

---

*Enhydra*

*Jolt Syntax Reference Guide*

*version 2.0*

**Developed by Lutris Technologies, Inc.**

---

---

TABLE OF CONTENTS

---

<b>CHAPTER 1</b>	<i>Overview of Enhydra Jolt</i> <b>1</b>
	Structuring an Enhydra Jolt Presentation <b>2</b>
	Enhydra Jolt Compiler Options <b>2</b>
	Executing a Presentation Object <b>3</b>
<b>CHAPTER 2</b>	<i>Enhydra Jolt Fields</i> <b>5</b>
	Accessing Enhydra Jolt Fields from JavaScript <b>6</b>
	Dumping Known Field Names <b>7</b>
<b>CHAPTER 3</b>	<i>Enhydra Jolt Tags</i> <b>9</b>
	Tests for Conditions <b>9</b>
	The <JOLT JAVADEF> Tag <b>10</b>
	The <JOLT HTMLDEF> Tag <b>12</b>
	The <JOLT JAVACALL> Tag <b>12</b>
	<i>Passing Arguments Using &lt;JOLT JAVACALL&gt;</i> <b>13</b>
	<i>Using Conditions Within &lt;JOLT JAVACALL&gt;</i> <b>14</b>
	The <JOLT CALL> Tag <b>16</b>
	The <JOLT HTMLCALL> Tag <b>16</b>
	The <JOLT HTML> Tag <b>16</b>
	The <JOLT JAVAIMPORT> Tag <b>18</b>
	The <JOLT INCLUDE> Tag <b>19</b>
	The <JOLT JAVACATCH> Tag <b>21</b>
	The <JOLT JAVAFINALLY> Tag <b>22</b>
<b>APPENDIX A</b>	<i>Glossary of Terms</i> <b>23</b>

---

**TABLE OF CONTENTS**

---

---

Enhydra Jolt enhances the static presentation capabilities of HTML, as it is designed to support the systematic embedding of Java functionality within an HTML page. Enhydra Jolt files, like HTML files, are simple ASCII files; Enhydra Jolt files are distinguished from standard HTML files by their “.jhtml” extension.

Within a Enhydra Jolt file, Java code supplies dynamic content, while a non-programmer can easily edit existing static content with any best-of-breed web authoring tool. The Enhydra Jolt syntax allows for the integration of Java with HTML, or the modular separation of HTML templates and Java libraries into separate files.

There are two categories within the Enhydra Jolt syntax: `<JOLT>` tags and Enhydra Jolt Fields. The `<JOLT>` tags, with their various attributes, isolate Java sections and conditionally insert static HTML content. Enhydra Jolt Fields support the ability to decode URL arguments, or to embed values directly from a Java object.

Enhydra Jolt files are the building blocks of Presentation Objects, which are compiled using the Enhydra Joltc compiler. These Presentation Objects are then executed from any web server supported by Enhydra.

---

## *Structuring an Enhydra Jolt Presentation*

[structure, ideas, direction? Or, just an example file that I can walk though and see what's going on and explain to others.]

---

## *Enhydra Joltc Compiler Options*

The Enhydra Joltc compiler, with the following commands, compiles a JHTML file:

```
joltc [options] src.jhtml packageName
```

The `.jhtml` file extension is mandatory to the source file `src`. The resultant class belongs to the Java package name `packageName`.

The compiler supports the following command line options:

- |            |   |
|------------|---|
| -k         | Keep the resultant <code>.java</code> files. This is useful for debugging. Normally the <code>.java</code> files are automatically removed after successfully compiling the <code>.class</code> file. |
| -d destdir | Specify the destination root for the Java class files. The default is <code>javadir</code> .  |
| -j javadir | Directory in which to generate the <code>.java</code> files. The default is the current working directory.  |

When developing in the Enhydra Development Environment, the use of Enhydra Joltc is usually transparent, as the distributed makefiles contain the rules for running the Enhydra Jolt compiler.

---

### *Executing a Presentation Object*

A compiled Presentation Object consists of a class with the same name as the original JHTML file. A well-defined entry method, such as:

```
run(HTTPPresentationComms)
```

is automatically inserted into the Presentation Object by the Enhydra Joltc compiler. Presentation Objects can also be constructed by hand, without the use of the Enhydra Joltc compiler.

- 1 Upon receipt of a URL ending in `.po` (e.g., `demoApp.po`), Enhydra will turn the request into a `run()` method on the appropriate PO (e.g., `demoApp.class`).
- 2 The `run()` method will sequentially execute each part of an Enhydra Jolt Presentation Object (dynamic Java or static HTML) according to the conditional rules set forth by `<JOLT>` tags.
- 3 Execution continues until the Presentation Object ends naturally, an unhandled exception is thrown, or an HTTP redirect is invoked.

---

## Overview of Enhydra Jolt

---

---

The values of `page.data` variables, when used within HTML sections, are called Enhydra Jolt Fields. Enhydra Jolt Fields and their values result from two possible scenarios. First, when entering a page environment, Enhydra Jolt Fields initially represent decoded CGI arguments from an HTTP GET or POST method.

For example, the argument:

```
.../foo.po?firstName=Pete&lastName=Smith
```

creates the Enhydra Jolt Fields `cgiArgs.firstName` and `cgiArgs.lastName`.

The values contained in Enhydra Jolt Fields (in this example, “Pete” and “Smith”) may be accessed from within static HTML content by pre-pending and post-pending “(“ and “)”, respectively, such as:

```
<BR>Your last name is (@cgiArgs.lastName@).
```

This construct would generate the following string, displayed by the client:

```
Your last name is Smith.
```

Additionally, Enhydra Jolt Fields can be the result of an Enhydra Jolt Java section, created using the `set` method of the `page.data` class. Both the `page.data` variable and Enhydra Jolt Field name must be a valid Java identifier.

In the example below, this statement from an Enhydra Jolt Java section creates an Enhydra Jolt Field called `minLength`, with a value of 8:

```
page.data.set("minLength", "8");
```

which is then referenced within static HTML content:

```
<BR>The minimum required password length is (@minLength@).
```

and displays:

```
The minimum required password length is 8.
```

Enhydra Jolt Fields represent the hierarchical structure of `page.data`. The '.' (period) character is significant, as it delimits the branches within the `page.data` hierarchy. As new Enhydra Jolt Fields are created, the `page.data` object accumulates their variables and values throughout the lifetime of the page. If a referenced Enhydra Jolt Field does not exist, or is illegally specified, an exception will be thrown when executing the Presentation Object.

If a portion of HTML needs to include the (@ or @) characters literally, they must be quoted with the regular HTML quoting mechanism:

```
<BR>Quoting characters look like &#040;@@&#041;
```

This construct generates the string:

```
Quoting characters look like (@@)
```

**NOTE:** *If referencing a directory when utilizing Enhydra Jolt Fields, an HTML-formatted dump of all known Enhydra Jolt Field names and their values under the specified directory will be reported.*

---

## *Accessing Enhydra Jolt Fields from JavaScript*

Enhydra Jolt Fields are evaluated before the HTML results of a Presentation Object are sent as a response to the client. Therefore, Enhydra Jolt Fields are ideal for adding simple dynamic content to pages containing JavaScript (or any other client-side language).

---

## Dumping Known Field Names

---

Previously in this reference, an example of an Enhydra Jolt Java section stored a minimum password length value in an Enhydra Jolt Field called `minLength`. The following JavaScript uses this Enhydra Jolt Field to alert the user before the form is submitted:

```
<SCRIPT LANGUAGE="JavaScript">

    if (form.password.value.length < (@minLength@))

        alert("(@minLength@) characters are required")

</SCRIPT>
```

Using the example above, remember that the integer value “8” has been stored in the Enhydra Jolt Field `minLength`. The following events occur:

- 1 JavaScript is called on the client-side
- 2 The “if . . .” statement asks if the password input by the users contains less than the required number of characters as specified in `minLength`.
- 3 If the result is true (if the password does not contain the minimum number of characters), the browser displays an alert pop-up window. The value of `minLength` (“8”) is referenced in the pop-up, which displays “8 characters are required”.

---

## *Dumping Known Field Names*

Enhydra Jolt Fields with no inserted name (`@@`) automatically generate an HTML-formatted dump of all Enhydra Jolt Field names and their values known within the page context. This is particularly useful during application development and debugging.

---

**Enhydra Jolt Fields**

---

---

The `<JOLT>` tags within the Enhydra Jolt syntax serve numerous functions ranging from referencing files and Java methods to conditionally including HTML content.

Depending upon its attribute and/or conditions, a `<JOLT>` tag can include Java code or HTML content within an Enhydra Jolt file. Some `<JOLT>` tags are also provided to catch exceptions upon compiling or executing a Presentation Object, providing a programmer error-catching control that is superior to CGI-based applications.

Further in this chapter, as individual tags are described, the tags are referenced using their attribute as an identifier. For example, the `<JOLT>` tag with the `JAVADEF` attribute is referred to as the `<JOLT JAVADEF>` tag.

**NOTE:** *In the examples provided for each tag, quotation marks (“”) are used to surround arguments. These quotation marks are optional, unless the value contains a space.*

---

### *Tests for Conditions*

Conditions can be applied to `<JOLT>` tags, to determine if HTML content (including JavaScript and nested `<JOLT>` tags, where applicable), will be included in the

resultant HTML file. Within each tag, only a single conditional test can be performed. The range of test options are described the table below:

<i>Conditional Test</i>	<i>Behavior</i>
IFEQ	If the contents of <code>FIELD</code> equal <code>VALUE</code> , then true.
IFNEQ	If the contents of <code>FIELD</code> are not equal to <code>VALUE</code> , then true.
IFDEF	If the <code>FIELD</code> is defined, then true.
IFNDEF	If the <code>FIELD</code> is not defined, then true.
IFCALL	If a method returns true, then true. Must return a boolean value.
IFNCALL	If a method returns false, then true. Must return a boolean value.

In this example, the Java method `addNewColor` is invoked only if an Enhydra Jolt Field named `color` exists:

```
<JOLT JAVACALL=addNewColor IFDEF FIELD=color>
    ...HTML CONTENT...
</JOLT>
```

Applicable examples of tests for conditions are included later in this chapter.

---

### *The <JOLT JAVADEF> Tag*

The general syntax of the `<JOLT JAVADEF>` tag is as follows, and multiple `<JOLT JAVADEF>` tags may not be nested:

```
<JOLT JAVADEF>
    ...Java field, method or inner class declarations...
</JOLT>
```

The `JAVADEF` attribute allows for any regular Java declarations to be made, including field, method and inner class declarations. Once defined within a `<JOLT JAVADEF>` section, all fields and methods become part of the Presentation Object class.

---

## The <JOLT JAVADEF> Tag

---

The <JOLT JAVACALL> tag can then be used to call these pre-defined methods, as long as the methods take a single mandatory `JoltPage` argument.

For example:

```
<JOLT JAVADEF>

private static final String COMPANY="Enhydra";

void setJoltFields (JoltPage page)

    throws Exception

{

    // Create some Jolt Fields...

    page.data.set("mode.on", new Boolean(true));

    page.data.set("company", COMPANY);

    ... more Java code...

}

</JOLT>

<JOLT JAVACALL="setJoltFields"></JOLT>

<BR>The company name is (@company@)

<BR>The mode is set to (@mode.on@)
```

The above <JOLT JAVADEF> declaration defines two fields (`company` and `mode`), as well as a method (`setJoltFields`) then called by the <JOLT JAVACALL> tag.

An HTML-defined method can be called directly from Java by using the `call(methodName)` method from the `JoltPage` class. This is an overloaded method allowing an optional `KeywordValueTable` to be layered on the `page.data` scope prior to calling the method.

For example:

```
page.call(myMethod, myArguments);
```

**NOTE:** Multiple <JOLT JAVADEF> tags are legal. Their contents are

*concatenated in the order they are defined.*

---

### *The <JOLT HTMLDEF> Tag*

The general syntax of the <JOLT HTMLDEF> tag is as follows, and multiple <JOLT HTMLDEF> tags may be nested:

```
<JOLT HTMLDEF="methodName">
    ...HTML CONTENT...
</JOLT>
```

The HTMLDEF attribute allows for a block of HTML to be defined and subsequently referenced by name, creating an environment for the modular development of code. The <JOLT JAVACALL> tag can then be used to call methods.

**NOTE:** *Methods defined by the HTMLDEF attribute and Java methods defined by the JAVADEF attribute are accessed in exactly the same manner.*

---

### *The <JOLT JAVACALL> Tag*

The general syntax of the <JOLT JAVACALL> tag is as follows, and multiple <JOLT JAVACALL> tags may be nested:

```
<JOLT JAVACALL="methodName" ARG.field1="value1"
ARG.field2="value2">
    ...HTML CONTENT...
</JOLT>
```

**NOTE:** *The CALL and JAVACALL attributes are functionally equivalent. JAVACALL is included for backwards compatibility.*

---

## The <JOLT JAVACALL> Tag

---

The `CALL` attribute instructs the Presentation Object to call the specified method, identified by `methodName`. Usually the method is declared within the JHTML file and either applies HTML to the output or sets Enhydra Jolt Fields.

However, any method may be called by importing the referred class or fully qualifying the method name. For example, Enhydra includes a utility class called `JoltDebug` that is automatically included by the Enhydra Joltc compiler.

In this example, the `getRequest` method dumps all the information about the request in a pre-formatted manner:

```
<JOLT JAVACALL="JoltDebug.getRequest">
    ...HTML CONTENT...
</JOLT>
```

An example of a fully qualified method would be:

```
<JOLT JAVACALL="com.lutris.jolt.Utils.exampleMethod">
    ...HTML CONTENT...
</JOLT>
```

### Passing Arguments Using <JOLT JAVACALL>

When passing arguments to a method, the `ARG.` prefix is removed and the fields are then accessible to Java-defined methods or Enhydra Jolt Fields in HTML-defined methods. For example:

```
<JOLT JAVACALL="exampleMethod" ARG.first="Pete"
ARG.last="Smith">
</JOLT>
```

This examples makes `exampleMethod.first` (the value is "Pete") and `exampleMethod.last` (the value is "Smith") accessible.

Optionally, valid HTML content may be present and delimited by the closing `</JOLT>` tag. In this case, the text is made available to the method in a page variable named `tagContents`. This can be used to good effect for creating a library of HTML formatting routines. For example:

```
<JOLT JAVADEF>

    void addColor (JoltPage page)

        throws Exception

    {

        // -> page.append() is used to write HTML.

        page.append("<BR><FONT COLOR=\"blue\">");

        page.append(page.tagContents);

        page.append("</FONT>");

    }

</JOLT>

...

<JOLT JAVACALL="addColor">

    Color me blue!!!

</JOLT>
```

In this example, `addColor` is a Java method that returns the content text (“Color me blue!!!”) as an HTML string with additional tags:

```
<FONT COLOR="blue">Color me blue!!!</FONT>
```

The variable `page.tagContents` is available within the scope of the method.

### Using Conditions Within <JOLT JAVACALL>

Following are ways that method invocation can be made conditional within a <JOLT JAVACALL> tag. For a quick look at the list of conditional tests, please see the table on page 10.

In the example below, using the `IFEQ` conditional attribute, the method `methodName` is invoked if the value of the variable called `fieldName` is equal to value `value`. If `fieldName` does not exist then the condition is not true and `methodName` is not called, regardless of `value`:

---

## The <JOLT JAVACALL> Tag

---

```
<JOLT JAVACALL="methodName" IFEQ FIELD="fieldName"
VALUE="value">

  ...HTML CONTENT...

</JOLT>
```

Similarly, using the `IFNEQ` conditional attribute, method `methodName` will be invoked if the value of the variable called `fieldName` is *not* equal to value `value`. If `fieldName` does not exist then the condition is true and `methodName` is called, regardless of `value`:

```
<JOLT JAVACALL="methodName" IFNEQ FIELD="fieldName"
VALUE="value">

  ...HTML CONTENT...

</JOLT>
```

Below, using the `IFDEF` conditional attribute, the method `methodName` is invoked simply if the variable called `fieldName` exists:

```
<JOLT JAVACALL="methodName" IFDEF FIELD="fieldName">

  ...HTML CONTENT...

</JOLT>
```

Using `IFNDEF`, the method `methodName` is invoked if the variable called `fieldName` does *not* exist:

```
<JOLT JAVACALL="methodName" IFNDEF FIELD="fieldName">

  ...HTML CONTENT...

</JOLT>
```

Here, using `IFCALL`, method `methodName` is invoked if `testMethod` returns a boolean true.

```
<JOLT JAVACALL="methodName" IFCALL="testMethod">
  ...HTML CONTENT...
</JOLT>
```

Similarly, using `IFNCALL`, the method `methodName` is invoked if `testMethod` returns a boolean false.

```
<JOLT JAVACALL="methodName" IFNCALL="testMethod">
  ...HTML CONTENT...
</JOLT>
```

**NOTE:** *Complex conditionals can also be achieved by nesting a `<JOLT JAVACALL>` tag within conditional `<JOLT HTML>` tags.*

---

### *The `<JOLT CALL>` Tag*

This tag is identical to the `<JOLT JAVACALL>` tag.

---

### *The `<JOLT HTMLCALL>` Tag*

This tag is identical to the `<JOLT JAVACALL>` tag, with the exception that additional arguments are not allowed.

---

### *The `<JOLT HTML>` Tag*

The general syntax of the `<JOLT HTML>` tag is as follows, and multiple `<JOLT HTML>` tags may be nested:

```
<JOLT HTML CONDITION_SYNTAX>
```

---

## The <JOLT HTML> Tag

---

```
...HTML CONTENT...  
  
</JOLT>
```

Similar to calling methods using the <JOLT JAVACALL> tag, when using the <JOLT HTML> tag, content will be included if the `FIELD` variable content is equal to the value associated with `VALUE`.

In the following example of using the conditional `IFEQ` with the <JOLT HTML> tag, if `mode.on` contains the value `"true"`, then the HTML content will be included in the Presentation Object by the Enhydra Joltc compiler.

```
<JOLT HTML IFEQ FIELD="mode.on" VALUE="true">  
  
  <BR>You are seeing this because  
  
  <BR>you are in verbose mode.  
  
</JOLT>
```

Using `IFNEQ` in the example below, if the field `fieldName` does *not* exist then the condition is *not* true and the HTML is *not* included, regardless of `value`.

```
<JOLT HTML IFNEQ FIELD="fieldName" VALUE="value">  
  
...HTML CONTENT...  
  
</JOLT>
```

However, also using `IFNEQ` in this example, if the `fieldName` does not exist, the condition becomes true and the HTML content is included regardless of `VALUE`:

```
<JOLT HTML IFNEQ FIELD="mode.on" VALUE="true">  
  
  <BR>You have chosen non-verbose mode.  
  
</JOLT>
```

In the example below, using `IFDEF`, HTML content will be included simply if the variable called `fieldName` exists:

```
<JOLT HTML IFDEF FIELD="fieldName">  
  
...HTML CONTENT...  
  
</JOLT>
```

```
</JOLT>
```

Similarly, using `IFNDEF`, HTML content will be included using the example below, simply if the variable called `fieldName` does *not* exist:

```
<JOLT HTML IFNDEF FIELD="fieldName">
    ...HTML CONTENT...
</JOLT>
```

In this example, using `IFCALL`, HTML content will be included if the Java method named `testMethod` returns true.

```
<JOLT HTML IFCALL="testMethod">
    ...HTML CONTENT...
</JOLT>
```

Similarly, using `IFNCALL`, HTML content will be included if the Java method named `testMethod` returns false.

```
<JOLT HTML IFNCALL="testMethod">
    ...HTML CONTENT...
</JOLT>
```

**NOTE:** *Complex conditionals can also be achieved by nesting `<JOLT HTML>` tags.*

---

### *The `<JOLT JAVAIMPORT>` Tag*

The general syntax of the `<JOLT JAVAIMPORT>` tag is as follows, and multiple `<JOLT JAVAIMPORT>` tags may not be nested:

```
<JOLT JAVAIMPORT>
    ...Java import statements...
</JOLT>
```

---

## The <JOLT INCLUDE> Tag

---

The `JAVAIMPORT` attribute is used to delimit one or more Java import statements. Since the Java language mandates that imports are included at the head of a Java file, using this tag ensures this condition. Multiple Enhydra Jolt import tags can be declared anywhere in the JHTML file.

**NOTE:** Imports can be included in simple `<JOLT JAVADEF>` tags, as long as this section is the first in the file. However, the use of this feature is not recommended.

---

## The <JOLT INCLUDE> Tag

The general syntax of the `<JOLT INCLUDE>` tag is as follows, and multiple `<JOLT INCLUDE>` tags may be nested:

```
<JOLT INCLUDE="RelativeFilePath">
</JOLT>
```

The `INCLUDE` attribute provides a convenient method of including content from one file within another. The named file is compiled into the page as if it were in-line code. Either Enhydra Jolt files or standard HTML files can be included, using the `<JOLT INCLUDE>` tag.

An HTML file, for instance, might be included as a copyright footer or common header. The included file may include additional `<JOLT>` tags and access Enhydra Jolt Fields contained within the current page context. The `</JOLT>` end-tag is especially important when using the `<JOLT INCLUDE>` tag, as the included file cannot cross the boundary of the current page.

By convention, included files use the `.jinc` extension. The example below includes the file `CommonFooter.jinc`, located in the directory above the current directory of the JHTML file.

```
<JOLT INCLUDE="../../CommonFooter.jinc">
</JOLT>
```

The `<JOLT INCLUDE>` and `<JOLT JAVADEF>` tags can be used effectively to split otherwise large JHTML files into a number of manageable pieces. This approach

---

## Enhydra Jolt Tags

---

can also be used to separate Java method calls from the HTML component, for support or maintenance reasons.

---

### *The <JOLT JAVACATCH> Tag*

The general syntax of the <JOLT JAVACATCH> tag is as follows, and multiple <JOLT JAVACATCH> tags may not be nested:

```
<JOLT JAVACATCH="ExceptionName">
    ...Java Code...
</JOLT>
```

To catch exceptions created at the Presentation Object level, a single <JOLT JAVACATCH> clause can be created. It does not matter where in the file the clause appears, but the order is maintained. However, this clause is rarely necessary, as exceptions are usually caught within the Java code within the Presentation Object, or by an external exception mechanism.

The content of the <JOLT JAVACATCH> clause is the body of a regular Java catch handler. For example:

```
<JOLT JAVACATCH="MyException">
    // The catch handler code for MyException...
</JOLT>

<JOLT JAVACATCH="Exception">
    // Catch all exceptions...
</JOLT>
```

The <JOLT JAVACATCH> tag offers the facility to catch exceptions thrown anywhere within a page. A more general mechanism is offered by Enhydra, enabling exception handlers to be declared to handle exceptions from a group of Presentation Objects.

---

### *The <JOLT JAVAFINALLY> Tag*

The general syntax of the <JOLT JAVAFINALLY> tag is as follows, and multiple <JOLT JAVAFINALLY> tags may not be nested:

```
<JOLT JAVAFINALLY>
    ...Java Code...
</JOLT>
```

Similarly to <JOLT JAVACATCH> clauses, a single <JOLT JAVAFINALLY> clause can be declared anywhere in a file. It does not matter where in the file the clause appears, but there can be only one. Also similar to <JOLT JAVACATCH> clauses, this clause is rarely necessary. The contents of the <JOLT JAVAFINALLY> clause is the body of a regular Java finally handler.

```
<JOLT JAVAFINALLY>
    // Any Java code here will always be executed...
</JOLT>
```

## *Glossary of Terms*

---

The following terms are frequently used in the Enhydra Jolt Syntax Reference Guide:

**Attribute** One or more strings that appears after an element name within the start-tag. For example:

```
<JOLT JAVACATCH>
```

where JAVACATCH is an attribute.

**Content** The HTML text that appears between a start-tag and end-tag. For example:

```
<B>
```

```
Text in bold
```

```
</B>
```

where "Text in bold" is the content.

**Element** A string that defines the structure of an HTML document. Elements are enclosed in angle brackets, referred to as "tags." For example, B is an HTML element. JOLT is also an HTML element.

**End-tag** An HTML tag that terminates an HTML statement. For example: </JOLT>

---

## Glossary of Terms

---

**Start-tag** An HTML tag that begins an HTML statement. For example:  
`<JOLT>`.

**Value** Values may be optionally assigned to attributes in the form of an attribute-value pair. For example:

```
<JOLT JAVACALL="class.myMethod"  
ARG.a="foo" ARG.b="bar">  
  
</JOLT>
```

contains three attributes (`CALL`, `ARG.a`, `ARG.b`) and three values, respectively (`class.myMethod`, `foo`, `bar`).

---

## INDEX

---

### Symbols

11, 12, 14, 16, 18, 19, 22  
&#040 6  
&#041 6  
' ' 6  
(@ 5  
.jhtml 1  
.jinc file extension 19  
.po 3  
) 5

### A

addColor 14  
addNewColor 10  
ARG. prefix 13

### C

CALL 12, 13  
call(methodName) method 11  
class file 2  
client-side language 6  
conditions 14

### D

-d destdir (joltc compiler option) 2

### E

entry method 3  
exception 6  
exceptions 9

### F

field names, dumping 7

### G

GET 5  
getRequest 13

### H

HTMLDEF 12

### I

IFCALL 10  
IFDEF 10  
IFEQ 10  
IFNCALL 10  
IFNDEF 10  
IFNEQ 10

INCLUDE 19

### J

-j javadir (joltc compiler option) 2  
JAVACATCH 21  
JAVADEF attribute 10  
JAVAFINALLY 22  
JAVAIMPORT 19  
JavaScript 6  
Jolt Fields 5  
Jolt, introduction to 1  
joltc compiler 2, 3  
JoltDebug 13  
JoltPage class 11

### K

-k (joltc compiler option) 2  
KeywordValueTable 11

### L

Lutris Business Server 3

### M

makefiles 2  
method invocation 14  
minLength 6

### P

page.tagText 14  
POST 5  
Presentation Object 1, 3  
Presentation Objects 3

### R

referencing files 9  
run() method 3

### S

source file 2

### T

tests for conditions 9