

RSP Markup Language - Reference Card

An RSP document consists of text with RSP-embedded markup. When *compiled*, independently of programming language, (i) comments are dropped, (ii) preprocessing directives are processed, and (iii) text and code expressions are translated into a code script. The translated code script can then be (iv) evaluated, which generates the output document, which in turn may be (v) postprocessed. [The R.rsp package knows how to postprocess output such as TeX, Markdown, Sweave, knitr etc.] Examples (in R): (1) main.tex.rsp → (main.tex.R) → main.tex → main.pdf. (2) main.md.rsp → (main.md.R) → main.md → main.html. (3) main.Rnw.rsp → (main.Rnw.R) → main.Rnw → main.tex → main.pdf.

Comments, Trimming & Escapes

Comments can be used to exclude text, code expressions and preprocessing directives.

Markup	Description
<code><!--(anything)--></code>	Drops (anything). Number (≥ 2) of hyphens must match. Comments can be nested, if different number of hyphens.
<code><%-></code> , <code><!--></code> , ...	“Empty” comments. Like above comments, these ones force following white space and line break to be dropped.
<code><% ... -></code> , <code><% ... +></code>	A hyphen (plus) attached to the end tag, forces following white space (including the line break) to be dropped (kept).
<code><%</code> and <code>%></code>	Inserts <code><%</code> and <code>%></code> .

Preprocessing directives

Preprocessing directives are independent of programming language used. They are applied after dropping comments and before translating text and code expressions to a code script. It is not possible to tell from the translated code script whether preprocessing directives have been used or not, nor are their variables accessible (except metadata).

Markup	Description
<code><%@include file="⟨file URL⟩"%></code>	Inserts the content of file ⟨file URL⟩ into the document before RSP-to-script translation.
<code><%@meta ⟨name⟩="⟨content⟩"%></code>	Assigns ⟨content⟩ to metadata variable ⟨name⟩. Metadata may be used by preprocessors, e.g. including HTML title.
<code><%@meta name="⟨name⟩"%></code>	Inserts the content of metadata variable ⟨name⟩.
<code><%@⟨type⟩ ⟨name⟩="⟨content⟩"%></code>	Assigns ⟨content⟩ to preprocessing variable ⟨name⟩ of type ⟨type⟩. Supported types are ‘string’, ‘numeric’, ‘integer’ and ‘logical’.
<code><%@⟨type⟩ name="⟨name⟩"%></code>	Inserts the content of preprocessing variable ⟨name⟩.
<code><%@ifeq ⟨name⟩="⟨content⟩"%></code> <code>⟨incl⟩ <%@else%> ⟨excl⟩ <%@endif%></code>	If preprocessing variable ⟨name⟩ <i>equals</i> ⟨content⟩, then ⟨incl⟩ is inserted otherwise ⟨excl⟩. <code><%@else%></code> is optional.
	<code><%@ifneq ...%></code> negates the test.

Code expressions

Code expressions are evaluated *after* translation. They may be of any programming language as long as there is a code translator for it. Code expressions have no access to preprocessing variables (except metadata). Output written to standard output is inserted into the final document.

Markup	Description
<code><%⟨code⟩%></code>	Inserts ⟨code⟩ (may be an incomplete expression) into the translated code script without including content in the output document.
<code><%=⟨code chunk⟩%></code>	Inserts ⟨code chunk⟩ (must be a complete expression) into the translated code script and includes the returned value in the output document.

Example of text file with RSP-embedded R code

1. RSP document:

```
<%@meta title="Example"%>
Title: <%@meta name="title"%>
Counting:<% for (i in 1:3) { %> <%=i-%>
  <%=i-%>
}<% } %>
```

2. Without comments and preprocessed:

```
Title: Example
Counting:<% for (i in 1:3) { %> <%=i-%>
<% } %>
```

3. Translated code script:

```
cat("Title: Example\nCounting:")
for (i in 1:3) {
  cat(" ")
  cat(i)
}
```

4. Output document:

```
Title: Example
Counting: 1 2 3
```

R.rsp commands

```
rconcat('Today is <%=Sys.Date()%>')      s <- rstring('Today is <%=Sys.Date()%>')      output <- rfile('⟨file|URL⟩')              rsource('⟨file|URL⟩')
rconcat(file='⟨file|URL⟩')                 s <- rstring(file='⟨file|URL⟩')                 output <- rfile('⟨file|URL⟩', postprocess=FALSE)
```