще не создан и идентификатор ему еще не присвоен. Независимо от того, будет ли этот идентификатор введен пользователем, или сгенерирован автоматически при сохраниении итема, это произойдет после того, как форма будет показана пользователю.

Наличие двух принципиально разных стадий обработки — стадии генерации формы для показа пользователю, и стадии обработки и сохранения результатов ввода – главное отличие форм постинга от всех других контекстов Communiware.

Операция создания или изменения итема может завершиться неудачей, в том случае если введенные значения нарушают какие-либо ограничения Communiware (например, производится попытка установить связь с несуществующим итемом). Поведение в случае удачи и в слу-

чае ошибки должно быть принципиально разным. В случае ошибки, наиболее разумным поведением является показать пользователю ту же самую форму еще раз, предоставив ему возможность исправить введенные данные. В случае удачи, скорее всего нужно показать какую угодно страницу, только не ту, на которой была сама форма.

3.1 Динамический элемент Post

Для создания формы служит динамический элемент Post. Это блоковый динамический элемент, внутри которого расположен фрагмент шаблона, формирующий собственно элементы ввода.

```
<:Post @ITEM_ID anon new:>
<:EditField TITLE:><br>
<:EditField TEXT:><br>
<:Input submit SAVE "Сохранить":>
<:Input submit BACK "Отмена":>
<:EndPost:>
```

Первым параметром динамического элемента Post является объект редактирования. Это может быть либо идентификатор итема, либо указание на то, что требуется создать новый итем определенного типа, например `new(ARTICLE)`.

Вторым параметром указывается уровень доступа к данной форме. Он может принимат значения anon, registered и оставаться пустым. Если он остается пустым, то работать с этой формой имеют право только пользователи, имеющие право на запись на указанный итем. registered означает, что доступ к форме имеют все зарегистрированные пользователи, а anon — дает право на доступ в том числе и для анонимных пользователей.

Третий параметр указывает действие, которое может быть `new` — перейти на вновь соданный или отредактированный итем, `back` — вернуться на страницу, откуда пользователь попал на страницу с формой, или явное указание итема (возможно с шаблоном), на который следует перейти. Допустимо также указывать внешнюю URL, тогда она должна начинаться с `http://`. В этом параметре допустимы подстановки, выполняемые во время постинга.

Спецификация объекта для редактирования может быть пустой. Это означает, что форма постинга работает одновременно с несколькими

объектами. В этом случае все элементы ввода (кроме, возможно, кнопок сабмита) должны находиться внутри динамических элементов `<:Item:>`. Подробрее множественный постинг мы рассмотрим в разделе 3.4.

3.2 Элементы ввода в форме постинга

В формах постинга можно использовать как обычный динамический элемент Input, так и специальный динамический элемент EditField.

Динамический элемент EditField предназначен для упрощения задачи создания полей ввода или редактирования атрибута. Его поведение меняется в зависимости от типа атрибута, т.е. он правильно создает необходимый элемент ввода, который может быть либо однострочным текстовым полем, либо многострочным (для атрибута типа RICHTEXT), либо выпадающим меню, если атрибут имеет таблицу разрешенных значений.

Особый случай представляет собой EditField TEXT. Как мы видели выше, во всех контекстах, кроме контекста постинга, текст итема не рассматривается как полноправный атрибут. В случае постинга он тоже интерпретируется несколько особым образом. Во-первых, возможно редактирвание текста как в том виде, в котором он хранится в базе, так и преобразованным в стандартный коммуниверный формат текста с выделениями. Этим управляет параметр representation.

```
<:EditField TEXT "" representation=HTML:>
```

позволяет редактировать текст в виде HTML-кода, а

```
<:EditField TEXT "" representation=Text:>
```

в несколько более удобочитаемом виде. Правда, при редактировании в таком представлении могут быть потеряны некоторые параметры форматирования, не поддерживаемые в атрибутах типа RICHTEXT.

Если не указывать representation явно, то будет использовано значение auto, которое приводит к тому, что итем будет показан как текст, если при этом не произойдет потери информации, и как Html, если в итеме присутствует разметка не представимая в формате RICHTEXT. Определить как будет вести себя представление по умолчанию можно по значению атрибута FORMATTING, который равен 0, если итем представим в формате RICHTEXT и 0 в противном случае.

Начиная с версии 0.91 Communiware включает в себя WYSIWYG-редактор HTML, использующий стандартный Active-X из поставки Internet Explorer 5.5. Поэтому пользователи этого браузера могут редактировать текст в WYSIWYG представлении независимо от указанного representation.

Использование этого редактора имеет такую особенность: размер поля ввода нельзя задавать с помощью атрибутов rows и cols. Необходимо использовать

```
"style={width: xxx; height: yyy}"
```

иначе спецификация размера будет работать только при отключенном WYSIWYG.

Обычно по умолчанию WYSIWYG выключен а внизу поля ввода показывается ссылка на стандартную страницу с описанием процедуры установки Active-X.

В случае, если WYSIWYG включен, то показывается ссылка на страницу его отключения. Включение/отключение управляется настройкой пользователя (кукой) WYSIWYG.

Под полем ввода с representation=Text также показывается ссылка на страницу с описанием списка доступных выделений.

С помощью дополнительных параметров `wysiwyg=no` и `help=no` эти ссылки можно отключить.

Некоторые другие разновидности EditField тоже имеют параметры, которые не просто выводятся в результирующий HTML как атрибуты соответствующего тэга, а влияют на работу динамического элемента. Например, для атрибута PASSWD, которому соответствуют два поля ввода, можно задать фрагмент HTML-кода, разделяющий эти поля.

Использование динамического элемента Input имеет смысл только в том случае, если дизайнер хочет получить дополнительный контроль над полем ввода, например сформировать список значений не в виде выпадающего меню, а в виде набора радиокнопок.

Input заставлятет дизайнера руками делать то, что EditField делает сам, внося дополнительную возможность ошибок и потенциальной несовместимости с будущими версиями, поскольку если изменится способ взаимодействия формы с обработчиком постинга, EditField будет откорректирован соответствующим образом, а написанный вручную шаблон перестанет работать.

В формах постинга нельзя использовать HTML-тэг `<input>`, поскольку имена полей ввода, передаваемых браузеру, отличаются от имен

атрибутов, использованных в динамических элементах Input и EditField. Это происходит потому, что обработчик постинга должен иметь возможность отличить введенные пользователем атрибуты редактируемого итема от обычных параметров контекста, а также одноименные атрибуты двух итемов в одной форме множественного постинга — друг от друга.

Это достигается посредством приписывания ко всем именам элементов ввода префикса, так называемого *идентификатора постинга*, отделенного от имени точкой.

Поскольку JavaScript не позволяет использовать имена, содержащие точку в качестве идентификаторов объектов, для использования JavaScript в формах постинга необходимо явно приписывать параметр id элементам ввода.

3.3 Обработка постинга. Вычисления времени обработки

Ключевым моментом для понимания функционирования форм постинга, является представление о существовании контекста времени обработки результата постинга (или для краткости — времени сабмита).

Этот контекст, вернее, группа контекстов (при множественном постинге) состоит *только* из значений, отправленных браузером в результате сабмита формы.

В случае создания итем, те атрибуты, которые присутствуют в контексте времени постинга, записываются в этот итем, а остальные получают значения по умолчанию. При редактировании итем те атрибуты, которые не были заданы в форме постинга, сохраняют свои старые значения.

Связи при постинге рассматриваются как атрибуты, значением которых является неупорядоченное множество идентификаторов итемов.

Существует возможность определить один из атрибутов времени сабмита через другие, т.е. выполнить операцию, аналогичную Define, но не во время генерации страницы, а во время обработки постинга.

Для этой цели используется динамический элемент Use, который получает имя атрибута и выражение задающее его значение. Use почти полностью эквивалентен Define, включая параметры ifabsent и ifempty, но исключая возможность задания типа. Дело в том что в контексте постинга обычно имеют смысл только атрибуты, тип которых заранее

известен, т.е. атрибуты, которые будут записаны в итем или специальные атрибуты (см раздел 3.6), тип которых известен ядру заранее. Атрибуты, определяемые разработчиком, могут быть использованы только как аргументы для других вычислений, а механизм вычисления выражений в Communiware не использует информации о типах.

Выражение в Use может содержать как обычные подстановки (через символ @), которые будут выполнены при генерации страницы, содержащей форму, так и подстановки времени постинга (через символ %) которые будут выполнены в момент обработки постинга.

Такие же подстановки возможны и в других динамических элементах которые выполняют действия на этапе обработки постинга, в том числе и в параметре URL возврата самого динамического элемента Post.

Поскольку в момент обработки постинга известно только то, что было прислано из клиентского браузера (неизвестно даже имя шаблона, в котором содержалась форма), все динамические элементы, задающие действия на этапе сабмита, на самом деле просто генерируют в форме постинга скрытые поля со специальными именами, интерпретируемые потом обработчиком постинга.

Поскольку порядок, в котором браузер посылает поля в обработчик, не определен, все вычисления времени сабмита выполняются в „естественном" порядке. Т.е. если некоторый атрибут зависит от другого атрибута, то сначала вычисляется тот, от которого зависит первый, а потом этот первый.

Использование нескольких динамических атрибутов Use с одинаковым именем приводит к появлению атрибута со значением типа список, что очень удобно для привязывания связей.

Например, можно написать

```
<;Post "":>
<:Item new(ARTICLE) i1:>
....
<:EndItem:>
<:Item new(ARTICLE) i2:>
...
<:EndItem:>
<:Item new(ARTICLE) i3:>
<:Use SEEALSO %i1.ITEM_ID:>
<:Use SEEALSO %i2.ITEM_ID:>
...
```

```
<:EndItem:>
```

и получить у третьей статьи две связи SEEALSO.

Следует учитывать что вычисления времени самбита имеют приоритет над заданными в форме значениями. Поэтому использование Use для задания константных значений, которые должны быть использованы наряду с введенными, невозможно. В этих случаях следует пользоваться Input hidden, поскольку все Input-элементы равноправны.

Типичный пример такого использования - показ связанных итем в виде списка чекбоксов, который содержит по чекбоксу на каждый итем, который можно связать в данном контексте, и чекбоксы соответствующие реально существующим связям, помечены.

Очевидная конструкция вида

```
<:Loop Possible_Candidates:>
<:Input chekbox TOPIC @ITEM_ID @FLAG:>
<!-- мы считаем что фильтр возвращает атрибут FLAG,
равный 1 если связь есть и NULL если нет-->
<:EndLoop:>
```

приводит к очень странному эффекту - она позволяет удалять и добавлять связи до тех пор пока существует хотя бы одна связь.

Если же попытаться удалить все связи, то браузер просто не отправит Communiware-серверу параметра с именем TOPIC, и у сервера не будет никакой возможности определить, что в форме вообще присутствовало поле с таким именем. Поэтому связи удалены не будут, так как атрибуты которые явным образом не изменялись в форме, остаются неизменными.

Очевидное решение — добавить в список связанных итем пустую строку (которая будет проигнорирована, в случае если там присутствуют непустые элементы)

```
<:Input hidden TOPIC "":>
<:Loop Possible_Candidates:>
<:Input chekbox TOPIC @ITEM_ID @FLAG:>
<:EndLoop:>
```

Если же вместо `<:Input hidden:>` использовать

```
<:Use TOPIC "":>
```

```
<:Loop Possible_Candidates:>
<:Input chekbox TOPIC @ITEM_ID @FLAG:>
<:EndLoop:>
```

То при любом сабмите данной формы **все связи типа TOPIC будут удалены**. Это происходит потому, что значение вычисленное в момент постинга при помощи Use, имеет приоритет перед значением присланным из пользовательского браузера. Поэтому в данной конструкции атрибут TOPIC будет всегда пустым списком независимо от количества помеченых чекбоксов.

Кроме динамического элемента Use в форме постинга можно использовать динамический элемент Check, который позволяет выполнить проверку, что данные соответствуют определенным условиям. В случае если указанное в этом динамическом элементе условие выполняется, Check генерирует ошибку или предупреждение.

По окончании успешной обработки постинга (после создания итем, обработки MailTo, Subscribe и установки кук), контекст времени постинга удаляется, после чего производится действие, описанное в третьем аргументе Post.

В случае наличия предупреждения в контексте страницы показанной пользователю после сабмита формы, будет присутствовать атрибут WARNING, содержащий текст предупреждения. Если предупреждений было несколько, то они будут соединены через символ перевода строки.

В случае ошибки, контекст постинга сохраняется и используется для инициализации элементов ввода при повторном показе формы, а в контекст добавляется атрибут ERROR, содержащий текст сообщения об ошибки.

Следует учесть, что сабмит формы (даже в случае ошибки) эквивалентен переходу по гиперссылке, поэтому значения атрибутов контекста, явно указанные в URL или определенные с помощью Define теряются. Если необходимо их сохранить, то в форме постинга нужно использовать динамический элемент PassParams. Этот динамический элемент работает по-разному, в зависимости от того, употреблен он в форме постинга или вне ее. Вне формы он обеспечивает проброс атрибутов через URL, сгенерированные динамическими элементами ItemLink, Continuation и др.

В форме постинга он обеспечивает проброс атрибутов через сабмит данной формы.

3.4 Множественный постинг

Очень часто возникает необходимость одним действием создать или изменить несколько связанных итемов. Для этой цели в Communiware предусмотрены формы множественного постинга. В таких формах в динамическом элементе Post не указывается спецификация итема. Вместо этого все поля ввода помещаются внутрь контейнеров

`<:Item:> ...<:EndItem:>`

и спецификация указывается в качестве первого аргумента у динамического элемента Item,

В качестве второго аргумента ему может быть указано собственное имя данного контейнера, на которое можно ссылаться в подстановках времени постинга из соседних контейнеров. В качестве третьего (и последующих) аргументов динамическому элементу Item указывается логическое выражение. Если это выражение в момент постинга ложно, данный контейнер не обрабатывается.

В принципе возможен „множественный" постинг, состоящий ровно из одного контейнера Item, но он не отличается от одиночного практически ничем кроме возможности молча проигнорировать действие при невыполнении логического выражения.

В случае множественного постинга подставляеые во время постинга атрибуты в третьем параметре динамического элемента Post должны быть **обязательно** квалифицированы именем контейнера.

3.5 Формы с несколькими действиями

Формы постинга обычно содержат несколько кнопок, соответствующих разным действиям. Например, при редактировании итема, можно его сохранить или удалить. Обработчик постинга определяет, какая из кнопок была нажата, по наличию атрибута с именем совпадающим с именем кнопки и непустым значением.

Определены следующие действия:

SAVE Сохранить изменения.

DELETE Удалить итем.

SKIP не выполнять никаких действий. Перейти на страницу указанную в третьем параметре динамического элемента Post.

BACK Перейти на страницу, с которой мы попали на страницу с формой.

В случае множественного постинга, атрибут с одним из этих имен может либо присутствовать в каждом контейнере, что позволяет произвести разные действия с разными итемами одним нажатием кнопки, либо быть глобальным для всей формы, тогда соответствующий динамический элемент Input должен размещаться внутри Post, но вне Item.

В случае если атрибут с именем действия содержится как в контейнере, так и глобально, приоритет имеет локальный атрибут.

Поэтому конструкция вида

```
<:Post:>
<:Item @ITEM top:>
....
<:EndItem:>
<:Loop PassiveLinked(PART):>
<:Item @ITEM_ID:>
<:EditField TITLE:>
<:Input checkbox DELETE 1:>Пометьте для удаления
<:EndItem:>
<:EndLoop:>
<:Input submit SAVE Сохранить:>
<:EndPost:>
```

приведет к тому, что при нажатии кнопки «Сохранить» будут сохранены изменения в корневом итеме и тех частях, для которых не помечен чекбокс «пометьте для удаления». У этих контейнеров нет своего собственного действия, поэтому действует глобальное SAVE.

У тех контейнеров, у которых чекбокс помечен, существует локальное действие DELETE, и выполняется именно оно.

3.6 Специальные атрибуты времени сабмита

Если в обычном контексте у нас кроме текущих атрибутов есть и другие способы доступа к информации, например вызовы фильтров, и другие способы воздействия на Communiware, например явное употребление динамических элементов, то в контексте постинга у нас практически есть только атрибуты. Поэтому все управление процессом создания и

редактирования итемов производится путем задания некоторых значений атрибутам со специальными именами.

Данный подход имеет еще и то преимущество, что задавать значения атрибутам можно разными способами — ввести с помощью поля ввода (EditField или Input), вычислить с помощью Use, вычслить в момент генерации формы с помощью Define, а потом пробросить через Input hidden.

3.6.1 Работа со связями

В большинстве контекстов Communiware набор связаных item не рассматривается как атрибут. В контекстах постинга это не так. В этих контекстах необходимо иметь возможность редактировать связи (вернее, списки связанных итемов) в элементах ввода, которые оперируют с атрибутами, или выполнять над этими списками операции.

Если в динамическом элементе Input или EditField употребить в качестве имени редактируемого атрибута имя связи, или имя связи с суффиксом `_FOR` то значением по умолчанию для данного элемента ввода будет список идентификаторов итемов, которые в связаны с данным указанным типом связи. Просто имя связи обозначает те связи, в которых редактируемый итем является пассивным, а имя связи с суффиксом `_FOR` те, где он является активным. Так например статья в рубрике имеет тему (TOPIC) соответствующую этой рубики а итем рубрики является *темой для* (TOPIC_FOR) некоторого набора статей.

В зависимости от типа элемента ввода, список связанных итемов представляется либо как строка (Input text, EditField) где идентификаторы перечислены через запятую, либо (Input scrollinglist, Input checkboxgroup, Input popupmenu, Input radiogroup) как один или несколько выбраных элементов списка.

В тех случаях когда возможности динамического элемента Input, позволяющего задать произвольный фильтр в для получения списка итемов, которые могут быть связаны с данным, недостаточно, можно сформировать список, содержащий отдельные кнопки Input radio или Input checkbox с помощью произвольного фрагмента шаблона (например, с древовидной сортировкой).

Здесь следует учитывать, что если вы хотите позволить удалить все связи данного типа, то нужно предусмотреть вариант, который создаст параметр формы с данным именем и пустым значением, например <:Input hidden имя-связи :>, поскольку если связь с некоторым именем

задана только набором чекбоксов, и пользователь отменит отметку всех чекбоксов с этим именем, то обработчик постинга не сможет отличить эту ситуацию от формы, в которой вообще не упоминается данная связь. Поэтому текущее значение не будет изменено.

Использовать в данном случае динамический элемент Use нельзя, так как если в форме присутствует динаический элемент Use с некоторым именем, то значение введенное через Input или EditField будет проигнорировано, и использован только результат вычисления в Use.

Если в форме используется несколько динамических элементов Use с одинаковым именем, то результатом вычисления будет список всех вычисленных значений, что очень удобно для задания связей с несколькими итемами с помощью вычисляемых выражений.

3.6.2 Наследование связей

При создании нового итема механизм извлечения текущего состояния связи применить нельзя, поскольку итем еще не создан, и никаких связей не имеет. Но часто удобно задать некоторые связи в соответствии с тем, на странице какого итема была выведена форма постинга.

Например, если реплика дается в ответ на другую реплику, вероятно она будет относиться к той же нити, что и реплика на которую отвечали.

С другой стороны, помещая статью в рубрику, мы скорее всего хотим создать связь с этой же рубрикой.

Идентификатор того итема, в контексте которого была показана форма постинга нового итема, доступен в контексте постинга как атрибут `PARENT_ID`.

Поэтому для привязки статьи к рубрике можно написать

```
<:Use TOPIC %PARENT_ID:>
```

Для того, чтобы унаследовать связи, бывшие у предка, нужно создать в контексте постинга атрибут `INHERIT`, значением которого является список типов связей. Например,

```
<:Post new(REPLIC):>
<!-- автор - текущий пользователь-->
<:Use AUTHOR @AUTHOR_ID:>
<!-- Связана с текущей репликой связью PARENT-->
<:Use PARENT %PARENT_ID:>
<!-- наследует связь THREAD-->
```

```
<:Use INHERIT THREAD:>
<!-- можно поменять заголовок-->
<:EditField TITLE:><br>
<!-- и текст-->
<:EditField TEXT:>
<:Input submit SAVE "Ответить":>
<:EndPost:>
```

Наследование связи THREAD выполняется специальным образом — если заголовок (TITLE) создаваемого итема соответствует заголовку предка, (того, что задан в PARENT_ID), то прописывается связь THREAD с тем же итемом, на который показывает THREAD предка. Если же заголовок был изменен, то THREAD будет указывать на сам вновь созданый итем.

3.6.3 Работа с картинками

Картинки в Communiware обычно не считаются самостоятельной сущностью, а рассматриваются как часть текста итема (статья с иллюстрациями, шаблон с элементами графического оформления).

Поскольку формат HTML не позволяет хранить и редактировать картинки непосредственно в тексте, в форме постинга предусматриватется возможность загрузки картинок отдельно от текста.

Для этого нужно создать поле ввода с именем PICn, где n — цифра от 1 до 9. `<:EditField PICn:>` генерирует поле аплоада файлов с кнопкой «Обзор», через которое можно загрузить картинку. В этом поле при редактировании никогда не появится текущее значение, поскольку стандарт HTML запрещает инициализировать поля типа файл со стороны сервера по соображениям безопасности.

Кроме того, цифра от 1 до 9 означает не вообще n-ную картинку в итеме, а n-ную картинку, загружаемую в текущей форме.

Получить список существующих (уже загруженных) картинок можно с помощью фильтра `::Pic`.

По умолчанию картинка загружается с тем же именем, что она имела на локальном диске пользователя.

Если поля ввода картинок размещены в той же форме, что и поле ввода текста с использованием WYSIWYG-плагина, то в тулбаре плагина появляется кнопка «вставить картинку», позволяющая разместить

в тексте как картинки, уже загруженные в редактируемый итем, так и картинки выбранные для аплоада в данной форме.

В некоторых случаях, когда картинка должна показываться не внутри текста, а средствами шаблона, удобно присваивать картинке, загруженной через определенное поле фиксированное имя.

Этого можно добиться задав атрибут PICNAMEn, где n — число, соответствующее номеру поля ввода картинки.

Значением этого атрибута является имя файла. Вместо расширения можно указать звездочку, тогда будет использовано то же расширение, что у загруженного файла.

Внутри обработчика постинга атрибут PICNAMEn определен всегда, когда определен соответствующий PICn, даже если PICNAME не был явно задан в шаблоне, и содержит имя (с корректным расширением), под которым картинка будет сохранена на диск (с именем картинки могут быть произведены дополнительные преобразования — приведение к нижнему регистру, замена пробелов на подчерки и т.д.).

Кроме этого, в контексте постинга определены атрибуты PICWIDTHn и PICHEIGHTn, содержащие размеры картинки в пикселах.

Ими можно воспользоваться, например, для генерации фрагмента HTML-текста, показывающего картинку.

Для того чтобы удалить с сервера загруженную картинку, нужно создать в форме атрибут PIC_DEL, значением которого является список имен файлов, которые должны быть удалены. Его можно сформировать, например с помощью фильтра ::Pic и Input checkbox, предложив пользователю пометить картинки, которые больше не нужны.

3.6.4 Загрузка файлов

Communiware позволяет не только вводить текст итема в форме, но и загружать его из файла на локальном диске. Для этой цели служит атрибут UPLOAD, вернее конструкция <:EditField UPLOAD:>.

Она создает поле ввода имени файла, сопровождаемое кнопкой «Обзор», с помощью которого пользователь может загрузить файл со своего локального диска.

Для итемов с Content-Type HTML это может быть либо файл HTML, либо текстовый файл (он будет проинтерпретирован как RICHTEXT) либо файл RTF. В любом случае при сохранении он будет преобразован в HTML, с использованием значений переменных AUTOFIX (если это Html) или FORMAT (если это TEXT)

Для итемов с Content-Type BINARY это может быть произвольный файл. Он будет сохранен на сервере без изменений, но едиственная операция, которую можно будет с ним выполнить — скачать на локальную машину.

Для итемов с Content-Type CODE — текстовый файл.

Для итемов с Content-Type TABLE — файл в формате comma-separated values.

Если вы в форме редактирования предлагаете пользователю возможность загрузки файла, вы должны также предложить и возможность получения текста итем в виде файла для последующего локального редактирования.

Для этого следует поместить ссылку на скрипт `get_item`:

```
<:Script get_item "Скачать текст" ITEM_ID:>
```

3.6.5 Специальные атрибуты влияющие на удаление итема

При удалении итема иногда необходимо провести определенные действия с итемами, связанными с ним какой-либо связью, поскольку иначе они могут оказаться изолированными от основного графа и недоступными по ссылкам.

Возможны два варианта поведения в такой ситуации:

1. Удалить все поддерево (например, вместе со статьей удалить всю дискуссию по ней)
2. Переместить потомков под какой-то другой итем (например, при удалении рубрики переместить все её статьи в другую рубрику.

Первый вариант поведения задается с помощью атрибута CASCADE. Его значением должно быть имя связи или метасвязи (обычно, это метасвязь BELONGS). Если при удалении связи был задан этот атрибут, то все потомки по этой связи, не имеющие такой же связи с итемами, лежащими за пределами удаляемого поддерева, будут удалены.

Второй вариант задается атрибутом MOVE_CHILDREN. Его значением должен быть идентификатор итема того же типа, что и удаляемый. Если он указан, то все потомки, имеющие связь BELONGS длины 1 с удаляемым итемом, вместо нее получают связь с указаным итемом.

В форме постинга, имеющей и кнопку «сохранить», и кнопку «удалить» зачастую бывает необходимо при разных действиях выполнять

переход на разные страницы — например, при сохранении - на измененный итем, а при удалении куда-то в другое место (т.к. перейти на более несуществующий итем невозможно).

Решить эту задачу позволяет указание в форме постинга атрибута (посредством Use или Input hidden) параметра `RETURN_ON_DELETE`, значение которого содержит либо идентификатор итема (с необязательным шаблоном) либо полную URL.

3.6.6 Управление форматированием текста

Независимо от того, в каком представлении пользователь редактировал текст — HTML или RICHTEXT, существуют несколько вариантов того, каким образом обработчик постинга будет обрабатывать этот текст.

Communiware предъявляет существенно более строгие требования к синтаксической корректности HTML, чем статические Web-сайты. Дело в том, что введенный пользователем текст будет комбинироваться на странице с фрагментами шаблона и содержимым других итемов. Поэтому, например, незакрытый тэг <FONT> или <TABLE> в тексте реплики может привести к искажению, а то и полной нечитаемости всего остального содержимого страницы.

Поэтому Communiware предоставляет два варианта при постинге HTML — выдавать пользователю ошибку при любой синтаксической ошибке в HTML или автоматически корректировать эти ошибки. Управляет этим поведением атрибут AUTOFIX (до версии 0.912 — NO_LINT). Если он имеет непустое значение, Communiware будет автоматически исправлять ошибки, что может привести к искажению задуманного автором внешнего вида текста, но гарантирует его читаемость.

Если этот атрибут пуст или равен 0, то производится проверка синтаксиса, и в случае обнаружения ошибки операция отменяется полностью. Такой подход гарантирует то что в базу данных попадет именно то, что ввел пользователь, но требует от пользователя несколько более высокой квалификации.

Если пользователь вводит текст в формате RICHTEXT, то с помощью атрибута FORMAT можно задавать набор тэгов который будет разрешен для ввода. Значением возможные значения этого атрибута описаны в документации на модуль Communiware::Format::Text.

3.6.7 Защита от одновременного редактирования

При редактировании уже существующего итема может возникнуть следующая неприятная ситуация:

1. Пользователь A открыл форму для редактирования и получил старое содержимое итема
2. Пользователь B открыл ту же самую форму и получил то же самое старое содержимое.
3. Пользователь A нажал кнопку «Сохранить» и изменил содержимое итема
4. Пользователь B ничего об этом не зная, нажал кнопку «Сохранить» и сохранил свои изменения, уничтожив изменения пользователя A.

Чтобы избежать такой ситуации, включите в форму редактирования существующего итема <:Input hidden LASTCHANGE @LASTCHANGE:>

Атрибут LASTCHANGE является системным и изменить его посредством постинга нельзя. Но то значение которое будет получено в форме, соответствует моменту последнего изменения итема в том виде, в котором он был взять на редактирование.

Если в момент сохранения будет обнаружено, что итем в базе был изменен с тех пор, как пользователь поличил форму для редактирования, то ему будет выдано об этом сообщение, и он сможет предпринять какие-то разумные действия для разрешения конфликтной ситуации.

3.6.8 Пароль

Пароль пользователя в Communiware хранится в зашифрованном виде. Но в момент постинга, когда этот атрибут только что создан или изменен, он доступен в контексте. Вернее доступны две его копии с именами PASS1 и PASS2

3.7 Обработка ошибок

В случае, если при постинге возникла ошибка, отменяются все операции, производившиеся в данной форме, и пользователю вновь показывается та же форма с введенными им данными (кроме имен загруженных

файлов), независимо от того, какое действие по завершении постинга было указано в третьем параметре динамического элемента Post.

Текст сообщения об ошибке помещается при этом в атрибут контекста ERROR.

Разработчик шаблона постинга должен предусмотреть показ этого атрибута, с тем чтобы пользователь мог отличить ошибочное завершение от успешного, и понять и исправить причину ошибки.

3.8 Процедурные вставки. Динамический элемент Do

Процедурные вставки, это фрагменты кода на обычном процедурном языке программирования, как правило, производящие определенные манипуляции с контекстом до того как он будет использован стандартными обработчиками шаблонов и постингов, а также выполняющие некоторые действия при постинге, которые невозможно выполнить стандартными средствами Communiware, такие как разрезание большого загруженного HTML на части, преобразование картинок из одного графического формата в другой и др.

Процедурные вставки представляют собой перл-модули, заранее установленные на сервере его системным администратором. Возможности редактировать perl-код через web-интерфейс не предоставляется, в связи с потенциальной угрозой этих действий для безопасности сервера.

Существуют следующие моменты времени, в которые возможно выполнение процедурных вставок:

- В процессе визуализации страницы

  immediate В момент разбора шаблона и генерации HTML. Может быть использована для выполнения сложных вычислений, которые невозможно выполнить с помощью Subst и Define. Реально, с этой целью чаще используются

  postrequest По окончании обработки шаблона и отправки контента пользователю. Может быть использована, например для записи в базу информации о доступе к странице.

- В процессе постинга

prepost - выполняется до обработки всех контейнеров, т.е. вообще до начала транзакции, изменяющей базу. В случае ошибки на последующих стадиях постинга, действия, произведенные на этой стадии не откатываются.

preedit Выполняется в рамках транзакции перед тем как будет создан или изменен тот итем, в контейнере которого она указана. Поскольку порядок обработки контекстов постинга не определен, не гарантируется что до обработки этой вставки не произошли какие-либо изменения в базе, или даже откат транзакции. Т.е. если произошла ошибка в одном из предыдущих контейнеров в данной форме, эта процедурная вставка может быть вообще не выполнена.

postedit Выполняется в рамках транзакции после того, как произведена операция создания или обновления итема. Может, например создать несколько дополнительных итемов. Не выполняется, если произошла ошибка в текущем или одном из предыдущих контейнеров.

postpost Выполняется, если все контексты постинга в данной форме обработаны успешно. Выполняется после завершения (commit) транзакции. Это единственный вид процедурных вставок в постинге, из которых возможен запуск внешних по отношению к Communiware процессов, которые должны воспользоваться результатом только что совершенного постинга.

Например, именно на этой стадии производится переконфигурирование apache после создания нового сайта.

Процедурные вставки реализуются с помощью динамического элемента Do. Его синтаксис `<:Do` *`момент ИмяМодуля параметры`*`:>`. Где *момент* — один из вышеперечисленных, а *ИмяМодуля* — имя перлмодуля, к которому будет добавлено `Communiware::Postprocess`.

3.8.1 Постпроцессоры

Более старым способом выполнить действие при постинге является использование специального атрибута времени сабмита POSTPROCESS.

Начиная с версии 0.914 для этой цели используются процедурные вставки (раздел 3.8). В более ранних версиях единственной возможностью выполнить произвольный перловый код в момент обрабтки постин-

га были постпроцессоры (соответствующие процедурным вставкам периода postedit.
Выполнение постпроцессора задается с помощью атрибута POSTPROCESS.

Например,

```
<:Use POSTPROCESS
"Thumbnail %PICNAME1 50x50 thumbnail.*":>
```

Данное значение задает выполнение постпроцессора Thumbnail, т.е модуля Perl с именем Communiware::Postprocess::Thumbnail, которому передается в качестве параметров имя первой загруженной картинки (значение атрибута PICNAME1, и строки „50x50" и „thumnail.*".

В форме постинга может быть определено несколько постпроцессоров — иметься несколько динамических элементов Use или Input hidden, задающих атрибут POSTPROCESS. В этом случае выполнены будут все постпроцессоры, хотя порядок их выполнения не определен.

API постпроцессоров отличается от API процедурных вставок. См. документацию на модуль Communiware::Posting.

Глава 4

Справочник по синтаксису шаблонов

4.1 Параметры динамического элемента

Параметры динамического элемента разделяются между собой пробелами. Если требуется, чтобы параметр содержал внутри себя пробелы, то нужно заключить параметр в двойные кавычки — `"параметр с пробелами"`.

Если нужно включить в текст параметра двойную кавычку, то перед ней надо поставить обратный слэш — `"здесь \" кавычка"`.

В параметрах динамических элементов производится подстановка атрибутов. Вообще говоря, это основной способ получить в тексте генерируемого шаблоном HTML значения из базы данных. Синтаксис подстановки `@ИМЯ_АТРИБУТА`.

Следует учесть, что подстановка производится до того, как динамический элемент увидит свои параметры, но после того, как описание динамического элемента разбито на отдельные параметры. Поэтому динамический элемент не может определить, где внутри параметра начинается подстановка, и даже какого типа атрибут был подставлен, но наличие внутри значения атрибутов специальных символов, таких как `@`, кавычка или даже обратный слэш не мешает нормальному выполнению шаблона.

При подстановке именем атрибута считается последовательность из

букв, цифр и подчерков. Если необходимо, чтобы за значением атрибута следовал буквенно-цифровой символ, или, если спецификация атрибута содержит другие символы, например, скобки, имя атрибута следует заключить в фигурные скобки — `"@{CHILDREN(TOPIC)}"`.

4.1.1 Модификаторы подстановки

Значения атрибута могут быть, вообще говоря, произвольными, т.е. содержать введенный пользователем текст или HTML-ную разметку. Поскольку далеко не во всех контекстах допустимо наличие прозвольных символов, существуют модификаторы подстановки.

@^ИМЯ Переводит все буквы в значении атрибута в заглавные.

@_ИМЯ Переводит все буквы в значении атрибута в маленькие.

@%ИМЯ Выполняет URL-escaping всех символов, недопустимых в URL. Используется для ручной генерации URL. (Динамические элементы, специально предназначенные для генерации гиперссылок, выполняют URL-escaping сами)

@&ИМЯ Применяется для вставки значения в HTML в контексте, в котором недопустимы никакие тэги, например для атрибута alt= у тэга img.

@*ИМЯ Выполняет преобразование значения атрибута типа RICHTEXT в редактируемый текстовый формат. Применяется при программной генерации значений в формах постинга.

Символ `@` тоже может быть защищен обратным слэшом, если требуется чтобы он был передан в динамический элемент в неизменном виде.

В формах постинга, кроме подстановки через `@` также применяется подстановка через `%` которую мы подробнее рассмотрим в разделе 4.4.

В параметрах групповых атрибутов и функций контекста производится второй раунд подстановки, т.е. конструкция вида `@{CHILDREN(@MY_LINK)}` будет проинтерпретирована как „количество потомков по связи, имя которой хранится в атрибуте MY_LINK”. Возможности второго раунда подстановки ограничены тем, что в нем нельзя использовать групповые атрибуты и функции контекста — парсер параметров динамических элементов не обрабатывает вложенные скобки.

4.1.2 Классификация параметров

Параметры динамических элементов могут по разному интерпретироваться динамическими элементами. Рассмотрим основные типы параметров

HTML-текст Текст, который будет подставлен в результирующую страницу так, как написан (с учетом подстановки).

Ссылка на item или шаблон Используется в основном для генерации URL, хотя в некоторых динамических элементах указанный шаблон фрагмента будет испольъзован незамедлительно. В параметрах такого типа, кроме явного указания идентификатора итем, можно использовать конструкцию `*имя_виртуального_итем` которая будет разыменована в контексте текущего сайта.

Фильтр Фильтр это способ сформировать список итем, который потом будет тем или иным способом показан, или использован с какой-то другой целью, например на предмет проверки вхождения в него заданного идентификатора. Синтаксис фильтров подробно рассматривается в разделе 6.5. В параметрах фильтра также производится разыменование *виртуальных итем*

Имя атрибута В некоторых динамических элементах значение параметра интерпретируется как *имя* атрибута, т.е. подстановка значения производится не при разборе параметров, а непосредственно самим динамическим элементом. Обычно, это делается в тех случаях, когда динамическому элементу важно знать тип атрибута. Побочным эффектом является то, что в такой параметр можно подставить имя атрибута из контекста, и таким образом, сгенерировать, например, список всех атрибутов текущего итем в с помощью фильтра.

Параметры со специальным синтаксисом В некоторых динамических элементах испоользуются параметры со специальным синтаксисом, например список пар `имя=значение` через запятую. В таких параметрах заключать в кавычки каждое значение не нужно, в кавычки должен быть (при наличии внутри пробелов) заключен весь параметр в целом. Если требуется использовать запятую, как часть значения, то ее следует защитить при помощи обратного слэша.

4.2 Арифметические выражения

Арифметические выражения в Communiware допустимы в очень ограниченном наборе динамических элементов — Subst (с. 91), Use (с. 120), Header (с. 94) и Define (с. 102).

Во всех этих динамических элементах допустимо переменное число параметров. Каждый операнд и знак операции (включая скобки) должен передаваться в динамический элемент как отдельный параметр.

В арифметических выражениях все операнды считаются константными. Значения из базы данных в них подставляются через обычный механизм подстановки.

В арифметических выражениях допустимы следующие операции:

+ Арифметическое сложение. Операнды операции сложения могут быть либо числами, либо датой и числом. При сложении даты с числом число интерпретируется как количество суток, которые надо прибавить к дате. Если необходимо прибавить меньший интервал времени, чем сутки, используйте дробное число.

- Арифметическое вычитание. Операндами могут быть два числа, дата и число, две даты. В последнем случае результатом будет (дробное) число суток, разделяющих эти две даты.

. Операция строковой конкатенации. Объединяет две строки, являющиеся ее операндами. Если один из операндов — число или дата, он преаобразуется в строковое представление по умолчанию (см. раздел 6.3.5).

x Строковое повторение. Первый операнд — строка, второй — целое число n. Результат — строка повторенная n раз.

% Операция форматирования. Первый операнд — значение определенного типа, второй — строка спецификации формата. Если значение числовое, допустим любой формат, понимаемый функцией `printf`, если дата — понимаемый функцией `strftime`. Возможные значения форматов для атрибутов типа RICHTEXT приведены в разделе 6.3.5.

***** Операция умножения. Допустима только над числами.

/ Операция деления. Допустима только над числами. Второй операнд не может быть нулем. Результат — вещественное число.

div Целочисленное деление.

mod Остаток от целочисленного деления.

4.3 Логические выражения (условия)

Логические выражения в Communiware применяются в динамических элементах If (с. 97), Cond (с. 98) Check (с. 121) и Item (с. 118).

Логические выражения состоят из *условий*, которые состоят из двух *аргументов*, разделенных знаком *операций отношения*. Кроме того, иногда встречаются условия состоящие из одного элемента — проверка значения на пустоту или проверка существования связи.

Условия могут быть скомбинированы с помощью знаков *логических операций*. Логических операций существует три — *отрицание*(`!`), *дизъюнкция*(`||`) и *конюнкция*(`&&`).

Приоритет подчиняется обычным правилам. Для управления приоритетом могут быть использованы круглые скобки (так же как и в арифметических выражениях, все знаки операций и скобки должны передаваться как отдельные параметры.

Например,

```
<:If A = B:>
<:If ! A IN filter:>
```

В динамических элементах Cond, Check и Item оба операнда логической операции — константы. В динамическом элементе If первый операнд первого отношения (т.е. первый параметр динамического элемента, или второй, после знака отрицания) указывается как имя атрибута, что позволяет сравнивать на неравенство атрибуты нестандартных типов, такие как статус.

Частным случаем логического выражения является просто указание имени атрибута (или константы в Check и Item). Такое логическое выражение истино тогда, когда атрибут имеет непустое значение, не равное 0.

Это выражение также может быть предварено знаком отрицания.

В динамическом элементе If допустимы также логические выражения вида `ИМЯ_СВЯЗИ`, которые истины тогда, когда существует указанная связь между текущим пользователем и текущим итем. Кроме этого

существует специальное условие ACCESS, которое истино тогда, когда текущий пользователь имеет право чтения текущего итем (текущий итем в процессе выполнения шаблона далеко не всегда тот, который был указан в URL. См. главу 5).

В Communiware поддерживаются следующие логические операции:

=, != Сравнение на равенство и неравенство. Определены для всех типов данных.

<,>,<=,>= сравнение на больше/меньше. Имеют смысл для чисел и дат, а в случае динамического элемента If, еще и для статуса итем. Сравнение статусов производится по порядковым номерам статусов, присвоенных им при создании типа.

~ Сопоставление с регулярным выражением.

IN В качестве второго операнда получает спецификацию фильтра. Истино, если первый операнд равен ITEM_ID одного из итемов, возвращенных фильтром.

Если необходимо сравнить значение не с ITEM_ID, а с каким-то другими атрибутом из числа возвращаемых фильтром, следует указать имя этого атрибута через двоеточие после значения.

:=: Сравнение множеств. Оба операнда этой операции — списки значений через запятую. Они рассматриваются как неупорядоченные множества, причем повторяющиеся и пустые элементы игнорируются. Истина тогда, когда оба списка содержат одинаковый набор значений.

4.4 Подстановка в постингах

Формы постинга, в отличие от всех остальных элементов шаблона, обрабатываются в две стадии — сначала генерируется и показывается пользователю HTML, содержащий саму форму, потом, после нажатия кнопки submit, обрабатываются результаты пользовательского ввода.

На этом втором этапе часто бывает осмыслено также выполнять различные проверки и подстановки, например создать одновременно два итема, и прописать одному из них связь со вторым, т.е. использовать в качестве значения связи значение, введенное как ITEM_ID другого итем в той же форме. Для этой цели применяется механизм подстановки

сходный с механизмом подстановки при генерации HTML, только в качестве специального символа-знака подстановки применяется символ %.

На этапе генерации формы конструкция вида `%ИМЯ_АТРИБУТА` трактуется как обычные константные данные, а попав в обработчик результатов постинга, разыменовывается. Обычные подстановки через `@` разыменовываются на этапе генерации формы, следовательно воспользоваться в них результатами пользовательского ввода нельзя.

В случае если в форме одновременно редактируется/создается несколько итемов, можно присвоить каждому из них условное имя (см. раздел 7.8.2) и ссылаться на его атрибуты как `%имя.АТРИБУТ`.

Глава 5

Контексты Communiware

Как вы успели заметить из предыдушей главы, в языке шаблонов Communiware постоянно упоминается понятие атрибута. Атрибут это, как правило, какое-то свойство итема. Поскольку каждая страница, сгенерированная шаблоном Communiware, как правило несет информацию о нескольких итем, возникает вопрос, каким образом при ссылке на атрибут мы определяем, об атрибуте какого итема идет речь. Ответом на этот вопрос является понятие контекста. Контекст это вообще говоря набор атрибутов, доступных в текущей точке шаблона.

5.1 Контекст итема

Основу большинства контекстов Communiware составляет набор атрибутов некоторого итема. Если в контексте присутствует ITEM_ID, то можно считать, что в нем присутствуют и все остальные атрибуты этого итем.

Т.е. к ним можно обращаться, не задумываясь о том, каким образом они попали в текущий контекст, например были ли они извлечены фильтром, если речь идет о контексте элемента списка. Располагая идентификатором итема, Communiware может извлечь и все остальные его атрибуты.

Кроме атрибутов итема в контексте доступны *вычислимые атрибуты* (см раздел 6.2.1), представляющие собой результат каких-либо вычислений над атрибутами итема, *групповые атрибуты* (см. раздел

6.3.1), характеризующие не сам итем, а группу связанных с ним итемов (например, количество статей в рубрике, дату публикации последней реплики в дискуссии) и *функции контекста* (см. раздел 6.3.2).

В контексте, созданном с помощью динамического элемента List или Loop с SQL или Perl-фильтром, кроме этого могут присутствовать атрибуты созданные фильтром. Все поля, возвращаемые SQL-запросом фильтра (т.е. явно указанные в предложении SELECT) доступны в контексте элемента списка как атрибуты. К сожалению, Communiware не имеет информации о типах этих атрибутов, поэтому, если необходимо интерпретировать их не как строки, а как даты, RICHTEXT или числа, следует воспользоваться *суффиксной нотацией*.

Атрибуты, имена которых заканчиваются на _N и тип явно не указан, рассматриваются как числовые, соответственно суффиксы _D и _R являются признаками даты или RICHTEXT.

При использовании perl-фильтров, разработчик фильтра имеет возможность описать тип атрибутов, возвращаемых его фильтром.

Кроме того, динамические элементы List и Loop, помещают в контекст ряд специальных атрибутов, описанных в разделе 7.6.1.

5.2 Глобальный контекст

Кроме контекста итема, который может меняться несколько раз на протяжении страницы, существует глобальный контекст, т.е. набор атрибутов которые сохраняют свои значения на протяюении всей страницы, независимо от того, какой итем отображается в данной ее точке.

5.2.1 Имя и настройки пользователя

Кроме атрибутов итем, в контексте доступна и другая информация. Например, если страница генерируется для зарегистрированного пользователя Communiware, то в контексте доступен его идентификатор, в виде значения атрибута AUTHOR_ID. Для анонимного пользователя этот атрибут имеет пустое значение. Для того чтобы облегчить задачу индикации текущего пользователя, кроме AUTHOR_ID существует атрибут USER, равный AUTHOR_ID пользователя, если он зарегистрирован и слову АНОНИМ, если нет.

Система настроек Communiware предназначенная для публичных сайтов, позволяет настраивать ряд параметров даже и анонимным поль-

зователям. Эти настройки хранятся в cookies. Поэтому все присланные пользовательским браузером cookies доступны как атрибуты контекста.

Стандартные коммуниверные настройки, кроме этого, описаны в базе данных как атрибуты вида COOKIE, что позволяет получать информацию об их типе. Если пользовательский браузер не предоставил значений этих стандартных cookies, то эти атрибуты будут иметь значение по умолчанию, заданное в описании этих атрибутов.

Если страница генерируется как ответ на заполнение формы ввода (за исключением форм постинга), то все значения, введенные пользователем как поля формы, тоже доступны как атрибуты контекста. Это позволяет легко реализовывать настройку параметров страницы или различные виды поиска.

5.2.2 Атрибуты, созданные разработчиком

Кроме того, новые атрибуты могут быть определены непосредственно в шаблоне, с помощью динамического элемента Define (с. 7.5.1).

При создании таких атрибутов можно явно задать их тип.

Кроме того, значения явно переданные в HTTP запросе, тоже рассматриваются как атрибуты контекста. Такие атрибуты могут быть либо результатом сабмита формы, либо явно указаны в URL при формировании ее с помощью динамического элемента `<:ItemLink:>`(см. раздел 7.2.1). С этими атрибутами также не связана информация о типах, поэтому может потребоваться использование суффиксной нотации.

Атрибуты контекста страницы, на которой расположена ссылка, могут быть переданы в глобальный контекст страницы, на которую ведет ссылка как явным указанием их в динамическом элементе `<:ItemLink:>` (в этом случае указывается только имя атрибута, но не значение), так с помощью динамического элемента `<:PassParams:>` (см. раздел 7.5.2) который указывает список атрибутов, передаваемых через **все ссылки на текущей странице.**

Правда, в этом случае, можно воспользоваться динамическим элементом Define с опцией ifabsent для задания значения по умолчанию и типа.

5.2.3 Магические атрибуты

Кроме того, существуют несколько «магических» атрибутов, создаваемых ядром Communiware.

К их числу относятся:

URL_PREFIX То что должно идти в URL страниц данного Communiware-сайта до идентификатора итема.

SCRIPT_PREFIX То что должно идти в URL интерфейсных скриптов Communiware до имени скрипта

PIC_PREFIX То, что должно идти в URL картинок и бинарных файлов до идентификатора итем.

USER_STATUS Определен, если текущий пользователь — зарегистрированный. Содержит его атрибут STATUS.

USER_AGENT Строка идентификации пользовательского браузера.

BROWSER_NAME Результат анализа ядром названия пользовательского браузера. Обычно "Mozilla"или "MSIE", реже "Opera"или "Lynx".

BROWSER_VERSION Номер версии браузера как вещественное число, если его оказалось возможным определить в результате анализа строки USER_AGENT.

OS_VERSION Для MSIE — версия Windows под которой он работает (в этом случае знание разновидности ОС важно для корректной настройки WYSIWYG плагина). В случае других пользовательских агентов и, тем более, других ОС может быть не определен.

ACCEPT_TYPE Значение HTTP-заголовка Accept-Type присланного браузером, в виде списка значений MIME-типов, который может быть использован как фильтр, или в операции сравнения множеств.

CURRENT_SERVER Идентификатор Communiware-сайта, для которого генерируется страница. Может не совпадать с атрибутом SERVER текущего итем.

CURRENT_TEMPLATE Идентификатор шаблона страницы, обработка которого производится. Заметим, что атрибут итема TEMPLATE_ID содержит идентификатор шаблона по умолчанию для данного итем.

REMOTE_ADDR IP адрес клиента. В отличие от параметра CGI-environment REMODE_ADDR, учитывает возможность наличия фронтэнда, и указывает первый IP-адрес за фронтэндом (а в CGI-переменной в конфигурации с фронтэндом всегда будет 127.0.0.1).

FORWARDED_FOR Список адресов прокси, следущих после REMOTE_ADDR, если они таковые сообщили.

RAND Случайное вещественное число от 0 до 1. См также функцию контекста RAND(n).

5.2.4 Особенности глобального контекста почтовых рассылок

При обработке шаблонов, генерирующих письма электронной почты, отправляемые с помощью системы подписки или с помощью динамического элемента MailTo, имеется ряд существенных отличий от контекста web-страницы.

Во-первых, в этих контекстах, очевидно, недоступна информация, связанная с типом браузера и IP-адресом адресата. Во-вторых, значение атрибута `AUTHOR_ID` не определено в шаблонах рассылки (кроме персонализованной), а в шаблонах, обрабатываемых динамическим элементом `MailTo` соответствует идентификатору *отправителя*, а не *получателя* письма.

Кроме того, при обработке рассылки в глобальном контексте присутствуют атрибуты `PERIOD_START` и `PERIOD_END` типа дата, которые соответствуют началу и концу периода времени, обновления за который рассылаются.

5.3 Контексты, не содержащие итем

В некоторых случаях в контексте может не содержаться ITEM_ID. Таковы например, контексты форм постинга нового итем, в которых ITEM_ID создаваемого итема недоступен просто потому, что он еще не присвоен.

Кроме того, возможны фильтры, которые не возвращают ITEM_ID, например, фильтр перебирающий статусы данного типа итем, или

картинки в данном итем. В контекстах, созданных с помощью таких фильтров доступны только атрибуты, непосредственно созданны фильтром.

5.4 Стэк контекстов

Первый контекст итема при обработке шаблона возникает еще до начала собственно обработки, при анализе URL коммуниверной страницы, которая всегда содержит ITEM_ID[1]

Страница Communiware как правило информацию о нескольких итемах. Например, страница рубрики содержит список статей, в каждом элементе которого текущим итемом является соответствующая статья. А описание каждой статьи содержит список авторов, которые также являются итемами.

Таким образом, основным (практически — единственным) способом получить контекст другого итем, является формирование списка, содержащего этот итем.

Контексты, сформированные списками, называются *вложенными контекстами*, поскольку по завершении обработки списка, происходит возвращение в исходный контекст. В свою очередь внутри обработки элемента списка можно создать еще один список (например, при обработке элемента списка статей — список авторов статьи), и получить контекст следующего уровня вложенности. Так образуется *стэк контекстов*

Вышележащие контексты при этом продолжают существовать (ведь нам предстоит в них вернуться), и к их атрибутам возможно обращение.

Для того, чтобы обратиться к атрибуту вышележащего контекста, нужно указать в конце имени атрибута #n, где n — число ступенек вверх по стэку, на которые нужно подняться. Например, `ITEM_ID#1` — ссылка на идентификатор итем уровнем выше.

Атрибуты контекста, которые не зависят от итем, такие как имя текущего пользователя, доступны во всех вложенных контекстах. Атрибуты, созданные с помощью динамического элемента Define, обладают тем же свойством.

[1] ссылка на корень сайта не содержит явно указанного ITEM_ID, но интерпретируется как ссылка на итем сайта

5.5 Контексты постинга

Под контекстами постинга мы понимаем в первую очередь контексты, которые возникают в обработчике результатов постинга.

Контексты форм постинга (фрагмента страницы, генерируемого с помощью содержимого динамического элемента Post) также обладают рядом специальных свойств.

5.5.1 Контекст формы постинга

Контекст формы редактирования практически не отличается от обычного контекста в любом другом месте шаблона. Единственным отличием является то, что в этом контексте присутсвтуют списки связанных итемов, а при редактировании шаблона, еще и список типов итем, для которых этот шаблон может быть использован в качестве стандартного (в виде атрибута SUBS_ITEM_TYPE).

Гораздо больше отличий у контекста формы создания нового итема. Эти отличия вызываны тем, что в момент генерации страницы итема еще не существует, поэтому его атрибуты не могут быть известны. Единствнным атрибутом вновь создаваемого итема, который доступен в контексте генерации формы создания, является TYPE_ID, так как он был задан в спецификации постинга.

5.5.2 Контекст обработки постинга

Контекст постинга состоит практически исключительно из тех значений атрибутов, которые были введены через форму ввода. В этих контекстах может быть ряд специальных атрибутов, которые позволяют управлять созданием/изменением итем.

Как правило, эти специальные атрибуты также создаются с помощью динамических элементов Input и Use.

В случае если в форме постинга присутствует несколько динамических элементов Input с одинаковым именем, полученный в результате атрибут контекста постинга представляет собой список их значений.

То же самое касается нескольких элементов Use. Но если в форме присутствуют и динамический элемент Input, и динамический элемент Use с одинаковыми именами, то значение, введенное с помощью Input будет проигнорировано и использовано только значе-

ние, вычисленное с помощью Use. Поэтому, если необходимо позволить менять часть значений, скажем списка связанных итемов, и сохранить неизменными остальные, то для передачи неизменных элементов списка следует использовать Input hidden, а не Use.

Еще одной особеностью контекста обработки постинга является то, что контексты всех контейнеров Item внутри одного динамического элемента Post существуют одновременно. Поэтому внутри арифметических выражений в динамических элементах Use и Check можно ссылаться на атрибуты „соседних" контекстов, используя нотацию *имя контейнера.АТРИБУТ*, **где** *имя контейнера* **— имя, указанное во втором параметре динамического элемента Item.**

Контейнеры, для которых не указано имя, получают некоторые имена по умолчанию, которые могут непредсказуемым образом изменяться при внесении изменений в шаблон. Поэтому ссылаться на атрибуты контейнера, которому явным образом не было задано имя, нельзя.

5.5.3 Специальные атрибуты контекста постинга

Задание связей с другими итемами

В форме постинга могут быть поля ввода с именами соответствующими именам типов связи. Их значениями являются списки идентификаторов итем.

В форме эти поля могут быть описаны либо как длинные текстовые поля, куда пишутся идентификаторы через запятую, либо как набор полей с одинаковым именем, (например checkbox) каждое из которых содержит один идентификатор.

Эти поля ответственны за связи, в которых редактируемый итем является пассивным.

Как в контексте формы постинга (внутри динамического элемента Post или Item), так и в контексте обработки результата формы, атрибуты с именем соотвествующим имени связи рассматриваются как имеющие значение списка идентификаторов итемов, связанных с данным указанной связью, причем данный итем является в этой связи пассивным, а перечисленные — активными.

Если мы хотим отредактировать связи, в которых данный итем является активным, к имени связи надо добавить суффикс `_FOR`, **например,** `TOPIC_FOR` **задает список итем,** *для* **которых данный**

итем является рубрикой.

Управление наследованием атрибутов

В форме создания нового итема автоматически появляется атрибут PARENT_ID, равный идентификатору того итем, в контексте которого была создана форма постинга.

Если указать атрибут INHERIT, значением которого является список типов связи, то эти связи будут унаследован.

В-третьих, с помощью атрибута POSTPROCESS можно указать модуль постпроцессора, который будет вызван в случае успешного создания/обновления итем. Значение этого атрибута имеет вид:

```
"ИмяМодуля параметры"
```

Например, конструкция

```
<:Use POSTPROCESS "Keywords":>
```

приведет к вызову постпроцессора Communiware::Postprocess::Keywords.

Использование атрибута **POSTPROCESS** является устаревшей конструкцией, описанной здесь только для облегчения понимания шаблонов существующих Communiware-сайтов. Во вновь разрабатываемых шаблонах используйте процедурные вставки (см раздел 7.5.3)

Загрузка основного содержимого

В контексте постинга может быть атрибут TEXT (обычно соответствующий многострочному полю ввода) или UPLOAD (поле типа файл), позволяющий загрузить основное содержимое данного итема.

В случае, если в постинге присутствуют оба атрибута одновременно, **UPLOAD** имеет приоритет над **TEXT**. Считается что **TEXT** может быть старым содержимым итема, которое было отгружено пользователю в качестве значения поля редактирования, а **UPLOAD** — обязательно результат явных действий пользователя.

На обработку Communiware значения этого атрибута влияют атрибуты AUTOFIX и FORMAT. Первый из них отключает синтаксическую проверку в случае, если загружаемый файл имеет формат HTML, и заменяет ее на автоматическую нормализацию. Второй —

содержит спецификацию формата, влияющую на преобразование текста с выделениями в HTML. и управляет способом форматирования текста, который был введен пользователем как текст с выделениям.

Управление иллюстрациями

***В-пятых* в форме ввода могут быть поля, управляющие картинками, загруженными в item.**

PICn где n — цифра от 1 до 9. Соответствует полю ввода типа file, через которое производится загрузка картинки.

PICNAMEn Обеспечивает возможность переименования картинки в момент аплоада на сервер. В это поле можно ввести конструкцию вида `имя.`, тогда будет подставлено то же расширение, что и у загруженного файла.

Если необходимо сослаться на имя загружаемой картинки, внутри формы постинга (например, при генерации текста итема с помощью Use, или при вызове постпроцессора), *всегда* следует использовать атрибут PICNAMEn, а не PICn, даже если в форме ввода такого поля не было. В момент обработки PICNAMEn содержит полное имя картинки, под которым она сохраняется на сервере.

PICWIDTHn, PICHEIGHTn размеры картинки в пикселах по горизонтали и вертикали. Эти поля никогда не прописываются явным образом в форме, они вычисляются автоматически, если была загружена картинка с соответствующим номером.

PIC_DEL Значение этого поля — список имен файлов картинок, которые уже существуют в итем. Если это поле не пустое, перечисленные в нем картинки удаляются.

Управление поведением формы

***В-шестых*, в форме постинга должно обязательно присутствовать поле, определяющее действие, которое следует выполнить с данным итем.**

Возможны следующие действия:

SAVE сохранить изменения

DELETE удалить весь итем

SKIP проигнорировать данный итем (возможно обработав другие в той же форме)

BACK не выполнять никакой обработки, вернуться на предыдущую страницу по отношению к той, на которой была изображена форма.

В отличие от остальных атрибутов постинга, которые обязаны быть привязаны к конкретному итем (помещены внутрь контейнера Item, если в форме постинга несколько таких контейнеров), действия могут быть глобальными для формы. Поиск действия относящегося к данному контейнеру выполняется сначала внутри него, потом среди глобальных (находящихся за пределами любых контейнеров) полей ввода.

Это позволяет создать множественный постинг, в котором большая часть контейнеров создает итемы, или изменяет существующие, а некоторые — удаляют.

Кроме того, в форме может присутсвовать атрибут `RETURN_ON_DELETE`**, который указывает альтернативную URL, на которую следует перейти после обработки шаблона, если текущим действием было удаление итема. Это имеет смысл в формах, в которых присутствует и кнопка SAVE, и кнопка DELETE, и в результате нажатия кнопки SAVE мы попадаем на страницу редактируемого итема. В этом случае после удаления этого итема мы попадем на несуществующий итем, что вызовет ошибку HTTP 404.**

Детектирование одновременного редактирования

Особенностью редактирования через web-интерфейс является то, что с момента получения пользователем формы для редактирования и до момента нажатия кнопки SAVE никакого взаимодействия между пользовательским браузером и сервером не происходит. Мы не можем знать, продолжает ли пользователь редактирование, или выключил компьютер и пошел спать.

Поэтому у нас нет возможности заблокировать редактируемые итемы от изменения другими пользователями.

В результате, если пользователь А получил форму для редактирования, потом форму для редактирования получил пользователь Б, потом пользователь А сохранил свои изменения, потом изменения сохранил пользователь Б, все изменения, сделанные А будут потеряны, так как в базу попадет та информация, которую получил Б (до того, как А сохранил свои изменения) плюс те изменения которые внес Б.

Некоторую защиту от подобных неприятностей предоставляет специальная обработка атрибута LASTCHANGE формами постинга.

Атрибут LASTCHANGE хранит время последнего изменения итема. Он устанавливается автоматически при каждом изменении итема, включая операции дампа и восстановления или непосредственную работу с базой данных минуя интерфейсы Communiware. Поэтому нет ни малейшего смысла предоставлять пользователю возможность редактирования данного атрибута.

Но в случае если в контексте постинга присутствует атрибут LASTCHANGE (созданный с помощью Input hidden LASTCHANGE @LASTCHANGE) считается, что он соответствует дате изменения итема на тот момент, когда пользователь получил форму для редактирования. Поэтому, если в момент сохранения изменений LASTCHANGE данного итема имеет другое значение, мы можем утверждать, что итем был изменен кем-то еще.

В этом случае, пользователю выдается сообщение об ошибке и ему предоставляется самостоятельно решить, что каким образом совместить изменения, сделанные им и изменения, сделанные другим пользователем.

5.5.4 Обработка пустых и отсутстсвующих значений

Следует помнить о том, что при редактировании итем атрибуты, отсутствующие в форме, не изменяются. Здесь проводится четкая разница между атрибутом с пустым значением, который очищает атрибут итем, и отсутствующим атрибутом, приводящим к сохранению старого значения.

Особенно частым источником ошибок является задание каких-либо связей в виде набора чекбоксов. Дело в том что если все чекбоксы с определенным именем не отмечены, браузер вообще не посылает на сервер поля с таким именем. Поэтому если необхо-

димо предоставить возможность отвязать все связи в таком интерфейсе, следует добавить скрытое поле с тем же именем и пустым значением.

5.6 Объектный интерфейс контекстов

В процедурных вставках контекст представляется в виде объекта `Communiware::Context::Object`, имеющего методы `get(NAME)`, и `put(NAME,$value)`, позволяющие манипулировать контекстом. В конексты постинга имеют также метод `action()`.

Глава 6

Объекты, используемые в языке шаблонов

6.1 Динамические элементы

Динамические элементы — основной объект, отличающий шаблон Communiware от статического HTML-файла. Все остальные объекты, рассмотренные ниже могут появляться в шаблоне только внутри параметров динамических элементов.

Динамический элемент имеет вид

<:*Имя параметры*:>

Список динамических элементов входящих в стандартный комплект Communiware приведен в главе 7.

Обработка параметров рассмотрена в разделе 4.1.

Каждый динамический элемент реализован в виде отдельного perl-модуля, имеющего стандартизованный интерфейс. Некоторые коммуниверные пакеты могут определять свои собственные динамические элементы.

6.2 Атрибуты и их типы

6.2.1 Стандартные и расширенные атрибуты

Атрибуты итема, хранящиеся в базе данных, делятся на стандартные и расширенные. Стандартные атрибуты есть у всех итемов, расширенные — специфичны для конкретного типа итем.

Из набора стандартных атрибутов следует выделить `TYPE_ID`**, определяющий тип итем. Возможность его изменения ограничена только теми типами, которые имеют тот же набор расширеных атрибутов, что и текущий.**

Другим выделеным атрибутом можно считать `STATUS`**. Он отличается тем, что набор его значений специфичен для данного типа итем, а также тем, что имеется специальный атрибут** `STATUS_UPDATE`**, хранящий дату последнего изменения статуса. Кроме того, со статусом связано понятие *workflow*. В общем тип итема следует рассматривать как свойство итема, неизменное на протяжении всей его жизни на сайте, в то время как статус — свойство которое специально предназначено для того, чтобы меняться, отражая различные фазы жизненного цикла итема.**

Понятие workflow непосредственного отражения в языке шаблонов не имеет. Это ограничения, накладываемые на то, какими правами должен обладать пользователь, чтобы перевести итем из данного состояния в другое.

6.3 Вычислимые атрибуты

Вычислимыми атрибутами называются атрибуты, которые нельзя отредактировать непосредственно через форму постинга или интерфейс модератора, но значения которых изменяются автоматически как результат изменения других атрибутов того же итем. Сюда относятся, например, LASTCHANGE, хранящий дату последнего изменения итем, или TEXTSIZE, хранящий текст.

Кроме того, можно определять свои собственные вычислимые атрибуты, которые должны быть результатом SQL запроса с единственным параметром, задающим ID итем. Таким способом, например подсчитывается статистика обращений к итем.

6.3.1 Групповые атрибуты

Атрибуты, значение которых зависит нет только от состояния текущего итем, но и от состояния других, как-то с ним связанных, называются статистическими атрибутами, или атрибутами с параметрами, поскольку записываются в виде `ИМЯ(параметры)` **где параметры позволяют указать, какие именно из связанных итем нас интересуют.**

Список параметров статистических атрибутов имеет вид

`ИМЯ(имя_связи,тип,статус,дата1,дата2),`

где все параметры кроме первого могут быть опущены (что означает — ограничений не наклазывается). В качестве значений параметров тип и статус могут выступать как одиночные идентификаторы, так и списки через запятую в квадратных скобках, например

```
CHILDREN(TOPIC,[ARTICLE,EXTERNAL],PUBLISHED)
```

посчитает количество итем типа ARTICLE и EXERNAL в текущей рубрике, имеющих статус PUBLISHED.

Имя связи может иметь суффикс -, и префикс .

Конструкция `*AUTHOR` **посчитает количество работ данного автора на текущем сайте, в то время как просто** `AUTHOR` **— всех работ данной персоны независимо от сайта.**

Конструкция TOPIC- перебирает только итем, имеющие связь TOPIC непосредственно с данным итем, в то время как просто TOPIC — на всю глубину рубрикатора.

Первая из двух дат задает нижнюю границу (поиск итем, новее, чем), а вторая — верхнюю.

По умолчанию проверяется атрибут PUBLISHED. Для того чтобы фильтровать по атрибуту LASTCHANGE, после даты надо поставить восклицательный знак.

В отличие от всех остальных случаев подстановки, при подстановке групповых атрибутов через @ в параметрах динамических элементов, в параметрах групповых атрибутов допустим второй уровень подстановки. Например,

```
<:Subst "@CHILDREN(TOPIC,[@TYPES]):>
```

6.3.2 Функции контекста

Кроме групповых атрибутов в контексте бывает еще несколько сущностей, имеющих похожий синтаксис ИМЯ(АРГУМЕНТ). Они называются *функциями контекста*. Их отличие от групповых атрибутов заключается во-первых, в том, что набор их параметров не является жестко фиксированным, а во-вторых, если создание нового группового атрибута возможно через интерфейсы редактирования онтологии, создание новой функции контекста требует модификации кода Communiware.

Определены следующие функции контекста:

TXT(АТРИБУТ) где атрибут перечислимого типа (т.е. имеет таблицу значений). Выдает описание текущего значения данного атрибута на текущем языке. (см. раздел 6.3.3).

TXT(значение:АТРИБУТ) выдает описание указанного значения на текущем языке, рассматривая его как значение указанного атрибута. Например, `TXT(TOPIC:TYPE_ID)` всегда выдаст название типа TOPIC, независимо от значения TYPE_ID в текущем контексте.

NAME(АТРИБУТ ИЛИ ТИП СВЯЗИ) выдает интернационализированное название данного атрибута или типа связи.

ENV(переменная) Выдает значение указанной переменно среды.

TYPE(АТРИБУТ) Выдает тип атрибута (NUMBER, STRING, RICHTEXT, DATE)

RIGHTS(keyword) проверяет права доступа текущего пользователя (раздел 6.3.4).

RIGHTS(keyword,user) проверяет права указанного пользователя.

ACTIVE(linkname,item_id) true, если данный итем имеет связь указанного типа с указанным, и является в данной связи активным.

PASSIVE(linkname,item_d) true, если данный итем имеет связь указанного типа с указанным, и является в данной связи пассивным.

LEN(АТРИБУТ) Длина значения атрибута в символах. Для атрибутов типа RICHTEXT — без учета разметки.

RANDOM(число) Выдает целое случайное число от 0 до указанного числа. Заметим, что существует еще магический атрибут RANDOM без параметра, выдающий вещественное случайное число от 0 до 1.

6.3.3 Интернационализированные названия

Если атрибут имеет таблицу значений, то конструкция `TXT(ИМЯ)` **выдает расшифрованное название данного значения на текущем языке. Вычислимые атрибуты** `ITEM_STATUS` **и** `ITEM_TYPE` **— синонимы** `TXT(STATUS)` **и** `TXT(TYPE_ID)`

Функция TXT позволяет „расшифровать" и константное значение. Для этого необходимо после значения через двоеточие указать имя атрибута, как значение которого мы хотим проинтерпретировать эту константу.

Интернационализированное название имени атрибута можно получить с помощью функции NAME(атрибут).

6.3.4 Проверка прав доступа

Под правами доступа в Communiware обычно понимается существование авторизующей метасвязи между пользователем и итемом.

Исключение представляет собой пользователь со статусом SUPERUSER, который по своему статусу имеет все права на все итемы данного физического сервера.

Кроме того, пользователь, имеющий определенное право на итем сайта, имеет аналогичное право на все итемы, принадлежащие данному сайту.

Каждой авторизующей связи, которая устанавливается между итемом пользователя или группы пользователей, и итемом, на который даются права, соответствует авторизующая метасвязь, распространяющая права доступа на всех потомков данного итема по метасвязи BELONGS.

Связь ALLOWEDUSERS и соответствующая ей метасвязь READS, дающая право на чтение, имеет несколько особый статус. Считает-

ся, что по умолчанию итем доступен для чтения всем, в том числе и анонимам, поэтому какие-либо ограничения начинают проверяться, только если у итема присутствует метасвязь READS хотя бы с одним пользователем.

Итем сайта обычно должен быть открыт для чтения всем, поэтому дать какому-либо пользователю право на чтение всего содержимого сайта, привязав его связью ALLOWEDUSERS к корню сайта, нельзя.

С другой стороны, считается, что наличие права производить любые действия с итемом дает право этот итем читать. Поэтому право на чтение дает не только метасвязь READS, но и любая другая авторизующая метасвязь (именно это право проверяется с помощью `RIGHTS(ACCESS)`).

Поэтому можно дать пользователю право на чтение всех защищенных рубрик сайта, привязав его к корню сайта авторизующей связью, которая нигде больше на сайте не используется.

6.3.5 Типы атрибутов

Атрибуты в Communiware могут быть следующих типов:

STRING Строка без форматирования. Как правило используется для хранения идентификаторов, таких как типы итем. Если с таким атрибутом связан жестко фиксированный набор значений (таблица значений), то говорят о *перечислимом атрибуте*. Используется также и для таких вещей как URL, параметры и прочие текстовые значения, которые форматирования содержать не могут, так как предназначены не только для восприятия человеком.

NUMBER Числовое значение. По умолчанию — вещественное. С числовыми атрибутами также может быть связана таблица значений. При выводе числовых атрибутов используется набор форматов, поддерживаемый функцией `printf`.

DATE Значение типа дата/время. Атрибут типа DATE в Communiware всегда хранит время с точностью до секунды. Внутренний формат представления имеет вид YYYY.MM.DD HH.MM.SS. Константы типа время следует указывать именно в таком

формате. При форматировании используется набор форматов функции `strftime`**.**

Если с атрибуту типа DATE соответствует атрибут с именем `ИМЯ_ACCURACY`**, принимающий те же значения, что и стандартный атрибут** `WRITTEN_ACCURACY` **то при показе этого атрибута с помощью динамического элемента Attr, все компоненты даты, задающие промежутки времени меньше объявленной точности, будут подавлены.**

RICHTEXT Содержит текст с форматрованием. Внутренне хранится как текст с HTML-тэгами, в полях редактирования показывается как текст с выделениями. Для форматирования атрибутов этого типа используется формат состоящий из набора букв pila, которые имеют следующий смысл:

p подавить тэги параграфов, элементов списка и прочие, вызывающие переход на новую строку.

l подавить гиперссылки (используется при показе текста внутри ссылки)

i подавить встроенные картинки

a подавить всю разметку

6.4 Идентификаторы итемов

Идентификаторы итемов могут встречаться в тексте шаблона как правило, либо при формировании ссылок на другие итемы, либо при явном создании контекста, при помощи „тривиального" фильтра. Поскольку большая часть итемов на сайте находится в ведении редактора и модератора, а не разработчика, следует насколько возможо, избегать явного указания итемов в шаблоне. Особенно это касается шаблонов, которые могут быть использованы на нескольких сайтах, поскольку каждый сайт будет иметь свой собственный набор итемов.

Несколько большей свободой располагает разработчик при использовании явных имен шаблонов.

На каждом сайте обычно есть ряд итемов, играющих специфическую роль. Например, корневая страница сайта, стандартный

шаблон оформления, корневой итем документации. Для того, чтобы можно было ссылаться на эти итемы не по их идентификаторам, уникальным для каждого сайта, а по их роли на сайте, вводится понятие *виртуальных страниц*. Это набор стандартных имен, которым на каждом сайте ставится в соответствие определенные итемы. Установить это соответствие можно либо при постинге итема, с помощью атрибута времени самбита `VIRTUAL`, либо с помощью соответствующего скрипта, входящего в комплект инструментария для разработки онтологий.

В шаблонах на виртуальные страницы можно ссылаться практически везде, где допустим идентификатор итема, используя нотацию **имя*.

6.5 Фильтры.

Фильтр, это основной способ получить информацию из базы данных. Большинство фильтров в Communiware возвращают список итемов, тем или иным способом зависящий от того итем, в контексте которого они вызваны.

Фильтры бывают трех основных типов:

SQL-фильтры Имеют вид `имя(параметры)` где параметры — необязательны. Эти фильтры хранятся в базе данных и их можно просмотреть через соответствующий пункт верхнего меню в интерфейсе модератора.

Perl-фильтры Имена таких фильтров всегда начинаются с двух двоеточий. Каждому такому фильтру соответствует модуль в пространстве имен Communiware::Filter. Документация по таким фильтрам, установленным в системе обычно доступна в соответствующем разделе служебного сайта. В стандартный комплект поставки вхходят фильтры `::Pic`, показывающий список картинок в текущем итеме, `::Mail` используемый в почтовых рассылках, и `::MnogoSearch`, реализующий полнотекстовый поиск.

«Вырожденные» Явно указанная последовательность ITEM_ID через запятую в круглых скобках также является полноправным фильтром, и может использоваться везде, где используются фильтры. Так, например конструкция

```
<:Loop (some_item):>
...
<:EndLoop:>
```

позволяет получить контекст явно указанного итема.

Следует учесть, что фильтры такого вида всегда создают контексты с указанным ITEM_ID, даже если такого итема не существует.

Если необходимо, чтобы последовательность значений интерпретировалась не как ITEM_ID, а как какой нибудь другой атрибут, то нужно указать имя атрибута после закрывающей круглой скобки через двоеточие.

```
(TOPIC,ARTICLE,BINARY):TYPE_ID
```

При разработке фильтров следует соблюдать баланс между гибкостью и простотой использования. Надо позволить переопределить через параметры некоторые особенности поведения фильтра, например типы связи, по которым он выбирает связанные итем, но все что можно взять из текущего контекста, надо брать из текущего контекста.

Глава 7

Описание динамических элементов

7.1 Визуализация атрибутов итем

7.1.1 Attr

Выводит указанный атрибут итем.

Синтаксис

```
<:Attr ИМЯ [формат]:>
```

Динамический элемент Attr выводит значение указанного атрибута итема, форматируя его по указанной спецификации формата, или некоторым способом по умолчанию.

Возможности форматирования зависят от типа атрибута, поэтому в этот динамический элемент передается имя атрибута, а не значение для вывода.

Числовые и строковые атрибуты можно форматировать с помощью любых форматов, допустимых для функции `printf` языка C, атрибуты типа «дата» — форматов, допустимых для функции `strftime`.

Результат этого форматирования будет показан в пользовательском браузере как есть. Т.е. если атрибут содержит символ & или <, то именно этот символ и появится на экране.

Атрибут типа RICHTEXT показывается с использованием содержащейся в его значени HTML-разметки.

Формат для атрибутов типа RICHTEXT (форматированный текст) представляет собой набор букв-флагов, указывающих какие тэги следует подавить при выводе.

Допустимы флаги i,p,l и a. i — удаляет тэги img, заменяя их на альтернативный текст. l — удаляет гиперссылки (что позволяет использовать текст итем в качестве текста гиперссылки. См LinkBox, раздел 7.2.2), p — удаляет параграфы, списки и прочие элементы разметки вызывающие явные разрывы строк. a — удаляет всю HTML-разметку.

Если выводимый атрибут NAME имеет тип DATE, и в контексте присутствует атрибут NAME_ACCURACY, то вывод более мелких компонентов даты подавляется. Например, при WRITTEN_ACCURACY=DAY, из шаблона по которому форматируется атрибут WRITTEN будут удалены спецификаторы формата %H,%M и %S.

7.1.2 Counter

```
<:Counter ИМЯ "счетное слово"  [K|M] [смещение]:>
```

Выводит числовой атрибут итема, сопровождая его счетным словом в корректной форме. Счетное слово задается в виде `основа(1|2|5)` **где** `основа` **— общая часть для всех форм,** `1` **— окончание в единственном числе,** `2` **— окончание для случая 2,3,4 и** `5` **— окончание для случая 5–10. Например,** `реплик(а|и|)`**, или** `байт(|а|ов)`**.**

В случае, если третьим аргументом указано K или M, значение атрибута делится на 1024 или 1048576 соответственно, и показывается как дробное число, сопровождаемое суффиксом K или M. Счетное слово при этом не используется.

Дополнительный аргумент «смещение» позволяет прибавить фиксированную константу к значению.

7.1.3 Text

Показывает текст.

Синтаксис:

`Text` *параметр=значение ...*

Параметры этого динамического элемента, в отличие от всех остальных, именованные, а не позиционные, т.е. могут быть указаны в любом порядке, и распознаются по ключевым словам перед знаком равенства.

Допустимы следующие параметры:

default_lines число строк текста, показываемых по умолчанию. Если текст подлежит обрезанию, то будет показано примерно указанное число строк. Поскольку в HTML разбиение текста на строки сильно зависит от настроек браузера, строкой считается 63 символа.

max_lines максимальное число строк которые надо показать. Если этот параметр указан, то тексты более указанного числа строк будут обрезаны и от них показано только число строк, указанное в параметре default_lines. Если указан только default_lines, то max_lines считается равным default.

В случае если включена обрезка текста, в конце каждого текста формируется ссылка на итем с его шаблоном по умолчанию (предполагается что в нем-то текст показан полностью) или по явно заданному шаблону. В случае, если текст был обрезан, эта ссылка предваряется многоточием.

link_text текст ссылки на полный текст итема. По умолчанию — два знака больше. Если текст равен `nolink`, вывод ссылки подавляется.

link_template шаблон, который следует указать в URL ссылки на полный текст итема (если он не совпадает с шаблоном по умолчанию).

link_style Дополнительные атрибуты HTML-тэга A, образующего ссылку на полный текст. См. описание динамического элемента ItemLink (раздел 7.2.1).

format Дополнительные операции, которые следует предпринять над текстом перед его выводом. Необходимость этого параметра объясняется тем, что динамический элемент Text можно использовать в контекстах, где допустимы не все тэги HTML, которые могут встретиться в текстах итема. Значение этого параметра представляет собой комбинацию букв

i,p,l и a. i — удаляет тэги img, заменяя их на альтернативный текст. l — удаляет гиперссылки (что позволяет использовать текст итема в качестве текста гиперссылки. См LinkBox, раздел 7.2.2), p — удаляет параграфы, списки и прочие элементы разметки вызывающие явные разрывы строк. a — удаляет всю HTML-разметку.

7.1.4 Source

Показывает текст в том виде как он хранится в базе, т.е. делает все тэги видимыми на экране.

Синтаксис

`Source`

Никаких параметров этот динамический элемент не имеет.

7.1.5 Subst

Вычисляет свои аргументы как арифметичекое выражение (см. раздел 4.2) и выводит результаты на экран.

Синтаксис

`Subst` *параметры*

Динамический элемент Subst применяется в тех случаях, когда нужно произвести подстановку атрибутов в параметры произвольного HTML-тэга.

7.2 Формирование гиперссылок

7.2.1 ItemLink

Формирует ссылку на итем.

Синтаксис

`ItemLink` *итем текст-ссылки* [*длина* [*шаблон* [*параметры* [*атрибуты*]]]]

Формирует ссылку на итем. Первый аргумент может быть как именем атрибута, содержащего идентификатор итема, так и явно указанным ITEM_ID или ссылкой на виртуальную страницу.

Второй аргумент — текст ссылки. Если третий аргумент не пуст, то текст обрезается до указанного числа символов. Текст ссылки может содержать HTML-разметку.

Шаблон — имя шаблона, по которому должен быть показан итем в результате перехода по ссылке.

Параметры — дополнительные параметры HTML-тэга A, такие как class, target или вызовы JavaScript функций. Должны быть указаны как один параметр динамического элемента, в виде списка пар имя=значение, разделенных запятыми. Если запятая должна быть включена внутрь значения (что часто бывает в JavaScript-коде), то ее следует защитить обратной косой чертой, например

```
"class=link,onClick=do_something(a\,b)"
```

Атрибуты — список атрибутов, которые необходимо передать в контекст итема, вызванного по ссылке. Имеет вид списка через запятую, в котором можно использовать либо просто имена, либо пары имя=значение. В первом случае будет передано то значение, которое атрибут имеет в текущем контексте, во втором, указанное.

7.2.2 LinkBox

Аналогично ItemLink, но позволяет использовать в качестве текста ссылки произвольный фрагмент шаблона, содержащие динамические элементы.

Синтаксис

`<:LinkBox` *итем* [*шаблон* [*параметры* [*атрибуты*]]]`:>`
фрагмент шаблона
`<:EndBox:>`

7.2.3 Script

Формирует ссылку на скрипт Communiware.

Синтаксис

`Script` *имя-скрипта текст* [*атрибуты* [*параметры*]]

Формирует ссылку на скрипт Communiware.

Скрипты едины для всего физического сервера, поэтому имя сайта (и, обычно, имя итема) им необходимо пробрасывать через URL как атрибуты. Кроме того, скрипты ожидают получения атрибута REFERER, указывающего на какую страницу следует передать управление по завершении работы скрипта.

Динамический элемент Script автоматически генерирует эти атрибуты в URL, если параметр *атрибуты* пуст. Если он не

пуст, то автоматически передаются только SERVER и REFERER, а ITEM_ID необходимо явно указывать в списке передаваемых атрибутов.

Так же как и в ItemLink, список атрибутов может содержать просто имена, тогда значения будут взяты из текущего контекста, и пары имя=значение.

Интерфейсные скрипты Communiware как правило рассчитаны на работу в два этапа — на первом они показывают пользователю некотрую форму, на втором, обрабатывают результаты ввода в эту форму.

Скрипт отличает второй этап от первого по наличию в контексте некоторого атрибута, имя которого совпадает с именем кнопки самбита сгенерированной скриптом формы. Поэтому, правильно сформировав список атрибутов в динамическом элементе Script дизайнер шаблона может заставить скрипт выполнять действие непосредственно в результате прохода по ссылке, без показа промежуточной формы. См. также динамический элемент Referer (раздел 7.7.1).

7.2.4 Reply (не поддерживается)

Формирует ссылку на скрипт reply.

Фактически эквивалентент `Script reply`.

Начиная с версии 0.93 этот динамический элемент не поддерживается. Его необходимо заменить либо на `Script reply`**, либо на шаблон постинга реплики.**

7.2.5 Image

Формирует встроенную в веб-страницу картинку.

Синтаксис:

`Image src=`*filename имя=значение ...*

Динамический элемент Image формирует HTML-тэг img. В качестве параметров он получает пары атрибут=значение. Допустимы все атрибуты, допустимые у тэга img, кроме того можно использовать параметр `item=`*идентификатор* **который указывает, что указанный файл следует искать в области иллюстраций указанного итема. По умолчанию поиск производится в итеме, являющемся виртуальной страницей** `*pictures` **текущего сайта.**

Значением параметра `src` **может быть только имя файла, без пути.**

В случае, если явно указаны параметры `width` **и** `height`**, то используются их значения. В противном случае подставляются реальные размеры графического файла.**

Если файл имеет расширение `.swf`**, то вместо тэга** `img` **формируются тэги** `object` **и** `embed` **необходимые для визуализации Shockwave Flash.**

7.2.6 Lib

Формирует ссылку на библиотеку стилей (`.css`**) или ссылку на библиотеку JavaScript-процедур (**`.js`**).**

Синтаксис:

`Lib` *filename имя=значение*

По умолчанию формирует ссылку (HTML-тэг `<link rel=text/css href=...>` **или** `<script src=...>` **на файл в области иллюстраций итема, являющегося значением виртуальной страницы** `*picures` **для текущего сайта. Указать другой итем можно с помощью параметра** `item=`*идентификатор*. **Все остальные параметры трактуются как дополнительные атрибуты генерируемого тэга.**

7.3 Работа с протоколом HTTP

7.3.1 Header

Формирует заголовок HTTP или (в случае шаблонов почтовой рассылки) письма.

Синтаксис

`Header` *имя-заголовка выражение*

Инструктирует Communiware включить в заголовок HTTP-ответа (или письма электронной почты) соответствующую строку. Описание возможных значений параметра *имя-заголовка* **см в RFC 2616 для HTTP и RFC 822 для электронной почты.**

Значение заголовка вычисляется по тем же правилам (см. раздел 4.2) что и значение атрибута в Define и Use.

В случае HTTP специальным образом обрабатываются следующие заголовки:

Location Изменяет код HTTP-ответа на 302 (redirect). Значением должна быть полная URL страницы, на которую надо осуществить редирект.

WWW-Authenticate Изменяет код HTTP-ответа на 401 (Authorization required). Пользоваться не рекомендуется, так как для реализации авторизации существует более высокоуровневые динамические элементы Authentify и Authorize.

Expires Стандарт HTTP требует, чтобы значением этого заголовка было абсолютное время устаревания страницы по Гринвичу. Поскольку яэык шаблонов не предназначен для подобного рода вычислений, динамический элемент Header поддерживает специальный синтаксис, позволяющий задать время относительно времени обработки шаблона. Значение должно иметь вид *число единица-времени* **где единица времени может быть s (секунды), m (минуты), h (часы) , d (дни), M (месяцы) и y(годы) .**

Например,

```
<:Header Expires 10m:>
```

задает время устаревания страницы через 10 минут после ее генерации.

Set-Cookie В параметре `expires` поддерживается синтаксис, аналогичный синтаксису заголовка Expires. Т.е. можно задавать срок устаревания куки в относительных единицах, таких как 2d (два дня) или 10y(10 лет).

Наличие динамического элемента Header, управляющего непосредственно web-сервером, делает ненужным использование HTML-тэга META HTTP-EQUIV, который интерпретируется распространенными браузерами с нарушением стандарта.

7.3.2 Authentify

Проверяет соответствие текущего пользователя заданному фильтру, запрашивая, при необходимости, имя и пароль через HTTP Basic authentication.

Синтаксис

`Authentify` [*фильтр* [*realm*]]

Динамический элемент Authentify проверяет, что пользователь — тот, за кого он себя выдает. Если аутентичность пользователя подтверждается только наличием куки AUTHOR_ID, то выдает код ответа 401, заставляя пользователя сообщить имя и пароль в соответствии с протоколом HTTP Basic authentication (RFC 2617). В случае если в заголовках HTTP-запроса уже присутствует имя и пароль, проверяет правильность пароля по базе данных Communiware.

Если параметр *фильтр* не указан, то любой зарегистрированный пользователь Communiware считается прошедшим проверку. Если указан фильтр, то страница показывается только пользователям, возвращаемым данным фильтром, а остальные получают код HTTP ответа 403 (Forbidden) и пустую страницу. В случае необходимости использования сложных проверок доступа, рекомендуется использовать динамический элемент `Authorize` (см. раздел 7.3.3), который позволяет выполнять более гибкие проверки, и, в большинстве случаев, работает более эффективно.

Параметр realm позволяет изменить имя авторизационной области. По умолчанию Authentify (так же как и все скрипты) использует realm Communiware. Поэтому указание другого realm заставит пользователя вводить пароль еще раз, даже если он его уже ввел для доступа к какому-либо скрипту.

Заметим что параметр realm никак не влияет на список пользователей, имеющих право доступа.

7.3.3 Authorize

Разрешает доступ на страницу, только пользователям, для которых выполняется заданное условие.

Синтаксис

`Authorize` *логическое выражение*

В случае, если пользователь запросивший страницу, не представил имени и пароля по basic authentication, вызывает запрос аутентикации, аналогично динамическому элементу Authentify (раздел 7.3.2).

Если в запросе содержится имя и пароль, и пароль соответствует имени, производит вычисление логического выражения 4.3, и

в случае если оно имеет значение false, возвращает пользователю HTTP-код 403 (forbidden).

Обычно этот элемент используется для проверки прав пользователя с помощью функции контекста RIGHTS (см разделы 6.3.2 и 6.3.4).

Следует обратить внимание на то, что проверка прав с помощью
`<:Authorize @{RIGHTS(`*право*`)}:>`
существенно более эффективна, чем эквивалентная проверка
`<:Authentify AllActiveTyped(`*право*`,AUTHOR):>`

7.4 Управление выполнением шаблона

7.4.1 If

Условный оператор.

Синтаксис

`<:If` [`!`] *атрибут* [*"логическая операция"операнд*] `:>`
фрагмент шаблона [`<:Else:>`
фрагмент шаблона]
`<:EndIf:>`

Вычисляет логическое выражение. Первым аргументом данного выражения является *имя атрибута*, а не значение. Это позволяет использовать сравнение на больше-меньше не только для чисел и дат, но и для некоторых перечислимых атрибутов, таких как статус. Знание имени (и, соответственно, типа) атрибута позволяет этому динамическому элементу использовать подходящую функцию сравнения.

Если в качестве логического выражения используется просто имя атрибута, то это выражение истино, если атрибут существует, и его значением не является пустая строка или число 0.

Кроме того, можно использовать в качестве логического выражения имя (мета)связи, оно истино тогда, когда существует такая связь между текущим итемом и текущим пользователем, и ключевое слово ACCESS, которое истино тогда, когда текущий пользователь имеет право на чтение текущего итема (фильтр может сгенерировать список, содержащий и недоступные итемы).

В случае если условие истино, то обрабатывается первый фрагмент шаблона, если ложно — то второй (если он есть).

7.4.2 IfAttr (не поддерживается)

Упрощенная форма условного оператора. Проверяет указанный атрибут на пустоту и в зависимости от этого выводит либо второй, либо третий аргумент.

Синтаксис

`IfAttr` *имя строка1 строка2*

Если атрибут *имя* **существует в контексте и имеет истинное, т.е. непустое и ненулевое значение, то в результирующий HTML будет выведена** *строка1*. **Иначе —** *строка2*.

Начиная с версии 0.93 этот динамический элемент не поддерживается и подлежит замене на If.

7.4.3 Cond и Case

Организует множественное ветвление. По синтаксису больше похож на оператор cond в языке LISP, чем на оператор CASE в Pascal или switch в C. Впрочем в близких по уровню абстракции к языку шаблонов Communiware языках PAL и xBase, семантика конструкции множественного ветвления очень близка к семантике нашего Cond.

Синтаксис:

```
<:Cond:>
<:Case логическое выражение:>
Фрагемент шаблона
<:EndCase:>
<:Case логическое выражение:>
Другой фрагмент шаблона
<:EndCase:>
....
<:EndCond:>
```

В блоке Cond могут содержаться только блоки Case и HTML-комментарии. При обработке шаблона все блоки Case в данном Cond просматриваются последовательно, и проверяются логические выражения, указанные в параметре Case. Если выражение вычислилось в истинное значение, то обрабатывается фрагмент шаблона, содержащийся в соответствующем блоке Case, а все последующие блоки игнорируются.

Поэтому для реализации конструкции вида если A, то X, иначе, если B, то Y иначе Z, следует использовать конструкцию вида

```
<:Cond:>
<:Case @A:>
X
<:EndCase:>
<:Case @B:>
Y
<:EndCase:>
<:Case 1:>
Z
<:EndCase:>
<:EndCond:>
```

Как мы видим, беспокоиться о том, что в некоторых случаях условия B и A будут истины одновременно, не следует. Если выполнилось условие A, будет выведено X, а блок с условием B не будет даже рассматриваться.

По этой же причине, использование в последнем блоке условия, которое истино всегда, например „1" или „otherwise" (которое будет воспринято Communiware как не пустая константная строка), означает — выполнять этот блок, если ни одно другое условие не подошло.

Следует отметить, что в динамическом элементе Cond первый операнд первого условия не имеет никакого особого смысла, в отличие от If. Поэтому подстановка атрибутов должна выполняться по общепринятым правилам, так же как и в динамических элементах Authorize, Check и Item.

7.4.4 Include

Включает шаблон фрагмента. Параметры, указанные после имени шаблона передаются в этот шаблон как атрибуты с именами ARG1, ARG2 и так далее.

Синтаксис

`Include` *шаблон* [*параметры*]

Включает в текущую точку генерируемого HTML результат обработки указанного шаблона фрагмента. В имени шаблона выполняется подстановка атрибутов и раскрытие виртуальных страниц.

Если в имени шаблона не использовано ни того, ни другого, при сохранении шаблона, содержащего этот динамический элемент, будет создана связь USES, соединяющая его со включаемым шаблоном. Поскольку связь USES — обычная иерархическая связь, не допускающая циклов, рекурсивное включение с помощью Include невозможно.

Если у динамического элемента Include указано более одного параметра, значения всех параметров после имени шаблона доступны в контексте включенного шаблона как атрибуты с именами ARG1, ARG2 и так далее.

Шаблон включаемый динамическим элементом Include должен иметь тип ELEMENT, и должен раскрываться в последовательность из одного и более закрытых HTML-тегов. Начиная с версии 0.94 планируется ввести проверку синтаксиса шаблонов типа ELEMENT, которая не будет позволять загрузить в систему шаблон, содержащий незакрытые тэги.

В тех случаях, когда нужно включить шаблон, обрамляющий некоторую часть страницы, следует использовать динамический элемент Surround (раздел 7.4.5).

7.4.5 Surround

Включает шаблон оформления, окружающий указанный фрагмент шаблона.

Синтаксис

`<:Surround` *шаблон логическое выражение* `:>` *фрагмент шаблона* `<:EndSurround:>`

Включает шаблон типа CONTAINER, который позволяет окаймить участок основного шаблона какими-либо тегами, например создать шапку и подвал с использованием HTML-таблицы.

Шаблон типа CONTAINER всегда содержит динамический элемент Content (раздел 7.4.6) на место которого включается блок шаблона, содержащийся между Surround и EndSurround.

В отличие от динамического элемента Include, Surround не позволяет передавать параметры включаемому шаблону, но позволяет условное включение.

Если после имени шаблона указано логическое выражение, то включаться шаблон будет только в том случае, если это выражение

истино. В противном случае будет просто обработан блок, содержащийся между Surround и EndSurround. Если логического выражения нет, шаблон включается безусловно.

Использование динамического элемента Surround позволяет гарантировать, что все тэги, которые были открыты во вложенном шаблоне, обязательно будут закрыты (поскольку они закрываются в том же шаблоне, что и открываются).

7.4.6 Content

Указывает место, где должен быть включен блок, окружаемый шаблоном типа CONTAINER

Синтаксис

`<:Content:>`

Этот динамический элемент не имеет никаких параметров. Он может присутствовать только в шаблонах типа CONTAINER, предназначенных для включения с помощью динамического элемента Surround (см раздел 7.4.5). При каждом применении шаблона типа CONTAINER должен быть обработан ровно один элемент Content. Т.е. в шаблоне может содержаться несколько элементов Content, например в разных ветках Cond, но в каждой конкретной ситуации может быть выполнен только один из них.

7.4.7 IncludeVirtual

Эмулирует директиву SSI <–#include virtual=...–>

Синтаксис

`IncludeVirtual` *url*

Включает в генерируемую страницу выдачу внешнего по отношению к Communiware CGI-скрипта, выполняющегося на том же сервере.

URL должна быть локальной (от корня сервера, например `/cgi-bin/mysc`
и скрипт должен быть именно CGI, т.е. выполняемый отдельным процессом, а не, например mod_perl.

Предполагается, что скрипт расчитан на использование в качестве включаемого, и генерирует фрагмент HTML, а не полный HTML-документ.

Основное назначение данного динамического элемента — включение в генерируемый HTML кода баннерных систем.

7.5 Изменение контекста

7.5.1 Define

Определяет новый атрибут контекста.

Синтаксис

`Define` *имя арифметическое выражение* [*тип*] [`ifabsent` | `ifempty`]

Определяет атрибут *имя* **со значением, равным результату вычисления выражения. Арифметическое выражение занимает произвольное число параметров. Если указан тип, то созданный атрибут помечается, как атрибут указанного типа. Если указано ключевое слово** `ifabsent`**, то атрибут получает значение только в том случае, если он отсутствует в контексте. Т.е. если атрибут с тем же именем был получен в результате сабмита формы настройки, в куках, или явно указан в URL, значение не переопределяетя. Если при этом указан тип, то атрибут помечается как имеющий соответствующий тип. По умолчанию считается что атрибут имеет тип STRING.**

Значение `ifempty` **позволяет переопределить атрибут, даже если он явно задан, но имеет пустое значение, например, соответствует незаполненному полю ввода.**

Атрибуты Communiware это не переменные императивного языка программирования. Результат выполнения нескольких динамических элементов Define с одним и тем же именем атрибута не определен.

7.5.2 PassParams

Управляет передачей дополнительных атрибутов (введенных в форме или определнных через Define) через ссылки или через форму постингов.

Синтаксис

`PassParams` *список имен атрибутов через запятую*

Если этот динамический элемент употреблен вне формы постинга, то ко всем ссылкам, сгенерированным с помощью динамических элементов Communiware (ItemLink, LinkBox, Continuation, Script) на этой странице (в текущей версии — только ниже по тексту этого динамического элемента) будет добавлена передача этих

атрибутов контекста.

Если он употреблен внутри формы постинга, то он создает HTML-тэги input (типа hidden), которые приводят к тому, что в контексте, образующемся в результате сабмита формы, будут присутствовать указанные атрибуты с теми значениями, которые они имели в момент отображения формы.

И в том и в другом случае пробрасывать имеет смысл атрибуты, введенные пользователем в форме настроек или созданные с помощью Define.

Следует учесть, что использование PassParams приводит к появлению длинных неудобочитаемых URL и разрастанию размеров сгенерированной HTML-страницы. Поэтому пользоваться им имеет смысл с осторожностью. Возможно, лучшим решением окажется запоминание этих атрибутов в куках.

7.5.3 Do

Задает процедурную вставку (перл-модуль или процедуру) выполняемую в текущем контексте.

Синтаксис:

`Do` *спецификация-времени Модуль*

Выполняет указанную функцию в указанный момент обработки HTTP-запроса. Функция задается либо именем perl-модуля, в этом случае по соглашению, принятому в `mod_perl` в данном модуле вызывается функция `handler`, либо именем модуля и именем функции, например `MyModule::myfunc`. Как и в прочих случаях использования пользовательских модулей в Communiware, модуль должен быть расположен в определенном namespace внутри namespace Communiware. В данном случае это `Communiware::Postprocess`. Т.е. задание в динамическом элементе Do модуля `MyModule`, приведет к выполнению функции `Communiware::Postprocess::MyModule::handler()`.

Спецификация времени имеет различную форму в зависимости от того, находится данный динамический элемент внутри формы постинга или вне ее.

Вне формы постинга доступны только две спецификации:

immediate **Непосредственно сейчас, т.е. в момент, когда парсер шаблонов обнаружил данный динамичекий элемент. В случае,**

если при выполнении такой процедурной вставки возникла ошибка, сообщение о ней выводится непосредственно на страницу

postrequest В конце обработки шаблона, после отдачи HTML-страницы пользователю. В этот момент вывести сообщение об ошибке пользователю уже невозможно, поэтому информация об ошибках доступна только в error_log-е сервера.

В формах постинга, предполагающих двухфазную обработку, т.е время визуализации и время обработки результатов ввода, количество доступных спецификаций увеличивается. Появляется возможность задать выполнение в моменты:

prepost До начала обработки постинга, т.е. до старта транзакции, изменяющей базу данных.

preedit до выполнения операции create_item, update_item или delete_item, соответствующей текущему контейнеру Item, но, возможно, после обработки других контейнеров.

postedit После выполнения действий, изменяющих базу данных, но до завершения транзакции. Возможно, до обработки других контейнеров.

postpost После завершения транзакции. Это единственное место, откуда можно запустить внешний процесс, которому необходимо воспользоваться результатами транзакции.

При возникновении ошибки в обработчиках времени сабмита, кроме postpost, сообщение помещается в атрибут контекста ERROR, и все действия, произведенные в данной форме постинга, отменяются.

В случае возникновения ошибок на стадии postpost, сообщение об ошибке помещается в атрибут WARNING и дальнейшее поведение соответствует поведению Communiware в случае срабатывания Check warn (см. раздел 7.8.5).

7.6 Формирование списков

7.6.1 List

Формирует список итема, показывая элемент по шаблону

Синтаксис

`List` *фильтр шаблон сортировка группировка обрезание*

Формирует список итемов, удовлетворяющих указанному *фильтру* **и показывает каждый из них по указанному** *шаблону*. **Шаблон должен быть шаблоном фрагмента. Порядок итемов определяется указанным критерием сортировки.**

Если критерий сортировки не указан, то итемы выводятся в том порядке, в котором они возвращены фильтром.

Сортировка списков

Критерий сортировки обычно представляет собой список атрибутов через запятую. Каждый атрибут может быть предварен знаком плюс или минус, указывающим порядок сортировки. По умолчанию — плюс, т.е. значения атрибута будут упорядочены по возрастанию. Наибольший вес при сортировке имеет первый из указанных атрибутов. Т.е. конструкция `ORDNUM,-PUBLISHED` **указывает, что итемы должны быть упорядочены по значению свойства связи ORDNUM, а итемы с одинаковым значением ORDNUM — в порядке убывания дат публикации.**

Древовидная сортировка

Особым случаем является древовидная сортировка. Она используется как правило при показе итемов, связанных транзитивной, или иерархической связью. При древовидной сортировке потомки данного итем располагаются непосредственно за ним, а его «братья» — после потомков. При этом в контекст добавляется числовой атрибут LEVEL, указывающий расстояние от данного итем до корня дерева, а также атрибуты NEXT_SIBLING и PREV_SIBLING, содержащие идентификаторы соседей на том же уровне (если они есть). Соседями считаются потомки того же предка, которые в порядке сортировки по дополнительным критериям оказались непосредственно перед или непосредственно после данного итема.

Кроме того, добавляется атрибут UNCLES, представляющий собой список нулей и единиц длины LEVEL. Единица в соответствующей позиции списка означает, что у предка на этом уровне есть младший брат (NEXT_SIBLING).

Для древовидной сортировки пригоден не всякий фильтр. Например, стандартный фильтр AllPassiveLinked, хотя и возвращает потомков по иерархической связи, не может быть использован для древовидной сортировки. Фильтр, пригодный для древовидной сортировки должен возвращать помимо идентификатора итема, еще и идентификатор его ближайшего предка по требуемой связи. Обычно, атрибут, содержащий данный идентификатор, называется PARENT_ID.

Синтаксис древовидной сортировки следующий:

`t:`*имя1*`,`*имя2*`,`*критерий сортировки <<братьев>>*

где *имя1* **— имя атрибута содержащего идентификатор текущего итема (обычно ITEM_ID),** *имя2* **— имя атрибута, содержащего идентификатор предка. Эти два атрибута используются для установления связи между предками и потомками.** *критерий сортировки «братьев»* **это обычный критерий сортировки, используемый для упорядочивания итемов на одном уровне иерархии.**

Еще один специальный вид критерия сортировки имеет вид `c:`*число*. **Это на самом деле не сортировка, а способ задани обрезания списка. При указании параметра сортировки** `c:`n **будут показаны только первые** n **элементов списка в том порядке, в каком эти элементы вернул фильтр.**

В отличии от обсуждаемого ниже параметра «обрезание» этот способ обрезания списка не позволяет автоматически формировать ссылку на следующие n **элементов с помощью динамического элемента Continuation.**

Пользоваться этим способом не рекомендуется, он сохранен только для совместимости с предыдущими версиями.

Группировка списков

Очень часто требуется сгруппировать итемы в списке по значению того или иного атрибута, т.е. визуально выделить группы итем с одинаковым значением атрибута. Типичный пример — древовидная сортировка, где посредством группировки по атрибуту LEVEL можно визуализировать древовидную структуру списка.

Критерий группировки имеет вид
ИМЯ:*тип*(*параметры*)
Где *ИМЯ* — имя атрибута по которому осуществляется группировка, тип — тип группировки, который может быть `tag`, `include` **и** `attr`.

Если указан параметр группировки, то в контексте элемента списка будет присутствовать атрибут с имерем `PREV_`*ИМЯ*, **где *ИМЯ* — имя атрибута группировки, со значением равным значению атрибута группировки в предыдущем элементе списка.**

Группировка типа `tag` **предназначена для работы с числовыми атрибутами. Если атрибут увеличивает свое значение, то указанный в качестве параметра HTML-тэг будет выведен $n_i - n_{i-1}$ раз, где n_i и n_{i-1} — значение атрибута группировки в текущем и предыдущем элементах списка. Если атрибут уменьшает свое значение, то соответвующее число раз будет выведен соответствующий закрывающий тэг.**

Таким образом визуализировать иерархический список рубрик в виде вложенных ненумерованных списков можно посредством конструкции

```
<UL>
<:Loop
TypedTree(TOPIC,TOPIC) t:ITEM_ID,PARENT_ID,ORDNUM
LEVEL:tag(UL):>
<LI><:Attr TITLE:>
<:EndLoop:>
</UL>
```

Тэг, указанный в качестве параметра группировки `tag` **может иметь параметры, например** `LEVEL:tag(DIV class=menu)`. **Эти параметры будут использованы при выводе открывающего тэга, но проигнорированы при выводе закрывающего. Подстановка атрибутов в этом параметре производится в момент обработки динамического элемента List, т.е. еще до выполнения фильтра. Поэтому надеятся на то что туда будет подставлено правильное значение, например, атрибута группировки, не стоит.**

Группировка типа `include` **позволяет включить между элементами списка указанный шаблон. Параметром этой группировки является имя шаблона. Шаблон будет обработан в контексте следующего элемента списка.**

Группировка типа `attr` не выполняет никаких действий. Ее единственным эффектом является создание в контекста атрибута с предыдущим значением атрибута группировки.

Обрезание списка

Параметр *обрезание* задает максимальное количество элементов списка, которые могут быть показаны на одной странице.

Этот параметр может иметь вид *число* или *число*!. Если указан восклицательный знак, то будут показаны первые n элементов списка и никаких прочих действий предпринято не будет.

Если восклицательный знак отсутствует, то включается механизм постраничного показа списков, который работает следующим образом:

1. **Если в контексте присутствует атрибут NEXT_ITEM то отбрасываются первые элементы списка до тех пор пока не встретится итем с ITEM_ID равным значению этого атрибута.**
2. **Показываются n следующих элементов списка.**
3. **ITEM_ID первого непоказанного элемента запоминается в атрибуте NEXT_ITEM для последующего использования динамическим элементом Continuation.**

Для формирования ссылок на следующую или начальную страницу разрезанного списка используется динамический элемент Continuation (см. раздел 7.6.3.

Особенности контекста элемента списка

В контексте элемента списка кроме атрибутов текущего итем и общих для всей страницы атрибутов HTTP запроса присутствует ряд дополнительных атрибутов.

***Во-первых*, фильтр может вернуть дополнительные поля, которые не являются атрибутами итем. Они будут доступны в запросе, как атрибуты с теми именами, которые были им указаны в предложении select SQL-запроса. Так например, многие стандартные фильтры возвращают значение ORDNUM, которое является свойством связи между итемами, и может быть однозначно определено**

только в момент выполнения фильтра, так как в этот момент известен и итем, соседи которого извлекаются, и извлекаемый итем.

Во-вторых **сам динамический элемент List (или Loop) создает несколько атрибутов, отражающих положение текущего итем в списке. Мы уже рассматривали атрибут LEVEL, возникающий при древовидной сортировке, и атрибут PREV_Имя, возникающий при задании критерия группировки.**

Кроме этого, в контексте элемента списка всегда присутствует атрибут SEQ_NO — порядковый номер элемента в списке, и ODD, принимающий значение 1 если текущий элемент — нечетный и 0 в противном случае. Этот атрибут позволяет легко реализовать „полосатые" таблицы.

В случае, если список был разрезан на несколько страниц, атрибут SEQ_NO имеет значение порядкового номера на текущей странице, а атрибут TOTAL_SEQ_NO — общего порядкового номера во всем списке.

7.6.2 Loop

Блоковый вариант List. Показывает итемы из списка по шаблону, заключенному между `<:Loop:>` **и** `<:EndLoop:>`.

Все параметры эквивалентны соответствующим параметрам List, за очевидным исключением параметра *шаблон* **который у Loop отсутствует.**

7.6.3 Continuation

Формирует ссылку на продолжение или начало разрезанного на несколько страниц списка.

Синтаксис:

`Continuation` *тип-ссылки текст-ссылки* [*шаблон* [*параметры* [*атрибуты*]]]

Формирует ссылку на начало или продолжение разрезанного на страницы списка. В случае, если эта ссылка бессмыслена, т.е. показывается первая либо последняя страница списка, не выводит ничего.

Параметр *тип-ссылки* **может иметь значение** `start` **— начало списка или** `next` **— следующая страница.**

Параметр `шаблон` **позволяет указать для страницы продолжения шаблон, отличный от текущего (что имеет смысл, если в шаблоне, по которому показывается первая страница, присутствует большой объем информации, не нужный на страницах продолжения, например, текст статьи, по которой ведется дискуссия).**

Параметры `текст-ссылки`, `параметры` **и** `атрибуты` **эквивалентны соответствующим параметрам динамического элемента ItemLink (см. раздел 7.2.1).**

Динамический элемент Continuation должен быть употреблен в том же контексте, что и List или Loop которому он соответствует, и обязательно после него.

7.7 Формирование форм ввода

7.7.1 Referer

Используется в формах, action которых указывает на интерфейсный скрипт Communiware. Создает скрытые поля для передачи параметров, обязательных для данных скриптов, которые в других случаях передаются через URL динамическим элементом Script (с. 92).

7.7.2 Input

Формирует поля ввода, инициализируя их соответствующими значениями из текущего контекста.

Синтаксис:

`Input` *тип имя значение тип-специфичные параметры ...*

Динамический элемент Input позволяет создавать самые разнообразные элементы ввода. Первый его параметр — тип элемента ввода, второй — имя под которым результат ввода попадет в контекст после сабмита формы, третий — значение по умолчанию, которое следует показать в форме, если такого атрибута нет в контексте в момент показа элемента.

Дальнейшие параметры зависят от типа элемента ввода.

Input text

Создает поле ввода для строкового значения (однострочное).

Первый дополнительный параметр — число, задающее размер поля ввода в символах. Второй — набор пар имя=значение через запятую, позволяющий задать дополнительные атрибуты HTML-тэга input.

Input textarea

Создает многострочное текстовое поле ввода.

Первые два дополнительных параметра задают число строк и колонок соответственно. Третий — набор дополнительных атрибутов HTML-тэга textarea.

Input date

Создает строковое поле для ввода даты. Первый дополнительный параметр — спецификация формата по которому выводить (и вводить) дату. Использует те же спецификации формата, что и динамический элемент Attr (раздел 7.1.1) при выводе, за исключением спецификаторов, которые выдают название дня недели и месяца в текстовом виде.

Input radio

Формирует „радиокнопку" — элемент ввода, позволяющий выбрать одну позцию из нескольких. В форме должно присутствовать как минимум две радиокнопки с одинаковым именем. Поэтому данный элемент ввода обычно используется внутри списка. См. также Input radiogroup.

Параметр „значение" для радиокнопки задает не значение по умолчанию, а то значение, которое получит атрибут, если данная кнопка будет выбрана.

Остальные аргумента (кроме последнего, который может быть набором атрибутов тэга) интерпретируются как логическое выражение. Если это выражение истино, кнопка выбрана по умолчанию.

Поведение браузера в том случае если имеется несколько выбранных по умолчанию кнопок среди группы радиокнопок с одинаковыми именами, непредсказуемо, поэтому при использовании этой возможности стоит позаботиться о том, чтобы выражение могло быть истино ровно в одном из элементов списка.

Input checkbox

Формирует „чекбокс", управляющий элемент позволяющи пометить несколько вариантов. Каждый чекбокс соответствует одному из вариантов. Параметр „значение" и дополнительных аргументов интерпретируется так же, как и для радиокнопок. В отличие от радиокнопок, одиночный чекбокс — явление осмысленное. Если он помечен, в контексте будет присутствовать атрибут с его именем и значением, если не помечен, атрибут с таким именем будет отсутствовать вообще.

Если в форме было выбрано несколько чекбоксов с одинаковым именем, то соответствующий атрибут будет иметь значение списка. При подстановке через получится список значений через запятую, который можно, например, передать в фильтр, принимающий множественные параметры, или, заключив в круглые скобки, использовать как фильтр, перебирающий выбранные значения.

Так же, как у Input radio, дополнительные параметры Input checkbox представляют собой логическое выражение.

Среди дополнительных атрибутов тега у Input checkbox специальным образом обрабатывается параметр `disabled=значение`. Если значение представляет собой логическое true (не 0 и не пустая строка), то помимо генерации соответствующего атрибута тэга генерируется JavaScript, запрещающий изменение значения этого чекбокса в браузерах, не поддерживающих атрибут disabled. В случае если это значение false, то атрибут disabled из списка дополнительных атрибутов тэга удаляется.

Input radiogroup

Позволяет «одним движением» создать группу радиокнопок, значения и подписи которых создаются фильтром.

Синтаксис:

`Input radiogroup` *имя значение фильтр поле-значений поле-меток метка-по-умолчанию атрибуты-тэга*

Параметр *фильтр* **задает спецификацию фильтра. Параметры** *поле-значений* **и** *поле-меток* **указывают имена атрибутов из числа возвращаемых этим фильтром, которые следует использовать в качестве значений радиокнопок и подписей рядом с ними.**

Значение по умолчанию может не совпадать ни с одним из

возвращенных фильтром значений (например, если radiogroup используется для задания связи между двумя итемами, есть дополнительное значение „связь отсутствует"). В этом случае в качестве метки соответствующей кнопки будет использовано значение параметра *метка по умолчанию*.

Данный элемент формирует HTML-таблицу из нескольких колонок, содержащую кнопки. Количество колонок можно изменить, указав columns=n **среди прочих атрибутов тэга.**

В некоторых случаях значения и метки хочется не брать из базы, а указывать непосредственно в шаблоне. Использовать для этого перечисление значений в круглых скобках нельзя, поскольку это фильтр, который возвращает единственный атрибут ITEM_ID, а фильтр, используемый в Input radiogroup должен возвращать как минимум два атрибута — для значения и для метки. Поэтому применяется особый синтаксис:

`Input radiogroup` *имя значение* `inline` *значение метка ... атрибуты-тэга*

То есть значения и соответствующие метки указываются как отдельные параметры динамического элемента после слова `inline`.

В этом случае значение по умолчанию обязано быть среди перечисленных значений и задать метку для него отдельно нельзя.

Input checkboxgroup

Аналогичен radiogroup, но формирует чекбоксы, а не радиокнопки, т.е. позволяет множественный выбор.

Input popupmenu

Формирует выпадающее меню с одиночным выбором. Набор параметров и их интерпретация аналогичны Input radiogroup.

Input scrollinglist

Формирует прокручиваемый список с множественным выбором. Это достаточно неудобный в обращении элемент ввода, но для множественного выбора из длинных списков это часто единственно приемлемое решение. Набор параметров аналогичен Input radiogroup.

По умолчанию, формирует список такого размера, чтобы были одновременно видны все элементы. Ограничить размер можно задав параметр lines=*n* среди прочих атрибутов тэга.

Input submit

Создает кнопку, нажатие которой приводит к сабмиту формы, т.е. отправке всех введенных значений на сервер. Дополнительно позволяет запомнить некоторые из атрибутов в куки.

Синтаксис:

`Input submit` *имя значение атрибуты-тэга список-атрибутов область-действия*

Параметр „значение" интерпретируется как текст на кнопке. Параметр *атрибуты-тэга* (этот элемент ввода — единственный, у которого он не последний) задает дополнительные атрибуты тэга input.

Параметр *имя* может иметь важное значение, так как если в форме ввода несколько кнопок, определить, какая из них была нажата можно по тому, что в контексте будет присутствовать атрибут с именем и значением, соответствующими этой кнопке, а атрибуты соответствующие остальным кнопкам будут отсутствовать.

В списке атрибута тэгов особым образом интерпретируется атрибут confirm. Если он присутствует, то в выходной HTML помещается JavaScript вызывающий появление popup-окошка с сообщением, заданным значением этого атрибута, позволяющего отменить ошибочное нажатие кнопки.

Параметр *список-атрибутов* задает список атрибутов, которые должны быть запомнены в куках. Параметр *область-действия* задает множество страниц, на которые пользовательский браузер будет посылать эти куки. Он может принимать значения:

server — все страницы с текущим hostname, включая интерфейсные скрипты, другие сайты с тем же hostname и статические страницы.

site — текущий сайт. Все страницы начинающиеся с текущего URL_PREFIX.

item — текущий итем, независимо от того, по какому шаблону он показан.

exact — текущий итем, показанный по текущему шаблону.

Input reset

Создает кнопку reset, сбрасывающую значения всех полей формы в те, которые они имели до начала редактирования. Атрибут „Значение" интерпретируется как текст кнопки.

Input hidden

Формирует скрытое поле, которое создает после сабмита формы то значение, которое было указано в качестве параметра „значение".

Часто используется в сочетании с набором чекбоксов, поскольку если не один из чекбоксов с некоторым именем не помечен, то атрибут с этим именем вообще не создается. Поэтому, если необходимо, чтобы в этом случае создавался атрибут с пустым значением, нужно разместить в той же форме скрытое поле с пустым значением.

7.7.3 Login

Формирует поле для ввода пароля, проверяет его и выставляет куку AUTHOR_ID.

Синтаксис

<:Login *имя=значение* ...:>

Формирует поле для ввода пароля. Все дополнительные парамет
должны иметь вид *имя=значение* и рассматриваются как атрибуты
генерируемого тега input.

Динамический элемент следует использовать внутри формы,
использующей метод post (иначе пароль будет виден в URL.
В этом случае авторизация по такому паролю будет отвергнута).

Форма может иметь ACTION, указывающий на другую страницу
Communiware, так как проверка запроса реально производится
ядром до начала обработки шаблона.

В случае ошибки авторизации, сообщение об ошибке помещается
в атрибут ERROR.

7.7.4 Subscribe

Модифицирует состояние подписки на обновление.

Синтаксис:

Subscribe *итем шаблон* [*период* [*format* [*адресаты* [*атрибуты-тега*]]]]

Управляет подпиской на обновления. Если не указаны *период* и *формат* то генерирует выпадающие меню, позволяющие пользователю выбрать желаемое значение. Если они указаны, то формирует скрытые поля, вызывающие соответствующие действия при нажатии кнопки сабмит без возможности пользователя повлиять на них.

Допустимые значения для периода: NEVER (отказ от подписки), HOURLY, DAILY, WEEKLY и MONTHLY.

Допустимые значения для формата: TEXT/PLAIN и TEXT/HTML.

По умолчанию управляет подпиской текущего пользователя. Если указан параметр *адресаты*, который должен представлять собой спецификацию фильтра, то позволяет управлять подпиской пользователей, отличных от текущего. Параметр *атрибуты-тэга* --- список пар имя=значение, который будет использован в качестве атрибутов создаваемых тегов select (для выбора формата и периода).

Данный динамический элемент может быть использован как внутри обычной формы (для подписки на существующий итем), так и в форме постинга, для подписки на вновь создаваемый/моди итем.

7.7.5 MailTo

Отправляет текущий итем по почте.

Синтаксис:

MailTo *шаблон список-пользователей*

Посылает текущий итем, отформатированный по указанному шаблону, указанным пользователям. Параметр *список-пользователей* должен представлять собой спецификацию фильтра.

Если этот динамческий элемент указан в обычной форме, отправка письма происходит в момент генерации страницы, выдаваемой в ответ на нажатие кнопки сабмит в форме. Если он указан в форме постинга --- в момент обработки постинга,

в самом ее конце. При этом текущим считается итем, созданный или отредактированный в результате постинга.

В случае, если форма постинга производит удаление итема, письмо будет сформировано до удаления, но отправлено только в случае успешного удаления.

7.8 Формы постинга

7.8.1 Post

Формирует форму постинга (создания или редактирвания итем).

Синтаксис

<:Post *спецификация авторизация возврат параметры*:>
фрагмент шаблона
<:EndPost:>

Создает форму постинга, позволяющую создать, отредактироват или удалить итем.

Если параметр *спецификация* не пуст, он должен представлять собой либо идентификатор итем (тогда это редактирование), либо конструкцию вида new(*идентификатор типа*) (создание нового итем).

Если этот параметр пуст, то все поля ввода внутри формы постинга должны содержаться внутри динамических элементов Item. Такая конструкция позволяет в одной форме создать/отредактировать несколько итем.

Параметр *авторизация* может быть пустым, тогда воспользоватьс этой формой постинга могут только пользователи, имеющие права модерирования данного итем, либо иметь значение registe тогда воспользоваться этой формой имеет право любой зарегистри пользователь, либо значение anon позволяющее анонимному пользователю воспользоваться данной формой.

Рекомендуется ограничивать доступ к шаблону, содержащему форму постинга при помощи динамического элемента Authorize или Authentify.

Параметр *возврат* может быть полной URL, начиная с http://, ссылкой на итем на текущем сайте (идентификатор/шаблон) или принимать специальные значения new --- перейти на созданны редактированный итем, или back --- вернуться на страницу,

передыдущую по отношению к той, на которой была показана форма постинга. Если этот параметр пуст, то после успешного постинга будет опять показана страница с формой. В случае ошибки постинга всегда повторно показывается страница с формой и в атрибуте контекста ERROR находится сообщение об ошибке.

7.8.2 Item

Формирует фрагмент формы постинга, ответственный за конкретный итем.

Синтаксис:

<:Item *спецификация имя условие*:>
фрагмент шаблона
<:EndItem:>

Создает контейнер, внутри которого размещаются поля ввода, относящиеся к одному итему в форме, позволяющей создать/отредактировать много итем.

Параметр *спецификация* аналогичен такому же параметру у Post, с той разницей, что он не может быть пустым.

Параметр *имя* задает имя, которое позволяет ссылаться на значения введенные в этом контейнере из соседних контейнеро Если он опущен, то контейнеру будет присвоено некоторое автоматически сгенерированное имя, но ссылаться на него из соседних контейнеров будет невозможно.

Параметр (вернее, группа параметров) *Условие* задает логическ выражение. Если это логическое выражение истино, то данный контейнер обрабатывается. Если оно ложно --- он игнорируется.

В случае отсутствия условия контейнер обрабатывается всегда.

7.8.3 EditField

Формирует поле для ввода/редактирования атрибута или списка связанных итем.

Синтаксис

EditField *имя значение-по-умолчанию параметры*

Динамический элемент EditField формирует ,,естественное'' поле редактирования в зависимости от типа редактируемого

атрибута.

Для полей типа NUMBER, STRING и DATE это строка ввода, для полей типа RICHTEXT --- многострочное поле ввода, либо строка, для полей имеющих список допустимых значений --- выпадающее меню.

В качестве имени можно указывать не только имя атрибута итем, но и идентификатор типа связи (тогда будет сформировано поле ввода, куда нужно вводить список идентификаторов итем, которые должны быть связаны с данным), TEXT (тогда будет сформировано поле для ввода текста), UPLOAD (поле для загрузки файла в текст итем), PICn (поле для загрузки картинок) и ряд других специальных атрибутов влияющих на обработку постинга. См. главу 3.

Значение-по-умолчанию используется только при создании нового итем.

Все остальные параметры имеют вид имя=значение, и задают атрибуты html-тэга, созданного данным динамическим элементом. При вводе атрибута типа RICHTEXT особое значение имеет параметр size. Если он указан, то создается однострочное поле ввода (input type=text). Если не указан --- создается многострочное поле (textarea).

Для атрибута PASSWD (пароль, расширенный атрибут типа AUTHOR), создается два поля ввода, в которые должны быть введены одинаковые значения. Это необходимо для проверки корректности ввода, поскольку пароль при вводе не отображается на экране. Между двумя полями ввода выводится фрагмент HTML который можно указать в параметре delimiter.

Наиболее гибким и сложным является EditField TEXT. Для итемов с Content-Type Html (наиболее распространенных в Communiware), он поддерживает три основных режима редактирован

В виде исходного Html-текста (representation=HTML), преобра в текстовый формат, в котором поддерживаются несколько видов простых выделений (жирность, подчеркивание, списки) и работа с гиперссылками (representation=Text), а также (при поддержке этого пользовательским браузером) с помощью WYSIWYG-редактора.

WYSIWYG-режим имеет приоритет над заданным параметром representation, поэтому если WYSIWYG

не отключен разработчиком шаблона явно, с помощью параметра wysiwyg=no, пользователь со включенным WYSIWYG режимом будет всегда получать именно его.

По умолчанию (если параметр representation не указан) используется режим representation=auto, т.е. если текст итема не содержит разметки, которая не может быть преобразован в формат текста с разметкой без потерь, показывается текст, а в противном случае --- HTML-исходник.

Поскольку формат текста с разметкой требует определенных пояснений, а процедура включения/выключения WYSIWYG-режима выдвигает определенные требования к конфигурации пользовательс компьютера, ниже поля ввода выдаются сообщения, содержащие ссылки на документацию по тому и по другому.

Эти сообщения представляют собой фрагменты HTML имеющие класс editfieldwarn, что позволяет разработчику шаблона выполнить их стилевое оформление.

Подавить ссылку на документацию по формату разметки можно с помощью параметра help=no. Ссылка на страницу включения/выкл WYSIWYG исчезает, если использование WYSIWYG в данной форме запрещено.

Ссылки в этих сообщениях ведут на итемы edit_form, dhtmled и plugin_setup служебного сайта Communiware, но в контексте текущего сайта.

7.8.4 Use

Вычисляет значение, которое следует присвоить атрибуту или связи в зависимости от значений, введенных в другие поля формы.

Синтаксис:

Use имя выражение [ifabsent | ifempty]

Вычисляет значение выражения и присваивает его указанному атрибуту. В отличие от динамического элемента Define, вычислен производится не в момент показа страницы, а в момент обработки результатов постинга.

В операндах *выражения* можно использовать подстановку времени сабмита, в том числе и ссылки на атрибуты соседних контейнеров Item.

Использование нескольких динамических элементов Use с одинаковым именем приводит к порождению значения типа список.

Параметры ifabsent и ifempty имеют тот же смысл, что и в динамическом элементе Define (с. 102).

7.8.5 Check

Проверяет что введенные данные удовлетворяют определенному условию.

Синтаксис:

Check *действие сообщение условие*

Вычисляет *условие* во время обработки результатов постинга, и если оно выполняется, производит указанное *действие*.

Действие может быть fail, при этом весь постинг будет отменен, форма показана снова и в атрибут контекста ERROR помещено указанное сообщение.

Действие warn приводит к помещению сообщения в атрибут контекста WARNING. Постинг при этом успешно завершается и спецификация возврата в динамическом элементе Post отрабатыв Поэтому в шаблонах, куда может быть передано управление после завершения постинга, содержащего динамический элемент Check, следует предусмотреть обработку этого атрибута.

В случае если в форме указано несколько динамических элементов Check fail, проверены они будут все, и в атрибут ERROR помещены все сообщения для которых выполняется условие.

То же самое касается нескольких Check warn.

Глава 8

Список стандартных фильтров

Набор фильтров на конкретном Communiware-сервере --- вещь переменная. Новые perl-фильтры (как впрочем и новые динамичес элементы) могут устанавливаться в составе пакетов-расширений Communiware.

Новые SQL-фильтры вообще могут создаваться разработчиками сайтов через web-интерфейс.

В данной главе мы опишем только те фильтры, которые входят в стандартный дистрибутив Communiware.

8.1 Perl-фильтры

8.1.1 ::Calendar

Синтаксис:

::Calendar([*дата*])

Возвращает список дней в текущем месяце. Если указан параметр, этот параметр должен представлять собой дату в стандартном формате Communiware. В этом случае будет возвращен список дней, содержащих эту дату.

Контексты, возвращаемые данным фильтром содержат следующие атрибуты:

DATE начало (0 часов) указанного дня в формате даты Communiwa

DAY порядковый номер дня в месяце в виде числа

DOW порядковый номер дня недели (1 --- понедельник, 7 --- воскресенье).

LASTDAY номер последнего дня в месяце. Проверка

```
<:If LASTDAY = @DAY:>
```

позволяет определить, что текущий элемент списка является последним.

MONTH номер месяца

YEAR четырехзначный номер года.

8.1.2 ::Mail

Используется шаблонами подписки. Единственное назначение данного фильтра, избежать повторного выполнения фильтра подписки и повторных проверок прав доступа. Никаких параметров не имеет, возвращает список итем, уже извлеченных cmwmaild в процессе анализа, есть ли новости, которые необходимо рассылать по данной подписке.

8.1.3 ::MnogoSearch

Обеспечивает полнотекстовый поиск. Работает в том случае, если на сервере функционирует индексатор mnogosearch (входящий в поставку Communiware).

Для работы данного фильтра требуется наличие в контексте атрибута SEARCHFOR (который обычно создается с помощью формы, в которой пользователь задает слова для поиска).

Параметры данного фильтра задаются в форме ИМЯ=значение. Некоторые параметры не требуют значения.

Определены следующие параметры:

FILTER=filterspec Производить поиск только среди итемов, возвращ данным SQL-фильтром.

LANG=language Производить поиск только среди итемов имеющих данный язык. По умолчанию, совпадает с языком текущего итема.

NOPARENTS По умолчанию, фильтр MnogoSearch выдает результаты пригодные для древовидной сортировки. В связи с этим ему может потребоваться извлечь итем, не содержащий искомого слова, но, например, являющийся рубрикой, содержащей такие итемы.

Указание данного параметра запрещает такое поведение.

PARENT=имя связи заставляет использовать для связывания с предками указанную связь, вместо связи TOPIC, используем по умолчанию.

QUERY=sql-statement Аналогична по смыслу опции FILTER, но позволяет явно задать SQL-оператор среди результатов которого будет проводиться полнотекстовый поиск. Маловеро что использование этой опции в шаблоне позволит добиться сколько-нибудь осмысленных результатов, так как практичес любой SQL-запрос будет распарсен как несколько отдельных параметров, и ничего кроме синтаксической ошибки не вызовет. Данная опция предназначена для обращения к фильтру MnogoSearch из других перловых фильтров.

В процессе работы фильтр генерирует некоторую информацию, которую помещает в контекст как значения атрибутов. В существу реализации эти атрибуты доступны только ниже по тексту шаблона.

ERROR Текст сообщения об ошибке, если таковая возникла в процессе выполнения фильтра.

ITEMS_FOUND количество найденых item.

В контекст каждого из возвращенных итемов кроме того помещается числовой атрибут WEIGHT, указывающий приблизительну степень соответствия итема поисковому запросу.

8.1.4 ::Pic

Возвращает список картинок, принадлежащих текущему итему.

Синтаксис:

::Pic
::Pic(sizes)
::Pic(check)
::Pic(*pattern*)
::Pic(*pattern*,sizes)
::Pic(*pattern*,check)

По умолчанию возвращает список всех картинок, принадлежащих текущему итему. Если указан шаблон, синтаксис которого соответствует требованиям shell, то возвращаются только картинки, соответствующие этому шаблону.

Контекст, возвращаемый фильтром, состоит из следующих атрибутов:

PICTURE имя файла картинки

BASENAME имя файла картинки без расширения

Если указана опция sizes, то в контекст добавляются следующие атрибуты:

FILESIZE размер файла в байтах

WIDTH ширина картинки в пикселах

HEIGHT высота картинки в пикселах

В настоящий момент WIDTH и HEIGHT могут быть вычислены только для файлов в форматах GIF, JPEG и PNG. Для файлов shockwave flash эти атрибуты будут не определены.

Поскольку оператор условия *значение* IN *filter* проверяет значение на совпадение с атрибутом ITEM_ID контекста, возвраща фильтром, при использовании фильтра ::Pic в логических выражениях следует указывать опцию check. При указании этой опции имя файла возвращается как ITEM_ID, а не как PICTURE.

8.1.5 ::SiteReport

Подсчитывает активность коммьюнити --- количество и объем материалов, опубликованных за указанный период и количество активных авторов.

Синтаксис:

::SiteReport *дата-начала*,*дата-конца*, *список-типов*)

Возвращает по контексту за каждый месяц, попадающий в интервал от начальной до конечной даты.

В контекст помещаются следующие атрибуты:

AUTHORS Количество авторов, опубликовавших за данный месяц хотя бы одну работу

BOOK_I_N Количество книг, опубликованных за этот месяц

BOOK_V_N Суммарный объем книг (а также статей и глав, вошедших в ранее опубликовынные книги)

ARTICLE_I_N Количество статей

ARTICLE_V_N суммарный объем статей

EXTERNAL_I_N количество внешних ссылок

BINARY_I_N количество файлов (итемов типа файл)

BINARY_V_N Суммарный объем файлов

Кроме этого имеется ,,расширенный режим'' работы данного фильтра, который позволяет произвести большее количество разнообразных настроек. Этот режим отличается тем, что параметры задаются в виде имя=значение и подробно описан во встроенной документации на данный фильтр.

8.2 SQL-фильтры

AllowedTypedTree Зависит от атрибута CURRENT_SERVER, получает один позиционный параметр --- имя типа связи.

Возвращает список всех итемов, которые могут участвовать в этой связи как активные, в виде пригодном для древовидно сортировки.

ActiveLinkedOf Список непосредственных предков итема, указанного как первый аргумент по связи, указанной как второй аргумент. Не зависит от контекста.

ActiveLinked Список непосредственных предков текущего итема по связи, заданной как единственный аргумент.

ActiveLinkTypes Список всех связей, в которых данный тип итема участвует как пассивный. Возвращаемые контексты являются контекстами текущего итема, содержащими атрибуты LINKTYPE_ID и LINK_NAME (название связи на текущем языке).

ActiveTyped Список непосредственных предков текущего итема по указанной связи (первый аргумент), имеющих заданный тип (второй аргумент).

ActiveTypedSt Список непосредственных предков текущего итема по указанной связи (первый аргумент), имеющих заданный тип (второй аргумент) и статус (третий аргумент)

AllActiveLinked Список всех предков текущего итема по указанной (как аргумент) иерархической или мета- связи.

AllActiveTyped Список всех предков текущего итема по указанной (первый аргумент) связи, имеющих указанный (второй аргумент) тип.

allauthors Список всех персон, имеющих связь с каким-либо итемом текущего сервера, в которой персона выступает в роли активной стороны.

All_Items Возвращает все итемы текущего сервера с типом, указанным в качестве аргумента.

AllLinksActive Выбор всех итемов, связанных с текущим любыми связями, в которых текущий итем пассивный. Возвращает атрибуты LINKTYPE_ID и LINK_NAME

AllLinksPassive Выбор всех итемов, связанных с текущим любыми связями, в которых текущий итем активный. Возвращает атрибуты LINKTYPE_ID и LINK_NAME

All_MultiTyped Выбор всех итем текущего сайта с типом из заданного списка (до 4).

All_MultiTypedTree Выбор всех итем текущего сайта с указанием предка по указанной связи (первый аргумент) как PARENT_ID, имеющих один из перечисленных (до 6) типов

AllOpenLinked Выбор всех ближайших потомков текущего итема, принадлежащих к текущему сайту, и связанных с текущим указанной связью, принадлежащих к указанному списку типов (до 4) и не имеющих ограничений на чтение

AllOpen_MultiTyped Выбор всех итемов текущего сервера, принадлеж к заданному (до 6) списку типов, не имеющих ограничений доступа на чтение.

AllOpenTyped Все итемы текущего сайта с заданным (как аргумент) типом, не имеющие ограничений доступа.

AllowedLinks Список всех связей, в которых может участвовать итем текущего типа. В качестве аргумента получает строку ACTIVE или PASSIVE

AllowedPageTemplates Список всех шаблонов, допустимых для данного итема в качестве шаблона по умолчанию

AllowedTyped Дерево итемов по указанной (как первый аргумент) связи, которые могут участвовать в этой связи в указанной роли (второй аргумент)

AllPassiveLinkedOf Все потомки указанного итема по заданной иерархической связи или метасвязи.

AllPassiveLinked Все потомки текущего итема по заданной иерархиче связи или метасвязи.

AllPassiveLinkedSt Все потомки текущего итема по заданной иерархической или мета- связи, имеющие заданный статус

AllPassiveMultiTyped Все потомки текущего итема по заданной иерархической или мета-связи, относящиеся к одному из списка заданных типов (до 4) .

AllPassiveTyped Выбирает всех ближайших потомков текущего итема по связи, заданной в качестве параметра с типом, заданым в качестве второго параметра.

AllPassiveTypedSt Выбирает всех потомков текущего итема по связи, заданной в качестве параметра с типом, заданым в качестве второго параметра и статусом, заданным третьим параметром.

all_servers Возвращает список всех сайтов на текущем физическом сервере, добавляет в контекст атрибуты LASTUPDATED (максимальное значение UPDATED среди итемов данного сайта) и COUNT_N (общее количество итем данного сайта)

all_threads Список нитей, возникших в дискуссии по данному итему или в дискуссиях по итему данной рубрики.

AllWithStatus Все item заданного типа на текущем сайте, имеющие заданный набор статусов. Первый параметр - тип итема, далее - несколько значений статуса (до 4

Authorized Список персон, имеющих авторизационную связь с текущим item, отсортированный по TITLE

auth Пользователи, имеющие указанные (до 4) связи с итемами текущего сервера

coauthors Получение списка соавторов данного автора в данной работе. Предполагает, что текущий контекст --- контекст работы, а вышележащий --- контекст автора. Выбирает всех авторов данной работы, кроме того, ID которого равен ITEM_ID#1. Предназначен для использования в списке работ на странице автора.

current_subscr Возвращает список всех подписок на текущий итем.

DefaultTemplate Возвращает стандартный шаблон для указанного типа в текущей рубрике

discussedin Список всех дискуссий, в которых есть реплики данного автора. Возвращает атрибуты LASTREPLIC_D ---

дата публикации последней реплики и REPLICS_N --- количес
реплик данного автора в данной дискуссии.

ExtendedAttr Возвращает список расширенных атрибутов текущего итема, имеющих заданный тип данных (STRING, RICHTEXT etc). Используется в шаблоне index_t, позволяя проиндексир все расширенные атрибуты типа RICHTEXT любых типов итем.

FilterList Список фильтров, используемых в данном шаблоне

GetHelp Список итемов указанного (первый аргумент) типа, у которых поле params равно второму аргументу

LinkedItemTypes Все типы итем, использование которых разрешено на данном сайте.

LinkedTree Дерево потомков по заданному типу иерархической связи, готовое для иерархической сортировки по ordnum

LinkedTreeUp Все предки итема по заданному типу иерархической связи, готовые для древовидной сортировки

LinkHierarchy Возвращает признак иерархичности указанной связи

LostItems Выбор всех ,,потерянных'' итемов, т.е. таких, которые должны иметь предка по одной из связей, входящих в метасвязь BELONGS, но не имеющих таковых

Moderators Все суперпользователи и все модераторы сайта, которому принадложит текущий item.

MultiTypeName Получает в качестве параметров список типов итем и возвращает пары TYPE_ID, интернационализированное название.

NextSibling Ближайший сосед (потомок того же предка) по указанно типу связи, имеющий ORDNUM больше текущего.

PassiveLinked Выбор всех ближайших потомков по указанному типу (не мета) связи

PassiveLinkedVirtual Вытаскивает все итемы, являющиеся потомками заданного виртуального итема на текущем сайте по заданной связи.

PassiveLinkTypes Список всех связей, в которых данный тип итема участвует как пассивный. Возвращаемые контексты являются контекстами текущего итема, содержащими атрибуты LINKTYPE_ID и LINK_NAME (название связи на текущем языке).

PassiveMTypedSite Выборка всех потомков данного итема с заданным типом связей, имеющих перечисленные (до 4) типы итема, принадлежащих текущему сайту

PassiveMultiTyped Выборка всех потомков данного итема с заданным типом связей, имеющих перечисленные (до 4) типы итема

PassiveTyped Список непосредственных потомков текущего итема по заданному типу связи, имеющих заданный тип.

PassiveTypedSt Список непосредственных потомков текущего итема по заданному типу связи, имеющих заданный тип и статус.

PrevSibling Ближайший сосед (потомок того же предка) по указанно типу связи, имеющий ORDNUM меньше текущего.

RecentItems Выбор всех итемов с указаным типом на текущем сайте за указанное число дней

SearchUsers Поиск персоны, имеющей отношение к текущему сайту (т.е. принадлежащие ему, или имеющие связь с каким-то его итемом, у которых в ITEM_ID или TITLE имеется указанна подстрока (заданная в атрибуте SEARCHUSER).

server_subscriptions Выбор всех пользователей, подписавшихся хоть на что-то на текущем сайте

Statuses Список всех статусов указанного типа

StatusMultiTyped Все потомки текущего итема по указанной связи с указанным статусом и нескольки аи типами.

StLessMultiTyped Все потомки текущего итема по указанной связи, со статусом меньше указанного вторым аргументом, с несколькими возможными типами.

StRangeMultiTyped Все потомки текущего итема по указанной связи, статус которых попадает в диапазон от второго до третьего аргумента с несколькими возможными типами (до 4)

subscr_children Фильтр подписки. Выбирает всех потомков по указанной связи, опубликованных в течении периода.

Subscriptions Выбор всех подписок, разрешенных для текущего итема, принадлежащих к одному или двум (а больше их не бывает) типам подписки.

subscr_new Фильтр подписки. Выбирает всех потомков кроме реплик, по указанной связи, у которых дата публикации попадает в период

subscr_rubr_disc Фильтр подписки. Выбирает все реплики во всех дискуссиях рубрики, опубликованные в течении периода

subscr_status Фильтр подписки. Выбирает все итемы в подрубриках текущей рубрики, у которых статус менялся в течении периода.

SubsItemTypes Возвращает список типов итем с непустым атрибутом CHECKED если текущий итем может выступать в качестве шаблона по умолчанию или шаблона подписки для данного типа

TemplateOfType Выбирает на текущем сайте все шаблоны с у которых атрибут template_type равен заданному параметру.

ThisServer Выбор всех итем текущего сайта, независимо от типа опубликованных за последние n дней, где n-параметр

threadlist Список всех нитей в дискуссиях текущего сайта. Добавляет в контекст атрибут LASTREPLIC (дата последней реплики в нити) и NREPLICS (число реплик в нити

TitleSearch Поиск подстроки (содержащейся в атрибуте текущего контекста SEARCHFOR) в аннотациях, заголовках, именах и id авторов текущего сайта

TypeAttrs Список редактируемых атрибутов текущего типа итема Имя атрибута извлекается как ITEM_ID, краткое интернациона название как TITLE, и длинное (если есть) как ABSTRACT

TypedSearch Поиск подстроки (содержащейся в атрибуте контекста SEARCHFOR) в id, аннотациях и заголовках по всем итемам текущего сайта, имеющим заданный тип.

TypedTree Дерево потомков по заданному типу связи, имеющих заданный тип, готовое для древовидной сортировки. Не работает для метасвязей.

TypedTreeUp Дерево предков по заданному типу связи, имеющих заданный тип. Не работает для метасвязей.

UsingMe Список всех итемов, использующих текущий в качестве шаблона.