Challenger1.3 - User

INTRODUCTION

The User's manual is intended for anyone who publishes web pages through a Roxen Challenger server. It describes what functionality Challenger provides that can be used to make it easier to create static web pages as well as dynamic content.

It is assumed that the reader is familiar with HTML. Most of Challengers functions are available as RXML tags, that will be easy to learn for anyone who knows HTML.

Introduction
This chapter, introducting the concepts and RXML.

Information tags
Simple RXML tags that provide information about the client or server.

String tags
Container tags that process input.

Variable tags
Tags that handle form variables.

URL tags
Tags that handle properties of URLs and HTTP like prestates, cookies and authorization.

If tags
Tags that make it possible to conditionally show different content.

Graphics tags
Tags that draw and manipulate graphical images.

Database tags
Tags that communicate with SQL databases.

Programming tags
Miscellaneous tags that are useful for programmers and advanced RXML users.

Supports System
Supports system that makes it possible to create pages that are optimized for any browser.

SSI
SSI, Server Side Include, tags that make Challenger compatible with the NCSA / Apache web servers.

.htaccess security
How to limit who can view your pages.

Image Maps
How to handle server side image maps.

Appendix
Lists of all RXML tags, different content types and supports.

Concepts

RXML

Content on the Web is written using HTML, that is a text format with markup in the form of `<<tags>>`. Everything the browser can display is controlled by different HTML tags. Challenger comes with it's own macro-language, RXML, that uses tags like the ones in HTML. But RXML is never sent to the browser, Challenger converts all RXML tags into HTML by use of the RXML parser.

RXML can be used for a number of things, creating graphical headers and diagrams, connecting to databases or creating pages that will work on any browser. The bulk of this manual describes the various RXML tags and how they can be combined. The key to RXML is that each tag solves a separate task. Hence several tags can be combined to perform an even greater task.

Modules

Roxen Challenger is written using a module system. The different functions of Challenger is handled by different modules. Modules are enabled and configured through the configuration interface by the administrator. Some modules handle different RXML tags. Therefore, which RXML tags you can use do depend on how the administrator has configured Challenger. The documentation for each tag includes information about which module handles it, ask your administrator to enable it, if you need the tag.

Modules that handle RXML tags can be written by third-party developers or programmers within your organization. It is also possible to create packages of RXML tags, for use with the `<use>` tag. Apart from learning the RXML tags in this manual you should learn the special tags available at your site.

Content Types

Each file fetched through a web server contains a MIME content type that identifies what type of file it is. Thus an HTML file has the content type `text/html`, while a GIF image has the content type `image/gif`.

On a Challenger server, the file extension determines the content type of that file. Usually `.html` or `.htm` files are given the content type `text/html` while `.gif` files are given the content type `image/gif`.

As a user, you usually don't have to bother with content types. If you just give your files their standard extensions everything will work. But sometimes, when you try out new plugins that use their own file format, the extension and content type that you want to use is not handled by the server. Then the administrator for the server has to change the configurations for the *Content types* module.

Some extensions might be handled by the web server itself. The most common use is to run files through the *Main RXML parser* module. This makes it possible to use RXML tags on such pages. Depending on the policy of your site this might be done for all `.html` files, or only for special `.rxml` files.

A list of the most common file extensions and content types can be found in the appendix.

RXML

RXML, RoXen Macro Language, is a language handled by the Challenger web server. RXML will always be translated to HTML by the server, before it is sent to the browser. The RXML tags are divided into the following different categories:

Information tags
Information tags are simple tags that provide information about the client, the server or the date.

String tags
String tags are container tags that transform some input into HTML. The input differs, some tags use HTML while other use tab separated text.

Variable tags
Variable tags are tags that handle form variables as well as the various variable types internal to Challenger. With variable tags it is also possible to define your own RXML tags.

URL tags
Tags that handle properties of URLs and HTTP like prestates, cookies and authorization.

If Tags
If tags handle conditional showing of different content. They make it possible to optimize the pages for all browsers as well as making advanced dynamic content.

Graphic tags
Graphic tags create and manipulate images. They can create graphical headers, real-time diagrams as well as animated clocks.

Database tags
Database tags communicate with SQL databases and makes it easy to incorporate data from those databases into RXML pages. It is possible to connect to any number of databases.

Programming tags
Programming tags are useful for doing advanced RXML as well as for debugging Challenger modules. It is also possible to run Pike code within your RXML pages.

INFORMATION TAGS

Information tags are simple tags that provide information about the client, the server or some external event. Examples are the `<accessed>` tag, that counts accesses to the page and the `<modified>` tag which shows when the page was last updated.

<accept-language>

`<accept-language>` is defined in the *Main RXML parser* module.

Returns the language code of the language the user prefers, as specified by the first language in the accept-language header.

If no accept-language is sent by the users browser `None` will be returned.

full
Returns all languages the user has specified, as a comma separated list.

Example

```
Your preferred language is
<accept-language>
```

Results in

```
Your preferred language is en
```

<accessed>

`<accessed>` is defined in the *Main RXML parser* module.

`<accessed>` generates an access counter that shows how many times the page has been accessed. In combination with the `<gtext>`tag you can generate one of those popular graphical counters.

A file, `AccessedDB`, in the logs directory is used to store the number of accesses to each page. Thus it will use more resources than most other tags and can therefore be deactivated in the *RXML parser* module. By default the access count is only kept for files that actually contain an `<accessed>` tag, but that can also be configured.

add=*number*
Increments the number of accesses with this number instead of one, each time the page is accessed.

addreal
Prints the real number of accesses as an HTML comment. Useful if you use the *cheat* attribute and still want to keep track of the real number of accesses.

capitalize
Capitalizes the first letter of the result.

cheat=*number*
Adds this number of accesses to the actual number of accesses before printing the result. If your page has been accessed 72 times and you add `<accessed cheat=100>` the result will be 172.

factor=*percent*
Multiplies the actual number of accesses by the factor.

file=*filename*
Shows the number of times the page `filename` has been accessed instead of how many times the current page has been accessed. If the filename does not begin with "/", it is assumed to be a URL relative to the directory containing the page with the `<accessed>` tag. Note, that you have to type in the full name of the file. If there is a file named tmp/index.html, you cannot shorten the name to tmp/, even if you've set Challenger up to use index.html as a default page. The `filename` refers to the **virtual** filesystem.

One limitation is that you cannot reference a file that does not have its own `<accessed>` tag. You can use `<accessed silent>` on a page if you want it to be possible to count accesses to it, but don't want an access counter to show on the page itself.

lang=ca es_CA hr cs nl en fi fr de hu it jp mi no pt ru sr si es sv
Will print the result as words in the chosen language if used together with *type=string*. Available languages are ca, es_CA (Catala), hr (Croatian), cs (Czech), nl (Dutch), en (English), fi (Finnish), fr (French), de (German), hu (Hungarian), it (Italian), jp (Japanese), mi (Maori), no (Norwegian), pt (Portuguese), ru (Russian), sr (Serbian), si (Slovenian), es (Spanish) and sv (Swedish).

lower
Prints the result in lowercase.

per=second minute hour day week month
Shows the number of accesses per unit of time.

prec=*number*
Rounds the number of accesses to this number of significant digits. If *prec*=2 show 12000 instead of 12148.

reset
Resets the counter. This should probably only be done under very special conditions, maybe within an `<if>` statement.

This can be used together with the file argument, but it is limited to files in the current- and sub-directories.

silent
Print nothing. The access count will be updated but not printed. This option is useful because the access count is normally only kept for pages with actual `<accesss>` on them. `<accessed file=filename>` can then be used to get the access count for the page with the silent counter.

upper
Print the result in uppercase.

since
Inserts the date that the access count started. The language will depend on the *lang* tag, default is English. All normal date related attributes can be used. See the `<date>` tag.

type=number string roman iso discordian stardate
Specifies how the count are to be presented. Some of these are only useful together with the *since* attribute.

Example

```
This page has been accessed
<accessed type=string cheat=90 addreal>
times since <accessed since>.
```

Results in

This page has been accessed ninetytwotimes since today, 03:02.

<clientname>

`<clientname>` is defined in the *Main RXML parser* module.

Prints the name of the browser the user is using.

full
Returns the full name of the browser.

Example

```
Your browser idientifies itself as
<clientname>.
```

Results in

Your browser idientifies itself as Mozilla/4.51.

<configurl>

`<configurl>` is defined in the *Main RXML parser* module.

Prints an URL to the configuration interface for this Challenger server.

Example

```
<a href='<configurl>'>
Link to the configuration interface</a>
```

Results in

Link to the configuration interface

<configimage>

`<configimage>` is defined in the *Main RXML parser* module.

Inserts an image used by the configuration interface.

src=back err_1 err_2 err_3 fold fold2 help ihfc manual-note manual-tip manual-warning pike power roxen unfold unfold2 unit
Specifies which image to use.

All other attributes are sent through to the generated `<img>` tag.

Example

```
<configimage src=fold>
```

Results in

▽

<countdown>

<countdown> is defined in the *Countdown* module.

This tag counts the time to or from a specified date.

Time related attributes

day=*number, weekday*
Sets the weekday.

hour=*number*
Sets the hour.

iso=*year-month-day*
Sets the year, month and day, all at once.

mday=*number*
Sets the day of month.

min=*number*
Sets the minute.

month=*number, month*
Sets the month.

sec=*number*
Sets the second.

year=*number*
Sets the year.

Presentation related attributes

combined
Shows an English text describing the time period. Example: 2 days, 1 hour and 5 seconds. You may use the *prec* attribute to limit how precise the description should be. You can also use the *month* attribute if you want to see years/months/days instead of years/weeks/days.

days
Prints the number of days until the time.

dogyears
Prints the number of dog years until the time, with one decimal.

hours
Prints the number of hours until the time.

lang=ca es_CA hr cs nl en fi fr de hu it jp mi no pt ru sr si es sv
Will print the result as words in the chosen language if used together with *type=string*. Available languages are ca, es_CA (Catalan), hr (Croatian), cs (Czech), nl (Dutch), en (English), fi (Finnish), fr (French), de (German), hu (Hungarian), it (Italian), jp (Japanese), mi (Maori), no (Norwegian), pt (Portuguese), ru (Russian), sr (Serbian), si (Slovenian), es (Spanish) and sv (Swedish).

minutes
Prints the number of minutes until the time.

months
Prints the number of month until the time.

nowp
Returns 1 if the specified time is now, otherwise 0. How precise now should be interpreted is defined by the *prec* attributes. The default precision is one day.

prec=year month week day hour minute second
A modifier for the *nowp* and *combined* attributes. Sets the precision for these attributes.

seconds
Prints how many seconds until the time.

since
Counts from a time rather than towards it.

type=string number ordered
How to present the result.

weeks
Prints the number of weeks until the time.

when
Prints when the time will occur. All valid <date> tag attributes can be used.

years
Prints the number of years until the time.

Example

```
<p>I am <countdown iso=1980-06-28
since years type=string> years old.</p>

<p>There are <countdown year2000 days>
days left until year 2000.</p>

<p>Is this a Sunday?
<br><if eval='<countdown day=sunday nowp>'>
Yes, this is a Sunday.</if>
<else>No, it isn´t.</else>
```

Results in

```
I am eighteen years old.

There are 215 days left until year 2000.

Is this a Sunday?
No, it isn´t.
```

<date>

`<date>` is defined in the *Main RXML parser* module.

This tag prints the date and time.

brief
Generates as brief a date as possible.

capitalize
Capitalizes the first letter of the result.

date
Shows the date only.

day=*number*
Adds this number of days to the current date.

hour=*number*
Adds this number of hours to the current date.

lang=ca es_CA hr cs nl en fi fr de hu it jp mi no pt ru sr si es sv
Used together with *type=string* and the *part* attribute to get written dates in the specified language. Available languages are ca, es_CA (Catala), hr (Croatian), cs (Czech), nl (Dutch), en (English), fi (Finnish), fr (French), de (German), hu (Hungarian), it (Italian), jp (Japanese), mi (Maori), no (Norwegian), pt (Portuguese), ru (Russian), sr (Serbian), si (Slovenian), es (Spanish) and sv (Swedish).

lower
Prints the results in lower case.

minute=*number*
Adds this number of minutes to the current date.

part=year month day date hour minute second yday
• `year`; The year
• `month`; The month
• `day`; The weekday, starting with Sunday.
• `date`; The number of days since the first this month.
• `hour`; The number of hours since midnight.
• `minute`; The number of minutes since the last full hour.
• `second`; The number of seconds since the last full minute.
• `yday`; The day since the first of January.
The return value of these parts are modified by both `type` and `lang`.

second=*number*
Adds this number of seconds to the current date.

time
Prints the time only.

type=number string roman iso discordian stardate
Specifies what type of date you want. Discordian and stardate only make a difference when *not* using *part*. Note that *type=stardate* has a separate companion attribute, *prec*, which sets the precision.

unix_time=*time_t*
This attribute uses the specified Unix time_t time as the starting time, instead of the current time. This is mostly useful when the `<date>` tag is used from a Pike-script or Roxen module.

upper
Prints the result in upper case.

Example

```
<date part=day type=string lang=de>
```

Results in

```
Sonntag
```

<file>

`<file>` is defined in the *Main RXML parser* module.

Prints the path part of the URL used to get this page.

raw
Prints the full path part, including the query part with form variables.

Example

```
<file>
```

Results in

```
/dump-page.html
```

<help>

`<help>` is defined in the *Main RXML parser* module.

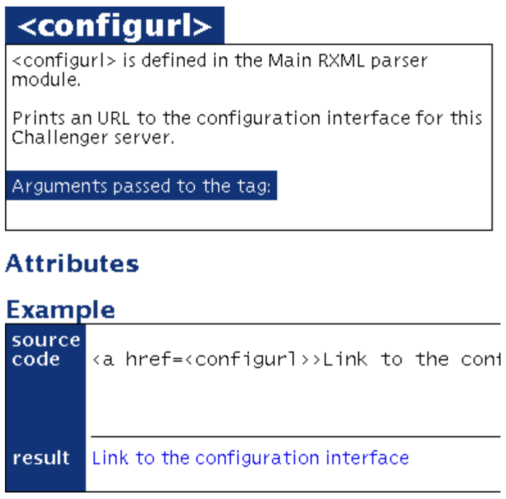
Gives help texts for tags. If given no arguments, it will list all available help texts.

for=*tag*
Gives the help text for that tag.

Example

`<help for=configurl>`

Results in



<configurl>

<configurl> is defined in the Main RXML parser module.

Prints an URL to the configuration interface for this Challenger server.

Arguments passed to the tag:

Attributes

Example

source code	`<a href=<configurl>>Link to the confi`
result	Link to the configuration interface

<available_languages>

`<available_languages>` is defined in the *Language* module.

Lists the number of additional languages the current page has been translated to, with links to them.

type=txt img
Whether to present the available languages with text or images. See the module documentation for information about how to configure which images to send.

<language>

`<language>` is defined in the *Language* module.

Prints the language of the current page.

type=txt img
Whether to present the language with text or an image. See the module documentation for information about how to configure which image to send.

<unavailable_language>

`<unavailable_language>` is defined in the *Language* module.

Shows the language the user wanted in case the page was not available in that language.

type=txt img
Whether to present the unavailable language with text or an image. See the module documentation for information about how to configure which image to send.

<line>

`<line>` is defined in the *Main RXML parser* module.

Prints the current line number of the current page.

Example

`The current line is line <line>.`

Results in

The current line is line 5.

<list-tags>

`<list-tags>` is defined in the *Main RXML parser* module.

Lists all available RXML tags.

verbose
Lists the tags with their help texts as well.

Example

`<list-tags>`

<modified>

`<modified>` is defined in the *Main RXML parser* module.

Prints when or by whom a page was last modified, by default the current page.

by
Print by whom the page was modified. Takes the same attributes as the `<user>` tag.

capitalize
Capitalizes the first letter of the result.

date
Print the modification date. All attributes from the <date> tag can be used.

file=*path*
Get information from this file rather than the current page.

lower
Print the result in lower case.

realfile=*path*
Get information from this file in the computers file-system rather than Challenger's virtual filesystem.

Example

```
This page was last modified <modified date
type=string>
```

Results in

This page was last modified May the 29th in the year of 1999

<number>

<number> is defined in the *Main RXML parser* module.

Prints a number as a word.

lang=ca es_CA hr cs nl en fi fr de hu it jp mi no pt ru sr si es sv
The language to use. Available languages are ca, es_CA (Catala), hr (Croatian), cs (Czech), nl (Dutch), en (English), fi (Finnish), fr (French), de (German), hu (Hungarian), it (Italian), jp (Japanese), mi (Maori), no (Norwegian), pt (Portuguese), ru (Russian), sr (Serbian), si (Slovenian), es (Spanish) and sv (Swedish).

Example

```
<number lang=es num=42>
```

Results in

cuarenta y dos

<pr>

<pr> is defined in the *Main RXML Parser* module.

Displays a *Powered by Roxen Challenger* logo.

size=small medium large
Defines the size of the logo.

color=blue brown green purple
Defines the color of the logo.

align=left center right
Defines the alignment of the logo.

Example

```
<pr size=medium color=green>
```

Results in

<referrer>

<referrer> is defined in the *Main RXML parser* module.

Prints the URL of the page on which the user followed a link that brought her to this page. The information comes from the referrer header sent by the browser.

alt=*string*
If no referrer header is found print this value instead.

Example

```
You came from
<a href='<referrer'>><referrer></a>
, didn't you?
```

Results in

You came from >.. , didn't you?

<user>

<user> is defined in the *Main RXML parser* module.

Prints information about the specified user. By default, the full name of the user and her e-mail address will be printed, with a mailto link and link to the home page of that user.

The `<user>` tag requires an *Authentification* module to work.

name
The login name of the user.

realname
Only print the full name of the user, with no link.

email
Only print the e-mail address of the user, with no link.

link
Include links. Only meaningful together with the *re-alname* or *email* attribute.

nolink
Don't include the links.

Example

```
<user name=wing realname>
```

Results in

```
Mattias Wingstedt
```

<version>

`<version>` is defined in the *Main RXML parser* mod-ule.

Print the version number of the Roxen Challenger web server you are using.

Example

```
This page has been brought to you by
<version>
```

Results in

```
This page has been brought to you by Roxen
Challenger/1.3.110
```

STRING TAGS

String tags are container tags that process their contents somehow. Examples are the `<sort>` tag that sorts its contents and the `<tablify>` tag that creates good looking tables from tab separated text files.

The contents of an RXML container tag may contain other RXML tags. However, this is not as simple as it may seem since the outer tag is, by default, handled first. The following example will try to explain what happens.

Our example contains an `<obox>` tag enclosing a `<smallcaps>` tag.

```
<obox>
<smallcaps>Hello World</smallcaps>
</obox>
```

Which will result in:

HELLO WORLD

The first thing that will happen is that the RXML parser handles the `<obox>` tag, which creates some HTML table code to draw a box around its contents. The result from the first pass will be something like:

```
<generated HTML table code>
<smallcaps>Hello World</smallcaps>
</generated HTML table code>
```

This result will then be parsed another time by the RXML parser, which will then run the `<smallcaps>` tag.

That the outer tag is handled first is usually not a problem, but in some special cases it will cause a problem. It is, therefore, possible to give the *preparse* attribute to all RXML container tags. This will cause the RXML parser to parse the contents of the tag before parsing the actual tag.

Below follows an example where the *preparse* attribute makes a huge difference.

```
<source>
<smallcaps>Hello World</smallcaps>
</source>
```

generates

```
<source>
<smallcaps>Hello World</smallcaps>
</source>
```

while

```
<source preparse>
<smallcaps>Hello World</smallcaps>
</source>
```

generates

```
H<font size=-1>ELLO</font> W<font size=-1>O
```

HELLO WORLD

Special Attributes

preparse is not the only special attribute that can be given to all RXML tags. They are

nooutput
The tag will generate no output at all. Side effects, for example sending queries to databases, will have effect.

noparse
Don't run the results of the tag through the RXML parser.

preparse
Run the contents of the tag through the RXML parser, before the tag itself is handled.

<ai> ... </

`<ai>` is defined in the *Indirect href* module.

Makes it possible to use a database of links. Each link is referred to by a symbolic name instead of the URL.

The database is updated through the configuration interface.

name
Which link to fetch from the database. There is a special case, *name=random* that will choose a random link from the database.

Example

```
<ai name=roxen>Roxen Platform</ai>
```

Results in

Roxen Platform

<autoformat> ... </

<autoformat> is defined in the *Main RXML parser* module.

Replaces all linefeeds in the content with <br> tags.

nobr
Don't add any <br> br tags.

pre
Replaces all double linefeeds with <p> tags.

Example

```
<autoformat>
It is almost like
using the pre tag.
</autoformat>
```

Results in

```
It is almost like
using the pre tag.
```

<case> ... </

<case> is defined in the *Main RXML parser* module.

Changes the case of the text in the contents.

lower
Changes all upper case letters to lower case.

upper
Changes all lower case letters to upper case.

capitalize
Capitalizes the first letter in the content.

Example

```
<case upper>upper case</case>
```

Results in

```
UPPER CASE
```

<comment> ... </

<comment> is defined in the *Main RXML parser* module.

The contents will be completely removed from the page. As opposed to HTML comments where you can still see the comment by doing *View Source* in the browser.

RXML tags within the <comment> tag will not be parsed.

<doc> ... </

<doc> is defined in the *Main RXML parser* module.

This tag simplifies writing html examples. Within the <doc> tag { will be replaced by < and } by >. Thus eliminating the need to write &lt; and &gt; manually.

pre
Encloses the section within a <pre> tag as well.

Example

```
<doc pre>
{table}
 {tr}
 {td} First cell {/td}
 {td} Second cell {/td}
 {/tr}
{/table}
</doc>
```

Results in

```
<table>
  <tr>
    <td> First cell </td>
    <td> Second cell </td>
  </tr>
</table>
```

<fl> ... </

<fl> is defined in the *Folder list tag* module.

This tag is used to build folding lists, that are like <dl> lists, but where each element can be unfolded. The tags used to build the list elements are <ft> and <fd>.

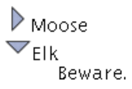
folded
An argument to both the <fd> itself as well as the <ft> tag. Will make all elements in the list or that element folded by default.

unfolded
An argument to both the <fd> itself as well as the <ft> tag. Will make all elements in the list or that element unfolded by default.

Example

```
<fl >
 <ft folded>Moose
 <fd>Tastes great.
 <ft unfolded>Elk
 <fd>Beware.
</fl>
```

Results in



<obox> ... </

<obox> is defined in the *Outlined box* module.

This tag draws outlined boxes.

align=left right
Vertical alignment of the box.

bgcolor=*color*
Color of the background and title label.

left=*number*
Length of the line on the left of the title.

outlinecolor=*color*
Color of the outline.

outlinewidth=*number*
Width, in pixels, of the outline.

right=*number*
Length of the line on the right of the title.

Note that the *left* and *right* attributes are constrained by the width argument.

spacing=*number*
Width, in pixels, of the space in the box.

style=caption groupbox
Style of the box. Groupbox is default

textcolor=*color*
Color of the text inside the box.

titlecolor=*color*
Color of the title text.

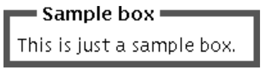
width=*number*
Width, in pixels, of the box.

If the title is not specified in the argument list, you can put it in a `<title>` container in the obox contents.

Example

```
<obox outlinecolor="#555555" outlinewidth="5"
width="200" align="left">
<title>Sample box</title>

This is just a sample box.

</obox>
```

Results in



<smallcaps> ... </

`<smallcaps>` is defined in the *Main RXML parser* module.

This tag prints the contents in smallcaps

size
Sets the base font size, which can be between 1 and 7. This is used for the upper case letters.

small
Sets the font size for the lower case letters.

space
Inserts a space between every letter.

Example

```
<smallcaps size=6 small=2 space>Roxen
Challenger</smallcaps>
```

Results in

R O X E N C H A L L E N G E R

<sort> ... </

`<sort>` is defined in the *Main RXML parser* module.

Sorts the contents divided by newline or the specified separator.

separator
The separator used to separate the elements that are to be sorted.

Example

```
<sort>
1
Hello
3
World
Are
2
You
Listening
</sort>
```

Results in

1 2 3 Are Hello Listening World You

<source> ... </

`<source>` is defined in the *Main RXML parser* module.

This tag is used to show examples of HTML or RXML code. It will first show the source code, then a separator and last the results of the code.

separator
Use this string as a separator between the presentation of the source of the result.

Example

```
<source separator="The result of the above
code">
<font size=+9><b>Bold</b></font>
<h5>This is a small heading</h5></source>
```

Results in

```
<font size=+9><b>Bold</b></font>
<h5>This is a small heading</h5>
```

The result of the above code

Bold

This is a small heading

<spell> ... </

`<spell>` is defined in the *Spell checker* module.

Checks and marks common misspellings in the contents.

warn
Report all unknown words.

Example

```
<spell warn>
Acctually spelling is not what I do best.
</spell>
```

Results in

Acctually spelling is not what I do best.

Spell checking report:

"acctually" is unknown to spellchecker

<tablify> ... </

`<tablify>` is defined in the *Tablify* module.

This tag generates tables from the contents, by default in tab separated form. This simplifies making tables significantly.

cellalign=left center right
Alignment of the contents of the cells.

cellseparator=*string*
The separator for separating columns, default is tab.

fields=num text
This is not an argument but rather a container tag used in the contents that sets the field type for each column. Fields marked numerically will be right aligned and formatted

nice
Generates tables with customizable layouts. The first row is referred to as the title row. The additional attributes are:

bgcolor=*color*
Sets the background color of your table.

titlebgcolor=*color*
Sets the background color of the title cell.

titlecolor=*color*
Sets the font color of the title cell.

fgcolorX=*color*
Sets the background color of cell X.

nicer
Generates tables with even more customizable layouts and gtext font capabilities for the title field. Nicer uses the same attributes as nice plus these:

font=*font*
Selects which gtext font to use for the title field.

scale=*factor*
Sets the scaling of the gtext font.

face=*font*
Sets the font face to use for the HTML text.

size=*number*
Sets the font size to use for the HTML text.

modulo=*number*
The number of rows that are to use the same color, default is one.

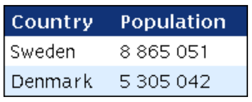
rowalign=left center right
This tag aligns the contents of the rows.

rowseparator=*string*
Sets the separator used for separating rows, default is newline.

Example

```
<tablify cellseparator="," nice="nice">
 Country, Population
 Sweden, 8 865 051
 Denmark, 5 305 042
</tablify>
```

Results in



Country	Population
Sweden	8 865 051
Denmark	5 305 042

<trimlines> ... </

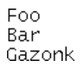
`<trimlines>` is defined in the *Main RXML parser* module.

This tag removes all empty lines from the contents.

Example

```
<pre>
<trimlines>

Foo

Bar

Gazonk

</trimlines>
</pre>
```

Results in



```
Foo
Bar
Gazonk
```

VARIABLE TAGS

Variable tags can be used to create dynamic web pages as well as making web pages thats easier to maintain. The tags have one thing in common, they store and retrieve information from different places:

variables
Form variables, as well as variables created with tags like `<set>` and `<cset>`. Variables are the backbone of RXML programming.

other
Other variables only exist in *output* tags, like `<sqloutput>`. The most common use is to transfer a value available from the *output* tag to a real variable, by using `<set variable=foo other=bar>`.

defines
or macros are fully internal to the server. They are mostly used for to save pages' authors typing repetitive blocks of text.

tags
It is possible to define new tags, or to redefine an existing HTML tag.

packages
Packages are ready-to-use defines and tags provided by the administrator of the server.

<set>

`<set>` is defined in the *Main RXML parser* module.

Sets a variable to a new value.

variable=*variable*
The variable to set.

debug
Provide debug messages in case the operation fails. `<set>` will normally fail silently.

define=*define*
Set the variable to the contents of this define.

expr=*expression*
Set the variable to the result of a simple mathematical expression. Operators that can be used are +, -, *, /, % and |. Only numerical values can be used in the expression.

eval=*rxml expression*
Set the variable to the result of this rxml expression.

from=*variable*
Set the variable to the value of the named variable.

other=*variable*
Set the variable to the value of this *other* variable. This is mostly useful from within *output* tags like `<sqloutput>` where all columns from the SQL result will be available as *other* variables.

value=*string*
Set the variable to this value.

If none of the above attributes are specified, the variable is unset. If debug is currently on, more specific debug information is provided if the operation failed.

Example

```
<set variable=foo value="Hello World">
```

```
<insert variable=foo>
```

Results in

```
Hello World
```

```
<set variable=foo eval="<date>">
```

```
<insert variable=foo>
```

Results in

```
03:07, May the 30th, 1999
```

<unset>

`<unset>` is defined in the *Main RXML parser* module.

Unsets a variable.

variable=*variable*
Specifies which variable to unset.

Example

```
<set variable=foo value="Hello World">
set: <insert variable=foo>
```

```
<br><unset variable=foo>
unset: <insert variable=foo>
```

Results in

```
set: Hello World
unset:
```

<cset> ... </

`<cset>` is defined in the *Main RXML parser* module.

Sets a variable to the contents of the tag.

variable=*variable*
The variable to set.

Example

```
<cset variable=foo>
Hello World
</cset>
```

```
<insert variable=foo>
```

Results in

Hello World

<append>

`<append>` is defined in the *Main RXML parser* module.

Append a value to a variable.

variable=*variable*
The variable to append to.

debug
Provide debug messages in case the operation fails. `<append>` will normally fail silently.

define=*define*
Append the contents of this define.

from=*variable*
Append the value of the named variable.

other=*variable*
Append the value of this *other* variable. This is mostly useful from within *output* tags like `<sqloutput>` where all columns from the sql result will be available as *other* variables.

value=*string*
Append the variable to this value.

Example

```
<set variable=foo value="Hello">
<append variable=foo value=" World">
```

```
<insert variable=foo>
```

Results in

Hello World

<define> ... </

`<define>` is defined in the *Main RXML parser* module.

Defines new tags, container tags or defines.

container=*name*
Define a new RXML container tag, or override a previous definition.

name
Sets the specified define. Can be inserted later by the `<insert>` tag.

tag
Defines a new RXML tag, or overrides a previous definition.

default_*attribute*=*value*
Set a default value for an attribute, that will be used when the attribute is not specified when the defined tag is used.

You can use a few special tokens in the definition of tags and container tags:

#args#
All arguments sent to the tag. Useful when defining a new tag that is more or less only an alias for an old one.

&attribute;
Inserts the value of that attribute.

Example

```
<define container=h1>
<gtext fg=blue #args#><contents></gtext>
</define>
```

```
<h1>Hello</h1>
```

Results in

Hello

```
<define tag=test default_foo=foo
 default_bar=bar>
The test tag: Testing testing.
Foo is &foo;, bar is &bar;
</define>
```

```
<test foo=Hello bar=World>
<br><test foo=Hello>
```

Results in

The test tag: Testing testing. Foo is Hello, bar is World

The test tag: Testing testing. Foo is Hello, bar is bar

<undefine>

`<undefine>` is defined in the *Main RXML parser* module.

Undefines a previously defined tag, container tag or define.

name
Undefine this define.

tag
Undefine this tag.

container
Undefine this container tag.

Example

```
<define container=h1>
<gtext><contents></gtext>
</define>

<h1>Hello</h1>

<undefine container=h1>

<h1>World</h1>
```

Results in

Hello

World

<insert>

`<insert>` is defined in the *Main RXML parser* module.

Inserts values from files, cookies, defines or variables. If used to insert cookies or variables `<insert>` will quote before inserting, to make it impossible to insert dangerous RXML tags.

cookie=*cookie*
Inserts the value of the cookie.

cookies=full
Inserts the value of all cookies. With the optional argument full, the insertion will be more verbose.

encode=none html
Determines what quoting method should be when inserting cookies or variables. Default is *html*, which means that <, > and & will be quoted, to make sure you can't insert RXML tags. If you choose *none* nothing will be quoted. It will be possible to insert dangerous RXML tags so you must be of what your variables contain.

define=*name*
Inserts this define, which must have been defined by the `<define>` tag before it is used. The define can be done in another file, if you have inserted the file.

file=*path*
Inserts the file. This file will then be fetched just as if someone had tried to fetch it through an HTTP request. This makes it possible to include things like the result of Pike scripts.

If path does not begin with `/`, it is assumed to be a URL relative to the directory containing the page with the `<insert>` tag. Note that included files will be parsed if they are named with an extension the main RXML parser handles. This might cause unexpected behavior. For example, it will not be possible to share any macros defined by the `<define>` tags.

If you want to have a file with often used macros you should name it with an extension that won't be parsed. For example, `.txt`.

fromword=toword
Replaces fromword with toword in the macro or file, before insering it. Note that only lower case character sequences can be replaced.

nocache
Don't cache results when inserting files, but always fetch the file.

variable=*variable*
Insert the variable.

Example

```
<define name=foo>This is a foo</define>

<insert name=foo>
<br><insert name=foo foo=cat>
<br><insert name=foo a=some foo=cats is=are>
```

Results in

```
This is a foo
This is a cat
Thare are some cats
```

<use>

`<use>` is defined in the *Main RXML module* module.

Reads tags, container tags and defines from a file or package.

file=*path*
Reads all tags and container tags and defines from the file.

This file will be fetched just as if someone had tried to fetch it with an HTTP request. This makes it possible to use Pike script results and other dynamic documents. Note, however, that the results of the parsing are heavily cached for performance reasons. If you do not want this cache, use `<insert file=... nocache>` instead.

package=*name*
Reads all tags, container tags and defines from the given package. Packages are files located in `local/rxml_packages/`.

By default, the package `gtext_headers` is available, that replaces normal headers with graphical headers. It redefines the h1, h2, h3, h4, h5 and h6 container tags.

The `<use>` tag is much faster than the `<include>`, since the parsed definitions is cached.

Example

```
<use package=gtext_headers>
```

```
<h1>Hello World</h1>
```

Results in

```
Hello World
```

<formoutput> ... </

`<formoutput>` is defined in the *Main RXML parser* module.

A tag for inserting variables into just about any context. By default anything within #'s will be interpreted as a variable. Thus #name# will be replaced by the value of the variable name. ## will be replaced by a #.

By default, the variable will be HTML quoted, that is, < will be inserted as &lt; > as &gt; and & as &amp;. However, there are instances when that is not what you want, for example, when inserting variables into SQL queries. Therefore, the quoting can be controlled by #variable : quote=*scheme*#. The different quoting schemes are:

none
No quoting. This is dangerous and should never be used unless you have total control over the contents of the variable. If the variable contains an RXML tag, the tag will be parsed.

html
The default quoting, for inserting into regular HTML or RXML.

dtag
For inserting into HTML or RXML attributes that are quoted with ". For example `<<img src="/base/#image#">>`.

stag
For inserting into HTML or RXML attributes that are quoted with '. For example `<<img src='/base/#image#'>>`.

url
For inserting variables into URLs.

pike
For inserting into Pike strings, for use with the `<pike>` tag.

js, javascript
For inserting into Javascript strings.

mysql
For inserting into MySQL SQL queries.

sql, oracle
For inserting into SQL queries.

quote
Select the string used for quoting the variable, default is #.

Example

```
<set variable=foo value="World">

<formoutput quote=$>
Hello $foo$
</formoutput>
```

Results in

```
Hello World
```

URL TAGS

URL tags are tags that somehow use or manipulate the URL or HTTP headers. Among other things they manipulate;

prestate options
Prestate options are a way to present options in the URL, that will be persistent for a user over several pages. A prestate for the options `txt` and `en` would be stored as `http://www.roxen.com/(en,txt)/my.page` in the URL. If you use prestate options you must only use relative URLs in your links.

cookies
Cookies are a way for a web site to store a small amount of information in the users' browsers. It is a much better way than prestates to handle information that should be persistent for a user over several pages.

authentification
HTTP can be used to transmit a user name and a password through HTTP.

expire
It is possible to tell the browser, and any proxy on the way to it, how long it is to cache a page.

<apre> ... </

<apre> is defined in the *Main RXML parser* module.

Adds or removes prestate options.

Prestate options are simple true/false flags that are added to the URL of the page. Use `<if prestate=...>` to test for the presence of a prestate. <apre> works just like a `<a href=...>` container tag, but the *href* attribute can be omitted in which case the current page is used.

option
Add the prestate option.

-option
Remove the prestate option.

href
Make the generated link point to this URL. The URL must be local to this web site.

Example

```
<apre foo>Add the option</apre>
<br><apre -foo>Remove the option</apre>
```

```
<p><if prestate=foo>
The option is set.
</if>
<else>
The option is not set.
</else>
```

<aconf> ... </

<aconf> is defined in the *Main RXML parser* module.

Adds or removes config options.

Config options are simple toggles that are stored in the cookie *roxen-config*. This ensures that they are persistent for that user, the same user will have the same config options even if he returns to the site another day. If cookies cannot be used, prestate variables are used instead.

Use `<if config=...>` to test for the presence of a config option. <aconf> works just like the `<a href=...>` container tag, but if no *href* attribute is specified, the current page is used.

Example

```
<aconf +foo>Add the option</aconf>
<br><aconf -foo>Remove the option</aconf>
```

```
<p>if config=foo>
The option is set.
</if>
<else>
The option is not set.
</else>
```

<set_cookie>

<set_cookie> is defined in the *Main RXML parser* module.

Sets a cookie that will be stored by the user's browser. This is a simple and effective way of storing data that is local to the user. The cookie will be persistent, the next time the user visits the site, she will bring the cookie with her.

name=*string*
The name of the cookie.

value=*string*
The value the cookie will be set to.

persistent
Keep the cookie for two years.

hours=*number*
Add this number of hours to the time the cookie is kept.

minutes=*number*
Add this number of minutes to the time the cookie is kept.

seconds=*number*
Add this number of seconds to the time the cookie is kept.

days=*number*
Add this number of days to the time the cookie is kept.

weeks=*number*
Add this number of weeks to the time the cookie is kept.

months=*number*
Add this number of months to the time the cookie is kept.

years=*number*
Add this number of years to the time the cookie is kept.

It is not possible to set the date beyond year 2038. By default the cookie will be kept until the year 2038.

Note that the change of a cookie will not take effect until the next page load. Therefore, a reload will be needed to see the effect of the example.

Example

```
<apre foo>Set the cookie</apre>
<br><apre -foo>Remove the cookie</apre>

<if prestate=foo><set_cookie name=foo
value="Hello World"></if>
<else><remove_cookie name=foo></else>

<p><insert cookie=foo>
```

<remove_cookie>

`<remove_cookie>` is defined in the *Main RXML parser* module.

Removes a cookie.

name
Name of the cookie to remove.

Note that removing a cookie won't take effect until the next page load. Therefore, a reload will be needed to see the effect of the example.

Example

```
<apre foo>Set the cookie</apre>
<br><apre -foo>Remove the cookie</apre>

<if prestate=foo><set_cookie name=foo
value="Hello World"></if>
<else><remove_cookie name=foo></else>

<p><insert cookie=foo>
```

<auth-required>

`<auth-required>` is defined in the *Main RXML parser* module.

Adds an HTTP auth required header and return code, that will force the user to supply a login name and password. This tag is needed when using access control in RXML in order for the user to be prompted to login.

Example

```
<apre foo>
Try it.
</apre>

<if prestate=foo>
<auth-required>
</if>
```

Results in

Try it.

<expire-time>

`<expire-time>` is defined in the *Main RXML parser* module.

Sets the expire-time for the document. Caches along the way to the user are only allowed to cache the page for this amount of time.

hours=*number*
Add this number of hours to the expire time.

minutes=*number*
Add this number of minutes to the expire time.

seconds=*number*
Add this number of seconds to the expire time.

days=*number*
Add this number of days to the expire time.

months=*number*
Add this number of months to the expire time.

years=*number*
Add this number of years to the expire time.

Bugs: it is not possible to set the date beyond the year 2038.

You can check our example by asking your browser about the *Page Info*.

Example

```
<expire-time hours=5>
```

<header>

`<header>` is defined in the *Main RXML parser* module.

Adds an HTTP header to the result from page.

See the Appendix for a list of HTTP headers.

name=*string*
The name of the header.

value=*string*
The value of the header.

Example

```
<apre foo>Try it</apre>

<if prestate=foo>
<header name=Location value=http://
www.roxen.com/>
<return code=301>
</if>
```

<redirect>

`<redirect>` is defined in the *Main RXML parser* module.

Adds an HTTP redirect header and return code to the response from this page.

to=*URL*
Redirect to this URL.

Example

```
<apre foo>Try it</apre>

<if prestate=foo>
<redirect to="http://www.roxen.com/">
</if>
```

<return>

`<return>` is defined in the *Main RXML parser* module.

Changes the HTTP return code for this page.

See the Appendix for a list of HTTP return codes.

code
The return code to set.

Example

```
<apre foo>Try it</apre>

<if prestate=foo>
<header name=Location value=http://
www.roxen.com/>
<return code=301>
</if>
```

<killframe>

`<killframe>` is defined in the *Killframe tag* module.

Prevents your page from being placed in a frame, by adding some JavaScript code.

As an added bonus `index.html` will be removed from the end of the URL, as shown in the *Location* field in your browser.

Example

```
<killframe>
```

IF TAGS

If-tags make it possible to make dynamic pages that show different content based on conditions. Authenticated users can get confidential information and pages can be optimized for all browsers. They also makes it possible to program web application in RXML, without using any programming language.

<if> ... </

`<if>` is defined in the *Main RXML parser* module.

`<if>` is used to conditionally show its contents. `<else>`, `<elif>` or `<elseif>` can be used to suggest alternative content.

It is possible to use glob patterns in almost all attributes, where * means match zero or more characters while ? matches one character. * Thus `t*f??` will match `trainfoo` as well as * `tfoo` but not `trainfork` or `tfo`.

accept=*type1[,type2,...]*
Returns true is the browser accept certain content types as specified by it's Accept-header, for example *image/jpeg* or *text/html*. If browser states that it accepts */* that is not taken in to account as this is always untrue.

config=*name*
Has the config been set by use of the `<aconf>` tag?

cookie=*name[is value]*
Does the cookie exist and if *value* is given, does it contain the value *value*?

date=*yyyymmdd*
Is the date yyyymmdd? The attributes *before*, *after* and *inclusive* modifies the behavior.

defined=*define*
Is the define defined?

domain=*pattern[,pattern...]*
Does the users computer's DNS name match any of the patterns? Note that domain names are resolved asynchronously, and the the first time someone accesses a page, the domain name will probably not have been resolved.

eval=*RXML expression*
Returns true if `RXML expression` returns a string that evaluates to true if casted to an integer in Pike, i.e. the string begins with 1-9 or a number of zeroes followed by 1-7 (octal greater than zero). **Future versions of Roxen (starting from version 1.4) will evaluate to true on a number of zeroes followed by 1-9 (decimal greater than zero).**

exists=*path*
Returns true if the file `path` exists. If `path` does not begin with `/`, it is assumed to be a URL relative to the directory containing the page with the `<if>`-statement.

filename=*filepattern1[,filepattern2,...]*
Returns true if the current page is among the listed filepatterns.

host=*pattern[,pattern...]*
Does the users computers IP address match any of the patterns?

language=*language1[,lang2,...]*
Does the client prefer one of the languages listed, as specified by the `Accept-Language` header?

match=*string[is pattern[,pattern,...]]*
Does the string match one of the patterns?

name=*pattern[,pattern...]*
Does the full name of the browser match any of the patterns?

prestate=*option1[,option2, ...]*
Are all of the specified prestate options present in the URL?

referrer=*[=pattern[,pattern,...]]*
Does the referrer header match any of the patterns?

supports=*feature*
Does the browser support this feature? See the support feature page page for a list of all available features.

time=*ttmm*
Is the date ttmm? The attributes *before*, *after* and *inclusive* modifies the behavior.

user=*name[,name,...]|any*
Has the user been authenticated as one of these users? If *any* is given as argument, any authenticated user will do.

variable=*name[is pattern]*
Does the variable exist and, optionally, does it's content match the pattern?

Modifier Attributes

after
Used together with the *date* attribute.

and
If several conditional attributes are given all must be true for the contents to be shown. This is the default behavior.

before
Used together with the *date* attribute.

file=*path*
Used together with the *user* attribute. An external file will be used to authenticate the user, rather than the current *Authentication* module. The file should have the following format:

```
user name : encrypted password
user name : encrypted password
```

Unless the *wwwfile* attribute is given the path is a path in the computers real file system, rather than Challenger's virtual file system.

group=*group, groupfile path*
Used together with the *user* attribute to check if the current user is a member of the group according the the groupfile. The groupfile is of the following format:

```
group : user1, user2, user3
group : user4
```

inclusive
Used together with the *date* and *before* or *after* attributes. The contents will also be shown if the date is the current date.

wwwfile
Used together with the *file* attribute to indicate what Challenger's virtual file system should be used to find the password file. This might be a security hazard, since anyone will be able to read the password file.

not
Inverts the results of all tests.

or
If several conditional attributes are given, only one of them has to be true for the contents to be shown.

Complex expressions
You might be tempted to write expressions like:

```
<if variable="foo is bar" or variable="bar is foo">Something</if>
```

This will not work, as you can only use an attribute once.

Another common problem is a misconception of how the *and*, *or* and *not* attributes work.

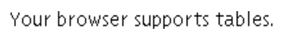
```
<if user=foo or not do-
main="*.foobar.com">...</if>
```

will not work since the *not* attribute negates the whole tag, not just the *domain* attribute.

Example

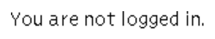
```
<if supports=tables>
Your browser supports tables.
</if>
```

Results in


Your browser supports tables.

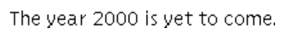
```
<if user=any>
You are logged in.
</if>
<else>
You are not logged in.
</else>
```

Results in


You are not logged in.

```
<if date=20000101 before>
The year 2000 is yet to come.
</if>
```

Results in


The year 2000 is yet to come.

<else> ... </

`<else>` is defined in the *Main RXML parser* module.

Show the contents if the previous `<if>` tag didn't, or if there was a `<false>` above. The result is undefined if there has been no `<if>`, `<true>` or `<false>` tag above.

Example

```
<false>
<else>
Show this.
</else>
```

Results in

```
Show this.
```

<elseif> ... </

`<elseif>` is defined in the *Main RXML parser* module.

Same as the `<if>`, but it will only evaluate if the previous `<if>` tag returned false.

<elif> ... </

`<elif>` is defined in the *Main RXML parser* module.

Alias for the `<elseif>` tag.

<true>

`<true>` is defined in the *Main RXML parser* module.

An internal tag used to set the return value of `<if>` tags. It will ensure that the next `<else>` tag will not show its contents. It can be useful if you are writing your own `<if>` lookalike tag.

<false>

`<false>` is defined in the *Main RXML parser* module.

Internal tag used to set the return value of `<if>` tags. It will ensure that the next `<else>` tag will show its contents. It can be useful if you are writing your own `<if>` lookalike tag.

GRAPHICS TAGS

Good looking graphics are an important part of the layout of web pages. But creating graphics can also be very time consuming. Especially, if it involves creating the same type of headers, only with different text.

Therefore, Challenger features graphic tags that create images. They can be used to draw analog clocks and graphical headers as well as diagrams.

File Formats

Some of the tags take images as attributes. For example, to use as background. The images can be in GIF, JPEG, PNM or PNG format.

Color Attributes

Color attributes can be specified in one of the following ways:

name
For example *black* or *darkred*.

#RGB value
The color is specified as a hexadecimal-digits, `#RRGGBB`. For example, *#ffdead* or *#00ff00*.

@HSV value
The color is specified with the syntax `@h,s,v` where h is the hue specified as degrees (0 to 359), s is the saturation specified as a percentage and v the value also specified as a percentage. For example, *@150,70,70*.

%CMTK value
The color is specified with the syntax `%c,m,t,k` where all the values are percentages. For example, *%10,20,30,40*.

<gtext> ... </

`<gtext>` is defined in the *Graphics text* module.

Renders a GIF image of the enclosed text.

Note: If the background and text colors are not set in the `<body>` tag of the page, the *bg* and *fg* attributes must be set, otherwise the `<gtext>` tag will only render a "Please reload this page"-message.

alpha=*path*
Use the specified image as an alpha channel, together with the *background* attribute.

href=*URL*
Link the image to the specified URL. The link color of the document will be used as the default foreground rather than the foreground color.

alt=*string*
Sets the *alt* attribute of the generated `<img>` tag. Will be default set to the alt attribute of the contents of the `<gtext>` tag.

background=*path*
Specifies the image to use as background.

bevel=*width*
Draws a bevel box.

pressed
Inverts the direction of the bevel box, to make it look like a button that is pressed down.

bg=*color*
Tells the `<gtext>` tag what the background color of the page is. This is used for anti-alias purposes. The module can be configured to try to find out this by itself, by parsing at appropriate HTML tags.

black
Use a black, or heavy, version of the font, if available.

bold
Use a bold version of the font, if available.

border=*width,color*
Draws a border around the text of the specified width and color.

fadein=*blur,steps,delay,initialdelay*
Generates an animated GIF file of a fade-in effect.

fg=*color*
Sets the color or the rendered text. The module can be configured to try to find out an appropriate color by parsing HTML tags.

fs
Apply floyd-steinberg dithering to the resulting image.

fuzz=*color*
Apply a glow effect.

ghost=*dist,blur,color*
Apply a ghost effect. Cannot be used together with *shadow* or *magic*.

glow=*color*
Apply a glowing outline around the text.

italic
Use an italic version of the font, if available.

light
Use a light version of the font, if available.

magic=*message*
Used together with the *href* attribute to generate a JavaScript that will highlight the image when the mouse is moved over it.

magic_*argument*=*value*
Same as for any `<gtext>` attribute, except for the highlighting image.

magicbg=*color|path*
Same as the *background* attribute, except for the highlighting image.

maxlen=*number*
Sets the maximum length of the rendered text, by default 300.

mirrortile
Tiles the background and foreground images around x-axis and y-axis for odd frames, creating seamless textures.

move=*x,y*
Moves the text relative to the upper left corner of the background image. This will not change the size of the image.

nfont=*font*
Use this font. If no font is specified, the define `nfont` will be used, or the default font, if there is no define.

notrans
By default, the background of the image is set as a transparent color. This option overrides that behavior.

opaque=*percentage*
Generate text with this amount of opaqueness. 100% is default.

quant=*number*
Use this number of colors in the generated image. For GIF images, fewer colors implies smaller images but also aliasing effects. It is advisable to use powers of 2 to optimize the palette allocation.

rescale
Rescale the background to fill the whole image.

rotate=*angle*
Rotates the image this number of degrees counterclockwise.

scale=*float*
Scale the font this much.

scolor=*color*
Use this color for the shadow. Used with the *shadow* attribute.

scroll=*width,steps,delay*
Generate an animated GIF image of the text scrolling.

shadow=*intensity,distance*
Draw a drop-shadow with the specified intensity and distance. The intensity is specified as a percentage.

size=*width,height*
Set the size of the image.

spacing=*number*
Add space around the text.

split
Generate a separate GIF image out of each word. This will allow the browser to word-wrap the text, but will disable certain attributes like *magic*.

split=*character*
Split the string also at each occurrence of the character.

talign=left eight center
Adjust the alignment of the text.

textbelow=*color*
Place the text in a colored box.

textbox=*opaque,color*
Draw a box with an opaque value below the text of the specified color.

texture=*path*
Uses the specified images as a field texture.

tile
Tiles the background and foreground images if they are smaller than the actual image.

turbulence=*frequency,color;frequency,color;frequency,color*
Apply a turbulence effect.

verbatim
Allows the gtext parser to not be typographically correct.

xpad=*percentage*
Increases padding between characters.

xsize=*number*
Sets the width.

xspacing=*number*
Sets the horizontal spacing.

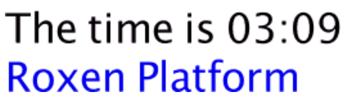
ysize=*number*
Sets the height.

yspacing=*number*
Sets the vertical spacing.

Example

```
<gtext>The time is <date time></gtext>
<br><gtext href=http://www.roxen.com/
magic>Roxen Platform</gtext>
```

Results in


The time is 03:09
Roxen Platform

<diagram> ... </

`<diagram>` is defined in the *Business Graphics* module.

The `<diagram>` container tag is used to draw pie, bar, or line charts as well as graphs. It is quite complex with six internal container tags.

Internal Tags

<data>
The data the diagram is to visualize, in tabular form.

<colors>
The colors for different pie slices, bars or lines.

<legend>
A separate legend with description of the different pie slices, bars or lines.

<xaxis>
Used for specifying the quantity and unit of the x-axis, as well as its scale, in a graph.

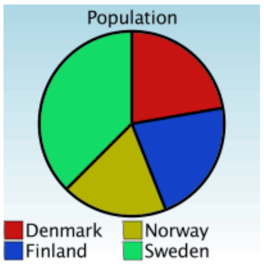
<yaxis>
Used for specifying the quantity and unit of the x-axis, as well as its scale, in a graph or line chart.

<xnames>
Separate tag that can be used to give names to put along the pie slices or under the bars. The names are usually part of the data.

Pie

```
<diagram type=pie width=200 height=200
name='Population'
tonedbox='lightblue,lightblue,white,white'>
<data separator=,>
5305048,5137269,4399993,8865051
</data>
<legend separator=,>
Denmark,Finland,Norway,Sweden</legend>
</diagram>
```

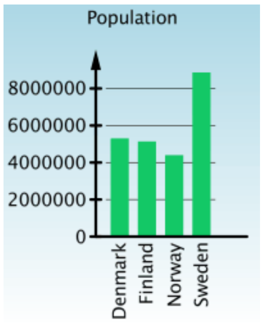
Results in



Bar

```
<diagram type=bar width=200 height=250
name='Population' horgrid
tonedbox='lightblue,lightblue,white,white'>
<data xnamesvert xnames separator=,>
Denmark,Finland,Norway,Sweden
5305048,5137269,4399993,8865051</data>
</diagram>
```

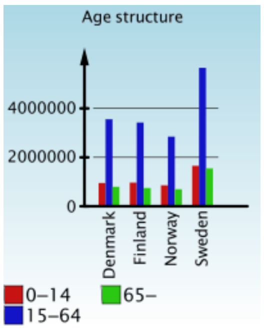
Results in



```
<diagram type=bar width=200 height=250
name='Age structure' horgrid
tonedbox='lightblue,lightblue,white,white'>
```

```
<data xnamesvert xnames form=column
 separator=,>
Denmark,951175,3556339,797534
Finland,966593,3424107,746569
Norway,857952,2846030,696011
Sweden,1654180,5660410,1550461</data>
<legend separator=,>
0-14,15-64,65-
</legend>
</diagram>
```

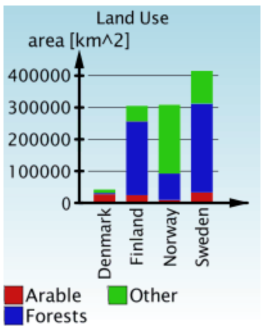
Results in



Sumbar

```
<diagram type=sumbar width=200 height=250
name='Land Use' horgrid
tonedbox='lightblue,lightblue,white,white'>
<data xnamesvert xnames form=column
 separator=,>
Denmark,27300,4200,10500
Finland,24400,231800,48800
Norway,9240,83160,215600
Sweden,32880,279480,102750</data>
<legend separator=,>
Arable,Forests,Other
</legend>
<yaxis quantity=area>
<yaxis unit=km^2>
</diagram>
```

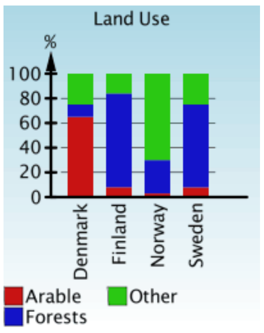
Results in



Normalized Sumbar

```
<diagram type=normsumbar width=200 height=250
name='Land Use' horgrid
tonedbox='lightblue,lightblue,white,white'>
<data xnamesvert xnames form=column
 separator=,>
Denmark,27300,4200,10500
Finland,24400,231800,48800
Norway,9240,83160,215600
Sweden,32880,279480,102750
</data>
<legend separator=,>
Arable,Forests,Other
</legend>
<yaxis quantity=%>
</diagram>
```

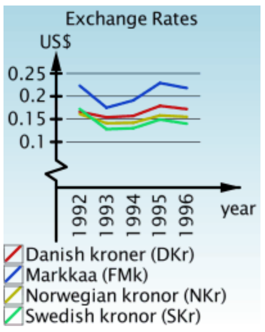
Results in



Line Chart

```
<diagram type=line width=200 height=250
name='Exchange Rates' horgrid
tonedbox='lightblue,lightblue,white,white'>
<data form=row separator=,
 xnamesvert xnames>
1992,1993,1994,1995,1996
0.166,0.154,0.157,0.179,0.172
0.223,0.175,0.191,0.229,0.218
0.161,0.141,0.142,0.158,0.155
0.172,0.128,0.130,0.149,0.140
</data>
<yaxis start=0.09 stop=0.25>
<legend separator=,>
Danish kroner (DKr),
Markkaa (FMk),
Norwegian kronor (NKr),
Swedish kronor (SKr)
</legend>
<xaxis quantity=year>
<yaxis quantity=US$>
</diagram>
```

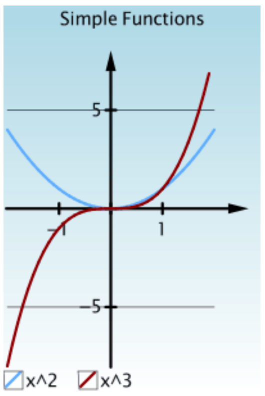
Results in



Graph

```
<diagram type=graph width=200 height=300
name='Simple Functions' horgrid
tonedbox='lightblue,lightblue,white,white'>
<colors separator=" ">#60b0ff darkred</colors>
<data separator=,><pike>
 float c;

 for (c=-2.0; c < 2.0; c+=0.1)
 output( "%f,%f,", c, c * c );
 output( "%f,%f", 2.0, 2.0 * 2.0 );
 return flush();
</pike>
<pike>
 float c;

 for (c=-2.0; c < 2.0; c+=0.1)
 output( "%f,%f,", c, c * c * c );
 output( "%f", 2.0, 2.0 * 2.0 * 2.0 );
 return flush();
</pike></data>
<axis start=-2.1 stop=2.1>
<axis start=-6.1 stop=6.1>
<legend separator=,>
x^2,x^3
</legend>
</diagram>
```

Results in



3d=*number*
Draws a pie-chart on top of a cylinder, takes the height of the cylinder as argument.

background=*path*
Use the image as background.

bgcolor=*color*
Set the background color to use for anti-aliasing.

center=*number*
Centers a pie chart around that slice.

eng
Write numbers in engineering fashion, i.e like 1.2M.

font=*font*
Use this font. Can be overridden in the `<legend>`, `<xaxis>`, `<yaxis>` and `<names>` tags.

fontsize=*number*
Height of the text.

height=*number*
Height of the diagram. Will not have effect below 100.

horgrid
Draw a horizontal grid.

labelcolor=*color*
Set the color for the labels of the axis.

legendfontsize=*number*
Height of the legend text.

name=*string*
Write a name at the top of the diagram.

namecolor=*color*
Set the color of the name, by default *textcolor*.

namefont=*font*
Set the font for the name.

namesize=*number*
Sets the height of the name, by default *fontsize*.

neng
As eng, but 0.1-1.0 is written as 0.xxx.

notrans
Make bgcolor opaque.

rotate=*degree*
Rotate a pie chart this much.

textcolor
Set the color for all text.

tonedbox=*color1,color2,color3,color4*
Create a background shading between the colors assigned to each of the four corners.

turn
Turn the diagram 90 degrees.

type=sumbars normsum line bar pie graph
The type of the diagram.

vertgrid
Draw vertical grid lines.

voidsep=*string*
Change the string that means no such value, by default VOID.

width=*number*
Set the width of the diagram.

xgridspace=*number*
Set the space between two vertical grid lines. The unit is the same as for the data.

ygridspace
Set the space between two horizontal grid lines. The unit is the same as for the data.

Regular `<img>` arguments will be passed on to the generated `<img>` tag.

<data>

form
column row How to interpret the tabular data, by default row.

lineseparator
Set the separator between rows, by default newline. lineseparator.

noparse
Do not parse the contents by the RXML parser, before data extraction begins.

separator
Set the separator between elements, by default tab.

xnames
Treat the first row or column as names for the data to come. The name will be written along the pie slice or under the bar.

xnamesvert
Write the names vertically.

<colors>

separator
Set the separator between colors, by default tab.

<legend>

separator
Set the separator between legends, by default tab.

<xaxis>, <yaxis>

start
Limit the start of the diagram at this value. If set to *min* the axis starts at the lowest value in the data.

stop
Limit the end of the diagram at this value.

quantity
Set the name of the quantity of this axis.

unit=*string*
Set the name of the unit of this axis.

<xnames>

separator
Set the separator between names, by default tab.

orient=vert horiz
How to write names, vertically or horizontally.

<gclock>

`<gclock>` is defined in the *Pike Image Module* module.

Draw an analogue clock that will always show the right time through creative usage of GIF animations.

background_image=*path*
Set the background image to use.

delay
Set the delay between the frames in the animation.

time_offset
Add or subtract a number of seconds to the actual time.

Example

```
<gclock>
```

Results in

<pimage> ... </

<pimage> is defined in the *Pike Image Module* module.

A <pike> tag optimized for creating images or GIF animations.

<imgs>

<imgs> is defined in the *Main RXML parser* module.

Works like an <img> tag where the server automatically sets the *width* and *height* attributes. That way, the page will be rendered faster by the browser while no information about the image is hard-coded into the page. If the image changes size, so will the *width* and *height* attributes. The server will read the first bytes of the image file to determine its size.

The <imgs> takes the same arguments as the <img> tag.

<config_tablist> ... </

<config_tablist> is defined in the *Config tab-list* module.

Generates a list of tabs, like the one in the configuration interface.

The <config_tablist> container tag does not take any arguments, but it must always contain one or more <tab> container tags. The following arguments apply to the <tab> tags.

alt
Alternative text for the image. The default is to use ascii-art to make it look like a tablist.

bgcolor
Set the background color. Default is white.

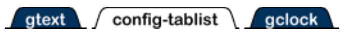
border
Set the width of the border of the image. Default is zero.

selected
Make this tab the selected tab.

Example

```
<config_tablist>
 <tab href="gtext.html">gtext</tab>
 <tab selected="selected">config-tablist</tab>
 <tab href="gclock.html">gclock</tab>
</config_tablist>
```

Results in



DATABASE TAGS

The database tags interact with SQL databases. They can be used to create interactive graphical reports as well as complete web applications.

The database tags are almost always combined with other RXML tags. Together with the `<diagram>` tag they provide real-time diagrams, with the `<tablify>` tag they provide nice looking tables. Combined with the `<wizard>` tag they make easy-to-use web applications.

Each database tag needs to know which database it should connect to. This is specified by the *host*-attribute which usually is a symbolic name for the database that the administrator has configured in the *SQL databases* module. It is also possible to specify the database host, user and password directly in the tags, but this is not recommended. See the Database chapter in the Administrator's manual for the syntax.

<sqlquery>

`<sqlquery>` is defined in the *SQL* module.

Executes a SQL query, but doesn't do anything with the result. This is mostly used for SQL queries that change the contents of the database, for example INSERT or UPDATE.

host=*database*
Which database to connect to, usually a symbolic name. If omitted the default database will be used.

query=*SQL query*
The actual SQL-query.

quiet
Do not show any errors in the page, in case the query fails.

parse
If specified, the query will be parsed by the RXML parser. Useful if you wish to dynamically build the query.

Example

```
<apre foo>Reset the database</apre>

<if prestate=foo>
 <sqlquery host=test
 query="DELETE from test">
</if>
```

<sqltable>

`<sqltable>` is defined in the *SQL* module.

Creates a HTML or ASCII table from the results of an SQL query.

ascii
Create an ASCII table rather than a HTML table. Useful for
interacting with the `<diagram>` and `<tablify>`
tags.

host=*database*
Which database to connect to, usually a symbolic name.
If omitted the default database will be used.

query
The actual SQL-query.

quiet
Do not show any errors in the page, in case the query fails.

parse
If specified, the query will be parsed by the RXML parser.
Useful if you wish to dynamically build the query.

Example

```
<tablify preparse="preparse"
nice="nice">CountryPopulation
<sqltable ascii host=test query="SELECT
country, population FROM countries ORDER BY
country">
</tablify>
```

<sqloutput> ... </

`<sqloutput>` is defined in the *SQL* module.

Insert the results of a SQL query into HTML or RXML. `<sqloutput>` works like all *output* tags. By default anything within #'s will be interpreted as a variable. Thus #column# will be replaced by the column value. ## will be replaced by a #. The inserted SQL results will by default be HTML quoted, < will for example be quoted to &lt;. See the formoutput page for more information about quoting.

The `<sqloutput>` tag will copy its contents and replace the columns for each row in the result of the query. If the result is empty, the `<sqloutput>` will not return anything.

Within the `<sqloutput>` the column values can be accessed as *other* variables. This is useful for transferring the result to normal RXML variables.

host=*database*
Which database to connect to, usually a symbolic name. If omitted the default database will be used.

query=*SQL query*
The actual SQL-query.

quiet
Do not show any errors in the page, in case the query fails.

parse
If specified, the query will be parsed by the RXML parser. Useful if you wish to dynamically build the query.

Example

```
<table >
 <tr>
 <th>Country</th>
 <th>Population</th>
 </tr>
 <sqloutput host=test
 query="SELECT country, population FROM
countries ORDER BY country">
 <tr>
 <td>#country#</td>
 <td>#population#</td>
 </tr>
 </sqloutput>
</table>

<sqloutput host=test
 query="SELECT population FROM countries WHERE
country='Sweden'">
 <set variable=swepop other=population>
</sqloutput>

The population of Sweden is <insert
variable=swepop>.
```

PROGRAMMING TAGS

Programming tags are tags that can be used for advanced RXML such as making web applications in RXML. There are also tags of interest to Challenger module programmers. And for the one who wants to combine programming with RXML there is the `<pike>` tag that lets you put pike code into your RXML pages.

<catch> ... </

`<catch>` is defined in the *Main RXML parser* module.

This tag does normally just pass along it's contents. However, in case there are an error in the RXML evaluation of the contents, or a `<throw>` tag is evaluated, only the error messages will be returned.

Example

```
<catch>
<h1>Hello World</h1>

<throw>Error dude.</throw>
</catch>
```

Results in

```
Error dude.
```

<cgi>

`<cgi>` is defined in the *CGI executable support* module.

Executes a CGI script, any attributes is forwarded from the tag to the CGI script. The same can be achived bu the `<insert>` tag or SSI `<-- #exec -->`, but the `<cgi>` tag has a nicer syntax.

script
The CGI script to invoke.

default-*argument*
This argument will be sent to the CGI script, unless a form variable exists with the same name.

<throw> ... </

`<throw>` is defined in the *Main RXML parser* module.

This tag throws an exception, with the enclosed text as the error message. The RXML parsing will stop at the `<throw>` tag.

Has a close relation to the `<catch>` tag.

Example

```
<catch>
 <set variable=foo value=Hi>
 <throw>Error dude.</throw>
 <set variable=foo value=Bye>
</catch>

<p><insert variable=foo>
```

Results in

```
Error dude.

Hi
```

<crypt>

`<crypt>` is defined in the *Main RXML parser* module.

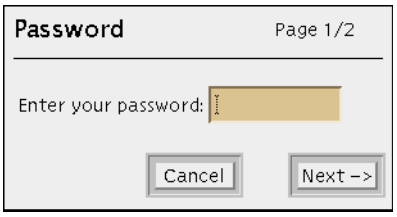
Encrypts the contents as a Unix style password. Useful when combined with services that use such passwords.

Unix style passwords are one-way encrypted, to prevent the actual clear-text password from being stored anywhere. When a login attempt is made, the password supplied is also encrypted and then compared to the stored encrypted password.

Example

```
<wizard name="Password">
<page>Enter your password:
<var name=password type=password size=10>
</page>
<page>Your encrypted password is
<tt><crypt><insert var=password></crypt></
tt>.
</page>
</wizard>
```

Results in



<debug>

<debug> is defined in the *Main RXML parser* module.

Sets debugging on or off. When debugging is on many RXML tags will output more detailed error messages. It is equivalent to giving the *debug* attributes to those tags.

on
Enables debug mode.

off
Disables debug mode.

Example

```
With debug:
<br><debug on>
<append variable=foo from=bar>

<p>Without debug:
<br><debug off>
<append variable=foo from=bar>
```

Results in

```
With debug:
Append: from variable doesn't exist

Without debug:
```

<default>

`<default>` is defined in the *Main RXML parser* module.

Makes it easier to give default values to `<select>` or `<checkbox>` form elements.

The `<default>` container tag is placed around the form element it should give a default value.

This tag is especially useful in combination with database tags.

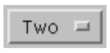
value=*string*
The value to set.

name=*string*
Only affect form element with this name.

Example

```
<form>
 <default value=2 name=number>
 <select name=number>
 <option value="1">One
 <option value="2">Two
 <option value="3">Three
 </select>
 </default>
</form>
```

Results in



<for> ... </

<for> is defined in the *Main RXML parser* module.

Makes it possible to create loops in RXML.

from
Initial value of the loop variable.

step
How much to increment the variable per loop iteration. By default one.

to
How much the loop variable should be incremented to.

variable
Name of the loop variable.

Example

```
<gauge>
 <for variable=i from=1 to=5>
 <set variable=foo from=i>
 </for>
</gauge>
```

Results in

```
Time: 0.003869 seconds
```

<gauge>

<gauge> is defined in the *Main RXML parser* module.

<gauge> measures how much CPU time is takes to run its contents through the RXML parser.

Example

```
<gauge>
 <for variable=i from=1 to=5>
 </for>
</gauge>

<gauge>
 <for variable=i from=1 to=50>
 </for>
</gauge>

<gauge>
 <for variable=i from=1 to=500>
 </for>
</gauge>
```

Results in

```
Time: 0.002087 seconds

Time: 0.016007 seconds

Time: 0.159013 seconds
```

<nooutput> ... </

<nooutput> is defined in the *Main RXML parser* module.

The contents will not be sent through to the page. Side effects, for example sending queries to databases, will take effect.

Example

```
<set variable=foo value=Hi>

<nooutput>
 <h1>Hi dude</h1>
 <set variable=foo value=Bye>
</nooutput>

<p><insert variable=foo>
```

Results in

```
Bye
```

<noparse> ... </

<noparse> is defined in the *Main RXML parser* module.

The contents of this container tag won't be RXML parsed.

Example

```
<use package=gtext_headers>

<h1>Hello</h1>

 <h1>World</h1>
```

Results in

Hello

World

<pike> ... </

<pike> is defined in the *Pike tag* module.

Runs the content as Pike code. This tag is not always available, since it can be a security hazard.

Example

```
<gtext><pike>
 string a;
 a = "Hello";
 a += " World";
 return a;
</pike></gtext>
```

Results in

Hello World

<random>

<random> is defined in the *Main RXML parser* module.

Randomly chooses a message from its contents.

separator=*string*
The separator used to separate the messages, by default newline.

Example

```
<cset preparse variable=num><random
>1
2
3
4
5</random></cset>

Your random number is <formoutput><number
num=#num#></formoutput>.
```

Results in

Your random number is two.

<realfile>

<realfile> is defined in the *Main RXML parser* module.

Prints the path to the file containing the page in the computers file system, rather than Challenger's virtual file system, or *unknown* if it is impossible to determine.

Example

```
<realfile>
```

Results in

unknown

<scope> ... </

<scope> is defined in the *Main RXML parser* module.

Creates a new scope for RXML variables. Variables can be changed within the <scope> tag without having any effect outside it.

extend
Copy all variables from the outer scope.

Example

```
<set variable=foo value="World">

<scope>
<h1>Hello <insert variable=foo></h1>
<set variable=foo value="Duck">
</scope>

<scope extend>
<h1>Hello <insert variable=foo></h1>
</scope>
```

Results in

Hello

Hello World

<sed> ... </

<sed> is defined in the *SED module* module.

Emulates a subset of *sed* operations in RXML. (*Sed* is the Unix "Stream EDitor" program which edits a stream of text according to a set of instructions.)

append

chars

lines

prepend

split=<*linesplit*>

suppress

```
Syntax :

<sed [suppress] [lines] [chars]
[split=<linesplit>]
 [append] [prepend]>
 <e [rxml]>edit command</e>
 <raw>raw, unparsed data</raw>
 <rxml>data run in rxml parser before edited</
rxml>
 <source variable|cookie=name [rxml]>
 <destination variable|cookie=name>
 </sed>

edit commands supported:
 ,
 ^^ numeral (17) ^^
 or relative (+17, -17)
 or a search regexp (/regexp/)
 or multiple (17/regexp//regexp/+2)

 D - delete first line in space
 G - insert hold space
 H - append current space to hold space
 P - print current data
 a - insert
 c - change current space
 d - delete current space
 h - copy current space to hold space
 i - print string
 l - print current space
 p - print first line in data
 q - quit evaluating
 s/regexp/with/x - replace
 y/chars/chars/ - replace chars

 where line is numeral, first line==1
```

<strlen> ... </

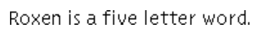
<strlen> is defined in the *Main RXML parser* module.

Returns the length of the contents.

Example

```
<cset variable=num preparse>
<strlen>Roxen</strlen>
</cset>

Roxen is a <formoutput><number num=#num#></
formoutput> letter word.
```

Results in



Roxen is a five letter word.

<trace>

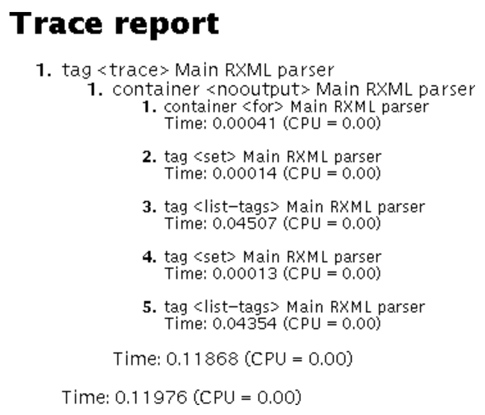
<trace> is defined in the *Main RXML parser* module.

Makes a trace report about how the contents are parsed by the RXML parser.

Example

```
<trace>
 <nooutput>
 <for variable=i from=1 to=2>
 <list-tags>
 </form>
 </nooutput>
</trace>
```

Results in



Trace report

```
1. tag <trace> Main RXML parser
    1. container <nooutput> Main RXML parser
        1. container <for> Main RXML parser
           Time: 0.00041 (CPU = 0.00)

        2. tag <set> Main RXML parser
           Time: 0.00014 (CPU = 0.00)

        3. tag <list-tags> Main RXML parser
           Time: 0.04507 (CPU = 0.00)

        4. tag <set> Main RXML parser
           Time: 0.00013 (CPU = 0.00)

        5. tag <list-tags> Main RXML parser
           Time: 0.04354 (CPU = 0.00)

       Time: 0.11868 (CPU = 0.00)

   Time: 0.11976 (CPU = 0.00)
```

<vfs>

<vfs> is defined in the *Main RXML parser* module.

Prints the mountpoint of the filesystem module that handles the page, or *unknown* if it could not be determined. This is useful for creating pages or applications that are to be placed anywhere on a site, but for some reason have to use absolute paths.

Example

```
<set variable=path eval="<vfs>">

<formoutput>
<a href='#path#/Challenger1.2/User/
programming/vfs.html'>
Link to this page</a>.
</formoutput>
```

<wizard> ... </

<wizard> is defined in the *Wizard generator* module.

The <wizard> tag generates wizard-like user interfaces, where the user is guided through several pages of controls. It is very useful for making web applications in RXML.

The <wizard> tag must contain at least one <page> page container tag. The <page> tag can in turn contain <var> tags or <cvar> container tags.

cancel=*URL*
The URL to go to when the *cancel* button is pressed.

cancel-label=*string*
The text on the *cancel* button.

done=*URL*
The URL to go to when the *done* button is pressed.

name=*string*
The title of the wizard.

next-label=*string*
The text on the *next* button.

ok-label=*string*
The text on the *ok* button.

page-label=*text*
The text *Page* in the upper right corner.

previous-label=*text*
The text on the *previous* button.

Attributes for <var> and <cvar>

cols=*number*
Sets the number of columns.

default=*value*
The default value.

name=*name*
The name of the variable.

options=*option1,option2,...*
Available for *select* or *select_multiple* variables.

rows=*number*
Sets the number of rows.

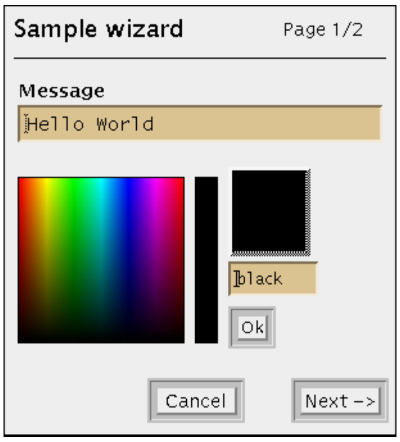
size=*number*
Sets the size or the input form.

type=string password list text radio checkbox int float color color-small font toggle select select_multiple
The variable type.

Example

```
<wizard cancel="wizard.html" ok-label="Done"
done="wizard.html" name="Sample wizard">

<page>
<b>Message</b>
<var name=message size=30
 value="Hello World">

<p><var name=color type=color-small>
</page>

<page>
<formoutput>
<gtext fg=#color#>#message#</gtext>
</formoutput>
</page>

</wizard>
```

Results in


Sample wizard Page 1/2

Message

Hello World

black

SUPPORTS SYSTEM

The supports system makes it possible to use features that are only supported by a few browsers and still be compatible with all browsers. This is done through a database of capablities supported by the different browsers. The `<if>` is then used on the pages to make versions that use different browser capabilities.

Pages are not customized for a certain browser, but rather for browsers that support different features. When a new browser is released, all that is necessary is to determine what features it supports. Once that has been done, and the database updated, all pages using the support system will work with it.

Some features might work to a lesser degree on some browsers. Old versions of the Macintosh version of Netscape supports JavaScripts, but some JavaScripts make the Netscape browser hang. If you have such JavaScripts, you would probably want the support system to make sure they are not sent to that version of Netscape. On the other hand, if you have less complicated JavaScripts you will probably want to send them.

To make the supports system work for you, you might need to tweak it yourself. This can be done by the Challenger administrator by changing the *Global Variables/Client supports regexps* variable (you will have to choose *more options* to see it).

Since new browsers get released all the time, updated versions of the supports database are by default fetched regularly from www.roxen.com.

Features

List of the available features:

backgrounds
The browser supports backgrounds according to the HTML3 specifications.

bigsmall
The browser supports the `<big>` and `<small>` tags.

center
The browser supports the `<center>` tag.

cookies
The browser can receive cookies.

divisions
The browser supports `<div align=...>`.

font
The browser supports `<font size=num>`.

fontcolor
The browser can change color of individual characters.

fonttype
The browser can set the font.

forms
Thr broswer supports forms according to the HTML 2.0 and 3.0 specifications.

frames
The browser supports frames.

gifinline
The browser can show GIF images inlined.

imagealign
The browser supports *align=left* and *align=right* in images.

images
The browser can display images.

java
The browser supports Java applets.

javascript
The browser supports Java Scripts.

jpeginline
The browser can show JPEG images inlined.

mailto
The browser supports mailto URLs.

math
The `<math>` tag is correctly displayed by the browser.

perl
The browser supports Perl applets.

pjpeginline
The browser can handle progressive JPEG images, `.pjpeg`, inline.

pnginline
The browser can hangle PNG images inlined.

pull
The browser handles Client Pull.

push
The browser handles Server Push.

python
The browser supports Python applets.

robot
The request really comes from a search robot, not an actual browser.

stylesheets
The browser supports stylesheets.

supsub
The browser handles `<sup>` and `<sub>` tags correctly.

tables
The browser handles tables according to the HTML3.0 specification.

tcl
The browser supports TCL applets.

vrml
The browser supports VRML.

File Syntax

By default, the supports database is located in the file `server/etc/supports` which is updated automatically from www.roxen.com.

The `server/etc/supports` file should not be edited directly, since that might interfere with the automatic updates. If you need to tweak the supports database it is better to create your own local supports file, and change the *Global Variables/Client supports regexps* variable (you will have to choose *more options* to see this variable).

The syntax used is:

`patternfeature, -feature, ...`

If the regular expression *pattern* matches the name of the client, all features will be added to the list of features handled by the client. If '-' is prefixed to the name of the feature, it will be removed instead.

\ can be used to escape newlines.

If a line starts with '#', it is skipped, unless it is:

`#include <path>`

which means include the contents of that file here

`#define fromto`

which means replace all occurrances of the word *from* with *to*

or

```
#section pattern {
...
# }
```

which is used to speed up parsing. If the name of the client matches *pattern* it will go through the section. If the pattern doesn't match the entire section will be skipped.

SSI

SSI, Server Side Includes, are similar to RXML tags and have the advantage of being a standard supported by many web servers. It is thus possible to write pages using SSI that are portable to other web servers.

The downside is that SSI is in no way as flexible or powerful as RXML. The tags are placed within HTML comments, which makes it impossible to combine different SSI tags. However, it is possible to combine SSI tags with regular RXML tags.

Challenger does not implement all the SSI functionality that Apache supports.

<!--#config-->

`<!--#config-->` is defined in the *Main RXML parser* module.

The config command is used to configure how things should be printed.

errmsg=*string*
Where `msg` is a message that is sent back to the client if an error occurs while parsing the document.

sizefmt=bytes abbrev
The value sets the format to be used when displaying the size of a file. *Bytes* gives a count in bytes while *abbrev* gives a count in Kb or Mb, as appropriate.

timefmt=*value*
The value is a string to be used when printing dates.

<!--#echo-->

`<!--#echo-->` is defined in the *Main RXML parser* module.

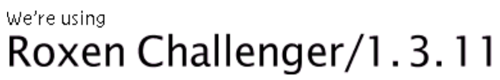
Prints a variable from the server or request.

var=sizefmt document name path translated document uri date local date gmt query string unescaped last modified server software server name gateway interface server protocol server port request method remote host remote addr auth type remote user http cookie cookie http accept http user agent http referer
The variable to print.

Example

```
We're using
<gtext><!--#echo var="server software"--></
gtext>
```

Results in


We're using
Roxen Challenger/1.3.11

<!--#exec-->

`<!--#exec-->` is defined in the *Main RXML parser* module.

Executes a CGI script or shell command. This command has security implications and therefore, might not be available on all web sites.

cgi=*URL*
Path to the CGI script URL encoded. That is, a character can be quoted by % followed by its hex value. The CGI script is given the PATH_INFO and QUERY_STRING of the original request from the client. The variables available in `<!--#echo>` will be available to the script in addition to the standard CGI environment. If the script returns a Location header, then this will be translated into an HTML anchor.

cmd=*path*
The server will execute the command using `/bin/sh`. The variables available in `<!--#echo>` will be available to the script.

<!--#flastmod-->

`<!--#flastmod-->` is defined in the *Main RXML parser* module.

Prints the last modification date of the specified file.

file=*path*
Path to the file.

virtual=*URL*
Path to the file URL encoded. That is, a character can be quoted by % followed by its hex value.

<!--#fsize-->

`<!--#fsize-->` is defined in the *Main RXML parser* module.

Prints the size of the specified file, subject to the sizefmt format specification.

file=*path*
Path to the file.

virtual=*URL*
Path to the file URL encoded. That is, a character can be quoted by % followed by its hex value.

<!--#include-->

`<!--#include-->` is defined in the *Main RXML parser* module.

Insert a text from another file into the page.

file=*path*
The file as a path relative to the directory containing the current page. It cannot contain `../`, nor can it be an absolute path.

virtual=*URL*
The path to the file URL encoded. That is, a character can be quoted by % followed by its hex value. The path may contain `../` and may be absolute, ie starting with a `/`

HTACCESS

htaccess is a system for handling access control to your pages. It works by placing an `.htaccess` file in a directory, which contains the access control lists for that directory. It is possible to get fine grained security and to configure exactly who can view which pages. The *.htaccess support* module must be enabled for htaccess to work.

It is possible to distinguish users either by the IP address or domain name of their computer or by letting them authentificate with a user name and password. The user name and password is by default compared via an Authentification module. That usually means that users are authentificated by the operating systems authentification system.

If you want to have password protected pages usable by users that are not handled by the current Authentification module you can create your own database of users by creating `.htpasswd` and `.htgroup` files.

htaccess is a standard supported by many web servers. The access control you have built with htaccess should therefore be portable to other web servers.

.htaccess

A `.htaccess` file consists of lines containing directives. Apart from the `Limit`; directive, all directives have the form

`directiveargument(s)`

where *argument(s)* is one or more arguments. The directives supported are:

AuthUserFile
Use this user and password file to authentificate users. Typically, the AuthUserFile is called `.htpasswd`

AuthGroupFile
Use this group file, which contains a database of which groups users are member of. Typically, the AuthGroupFile is called `.htgroup`, if used.

AuthName
Set the authentication realm, which can be any name you choose. The name will be used to tell browsers how to *label* user authentications within a session, so that the browsers can automatically repeat passwords the user has already entered when accessing new pages with the same access requirements.

Redirect
Redirect all accesses for pages in the directory to this URL.

ErrorFile
Show this page in case the requested page could not be found, maybe because the user did not have permission to view it.

Then there is the `<Limit>` container tag. The attributes are the HTTP method(s) that access should be limited to, *GET*, *PUT*, *POST* or *HEAD*. The contents of the tag are access control directives, one directive on each line. Possible directives are:

allow from *address*
deny from *address*
Allow or deny access to users from a DNS domain or IP number. `www.roxen.com` means the computer while `.roxen.com` means all computers on the domain *roxen.com*. The same way `194.52.202.3` means the computer while `194.52.` means the net starting with *194.52*

require user *user(s)*
require group *group(s)*
Allow access only for the named user(s) or group(s).

require valid-user
Allow access to any user present in the AuthUserFile or Authentification module.

satisfy all
satisfy any
Decide what happens if both *require* and *allow* rules are present; *all* indicates that the user must satisfy both kinds of requirements, while *any* means that it is enough that the user satisfies either kind.

order deny,allow
order allow,deny
order mutual-failure
The *order* rules decides how to prioritize deny and allow rules. If the order is set to *deny,allow*, deny rules will be processed before allow rules. With *allow,deny*, allows will be processed before denies, and with *mutual-failure*, hosts allowed by any *allow* rule will be allowed, and other hosts denied. *Deny,allow* is the default.

The rule evaluation does not stop until all rules have been processed, so the earlier a rule is processed, the lower priority is has in determining access. This only matters when different rules contradict each other, for instance when a wide-ranging deny rule forbids access to a certain domain, and an allow grants access to a smaller part of the domain.

Example
A typical `.htaccess` file would look something like this:

```
AuthUserFile /home/frotz/.htpasswd
AuthGroupFile /home/frotz/.htgroup
AuthName MyTestDomain
AuthType Basic

<Limit PUT HOST HEAD>
require user frotz
</Limit>

<Limit GET>
allow from all
</Limit>
```

The `.htaccess` file above would allow everyone to GET documents in the directory, but all other kinds of access would be restricted to the user frotz, and expect this user to login with the password listed for that user in the `.htpasswd` file in the user frotz's home directory.

.htpasswd

The format of the password file is straightforward, one line per user, with the line containing the user name, followed by a colon, followed by the user's password encrypted with the standard Unix password encryption. The `<crypt>` tag can be used to encrypt such a password.

In other words, an `.htpasswd` can look like this:

```
frotz:taeWr6tbTZKO6
gnusto:jKXVnZH6eXR7
```

with one line for each user.

.htgroup

The format of the group file is straightforward, one line per group, with the line containing the group name, followed by a colon, followed by the users in the group. Users are separated by commas.

In other words, an `.htgroup` file can look like this:

```
all:frotz,gnusto
admins:frotz
```

with one line per group.