

Roxen WebServer 2.2

1

Administrator Manual

2

Web Site Creator Manual

3

Tutorials

4

Programmer

5

Pike Tutorial

The inside of Internet

Table of Contents

Introduction	5
A request's path through Roxen	7
A simple module	9
More module examples	11
Reference for Roxen Java classes	15
AbstractLocationModule	15
ExperimentalModule	15
FileExtensionModule	15
Frame	15
HTTP	15
LastResortModule	16
LocationModule	16
Module	17
ParserModule	18
ProviderModule	18
RXML	18
RXML.Backtrace	18
RoxenClassLoader	19
RoxenConfiguration	19
RoxenFileResponse	19
RoxenLib	19
RoxenRXMLResponse	20
RoxenRequest	20
RoxenResponse	21
RoxenStringResponse	21
SecurityModule	21
SimpleTagCaller	21
UniqueModule	21

Table of Contents

Introduction

Prerequisites

The user is assumed to be familiar with the Java programming language, HTML, and recognize basic Roxen WebServer features, such as the administration interface, RXML, etc. The user also needs to be familiar with the operating system that Roxen WebServer is running on, to be able to install files in the Roxen WebServer filesystem.

Terminology

Roxen module

A component of a virtual server that fills some specific purpose, such as RXML parsing, providing a filesystem or similar. A Roxen module is a piece of code interfacing to the roxen module API, extending the server's features.

virtual filesystem

The virtual filesystem is the entire path namespace of your server, as seen in the URL's path segment. In a basic Roxen setup with just a common filesystem module mounted at / (the filesystem root), all files in the virtual filesystem map to this one filesystem, as is typically the case with other webservers where you have a single document root.

Roxen, however, has a slightly different approach to filesystems, that shares many characteristics with Unix filesystems. Several filesystems may be mounted on top of each other at different, or indeed the same, mountpoints in the virtual namespace of the server. If one filesystem is mounted at /, another at /I/live/ and a third at /I/live/here/, a request for the file /I/live/here/happily.html may deliver a file residing in either filesystem. Which one is decided first by the longest matching mountpoint, secondly the filesystem modules' priorities (as set by the administrator) and finally by which filesystem actually has a file to deliver.

Similarly, even entire other virtual servers may be mounted somewhere within your server's namespace, effectively blacking it out to your server. This also means that such a server's virtual filesystem does not map one-to-one to the path namespace as seen in the URL from the browser's point of view. What to the browser looks like a request of the file /my/secret/garden.jpg could quite possibly be the file /secret/garden.jpg from a virtual server mounted at /my/.

Due to the dynamic nature of Roxen, a file in the virtual filesystem need not exist as an actual file on disk (as it would in a real filesystem), but might be the result of some on-the-fly file-generation activity in a Roxen module of some sort. Indeed, even the error page received when roxen did not find a file, exists in the virtual filesystem (wherever anything else does not).

To find out what actually happens when requesting a given URL from your server; where the file has its origin, what modules touch it and so on, the "Resolve path..." option in the *Maintenance* menu under the Tasks tab in the administration interface does wonders.

real filesystem

The real filesystem, unlike the discussed above, refers to the filesystems provided by the operating system, where actual files reside, typically on a hard disk, network filesystem or similar. As the name hints, files on disk might be considered just the slightest bit more real than the figments of imagination that may be harvested from the virtual filesystem.

mountpoint

The term refers to where location modules are found in the virtual filesystem. A virtual file's path is considered to be below a certain mountpoint when the path begins with the the entire length of the mountpoint; for instance, /demo/4 is below both mountpoints / and /demo/ but not below /demo/module/ or /hello/world/. A mount point need not necessarily end in a slash, but it is a rather common convention.

virtual server

Back in the old days, a web server was one website serving files through one port at one host. Things change, and the world grows more complex. The widespread trend of abstracting a concept and tagging it "virtual" applies to virtual servers, as much as it does to the virtual filesystem; a virtual server in roxen is a collection of roxen modules working together under a common roof.

The criterion of being accessible on a single host:port combination does not apply in Roxen, nor the fact that the server uses the entire namespace of that domain. By using several server URL:s for a virtual server, it can respond to requests for several domains, on several ports possibly using several different protocols (eg HTTP, HTTPS and FTP, all to the same virtual filesystem). Or, just as possible, if given the sole server URL `http://my.domain/just/here/`, it will not see requests for any other part of the namespace of some other server handling `http://my.domain/` than those below `/just/here/`. Equally possible, although not the most common of configurations, is not to mount any port at all.

Roxen WebServer start-up process

Roxen WebServer is a collection of scripts and modules. The start script sets up some essential environment variables to keep dynamic libraries and various other external stuff happy. After a rapid progression of start script argument parsing and setting up paths, it rotates the logs and launches the first Pike script: `server/base_server/roxen-loader.pike`.

Next in turn, the Roxen loader bootstraps the server, making sure some dependencies are met, setting up some constants, and installs Roxen WebServer's own master program, `server/etc/roxen_master.pike`. This program is responsible for, among other things, the dumping and reloading of programs.

Then, the very core of Roxen WebServer, `server/base_server/roxen.pike` gets loaded. The configuration of

all virtual servers are loaded next, their respective ports get registered and unless the flag `--no-delayed-load` was passed to the start script, not much more happens next until the first request arrives to a server. This means, of course, this code in a module of yours will not run until a server using the module is needed.

At this stage, the server process is up and will listen to incoming requests on all registered ports. When a request to an uninitialized server is received, Roxen will initialize that server, loading and strapping all of its modules. The request is then sent through the common processing sequence. Your applications will not be affected by the delayed loading. To the application, whether it be a module, a script, an rxml page or a servlet, the delayed loading is transparent and does not interfere with the programming environment.

For the rest of the lifespan of the Roxen process, it will go about its business listening and responding to requests, until terminated with a signal to the process, the start script or by the restart/shutdown administrator action. When the Roxen process receives a SIGHUP signal, it reloads its configuration files. A SIGINT/SIGTERM signal takes down the process and makes the start script spawn off a new one to replace the old one (sending a SIGINT/SIGTERM to the start script shuts down the server without respawning another one).

A request's path through Roxen

This page follows a request's path from browser to server, through the various layers of Roxen, the module types and to the end of the chain where the resulting document is sent back to the browser. After reading this, you should have a good grasp on what happens when and how the various module types fit in the overall picture.

Note! Module types that can be written in Java will be noted as such.

For this example walk-through, we will follow a request from browser through the server and back; the example applies for all supported protocols. We will be abstracting all encountered modules by type instead of by name, since we're getting at the principle rather than the specifics of a specific server configuration.

Calling Sequence

This is a description of the order of calling modules in Roxen. Generally speaking, an incoming request passes through a number of *type levels*, which will be described in turn. A failure at a type level means that none of the modules of that type could treat the request. The case where there are no modules of a certain type is a trivial case of failure.

A failure usually means that the request is passed on to the next level. What happens when a module succeeds in treating the request depends on the level and module. The specific Java module interfaces are described in the Reference for Roxen Java Classes chapter in this manual.

1. An incoming request enters Roxen through the protocol module, which handles the lower level communication with the client.
2. If the protocol module got some form of authentication information from the client, the authentication module is invoked. Regardless of the success or failure, the request moves on to the next layer. The authentication status (fail or a valid user identity) is stored in the request information object.
3. The first try modules get the first shot at returning a response of some sort to the client. From here on, success or failure means breaking out of or staying with the flow of the calling sequence; handled requests are sent back to the client, unhandled are subject to enter the other module types further down the chain.
4. The request now enters a location module; which one depends on the path accessed. In this respect, the location modules work almost like your average file system; a given path refers to a certain file entry on some storage medium somewhere. Or, possibly, a directory entry or a non-existent file. In either of the latter cases, the request

moves on.

This module has a Java interface.

5. The request was found a directory at some earlier level, and it is now up to the directory listing module to generate some form of directory listing or representation of the directory at hand.
6. If some previous level sent handled a request by sending forth a file down the chain, it is processed by an appropriate file extension module (if one handling the proper extension was available).
7. The content-type module tags the resulting page with a suitable content-type for the file being sent back to the browser. Modules may of course override this, should they know what they want.
8. All requests then pass through the filter module stage. Filter modules may process and alter the request at leisure, watermarking, filtering out information or doing other forms of post processing.
9. If no module has yet handled the request, the last modules get a shot at catching and processing the request before a file not found error is sent back to the client. *This module has a Java interface.*
10. The protocol module which originally set this chain going is returned the result from previous stages and starts sending the result to the client in response to the request.
11. Finally, as the result is being transferred back to the client, the logger modules get their peek at the request. When the logger modules are done and the whole response is sent to the browser, the request information object dies and the request is over.

A simple module

How to Create Tag Modules With Java

In this first example we create and install a very simple module. The module gives us an RXML tag that just converts its contents to upper case.

Creating the module

First we create the source file ReverseTag.java

- It imports `com.roxen.roxen.*` and `java.util.Map`
- It extends `Module` - so it must implement the methods `info()` and `queryName()`
- It implements `ParserModule` - so it must implement the method `querySimpleTagCallers()`
- It implements `SimpleTagCaller` - so it must implement the methods `queryTagName()`, `queryTagFlags()` and `tagCalled()`

Then we compile the source file to get the file `ReverseTag.class`

Source code for UppercaseTag.java

```
import com.roxen.roxen.*;
import java.util.Map;

public class UppercaseTag extends Module
    implements ParserModule, SimpleTagCaller {

    public String info() {
        return "This tag converts its contents to upper case.";
    }

    public String queryName() {
        return "Uppercase Tag"; //
        The name of the module
    }

    public SimpleTagCaller[] querySimpleTagCallers(
    ) {
        return new SimpleTagCaller[] { this };
    }

    public String queryTagName() {
        return "uppercase"; //
        The name of the tag
    }

    public int queryTagFlags() {
        return FLAG_NONE;
    }

    public String tagCalled(String tag,
                            Map args,
                            String contents,
                            RoxenRequest id,
                            Frame frame) {

        return contents.toUpperCase(); //
        The actual action
    }
}
```

Installing the new module

The following steps will add the module to a certain site:

- Copy the class file to somewhere in the module search path, preferably the folder `[RoxenHome]/local/modules`. *Note: Do not use the server folder, since it is overwritten at site upgrades.*
- In the admin interface, go to the proper site and select 'Add Module'
- Select 'Reload module list'
- Select 'Local Modules' to expand and show all modules in this group
- At the newly added module (Uppercase Tag) select 'Add module'
- The module now appears in the module list for that site. If changes are made to the module, replace the old class file with the new and select 'Reload'.

A Closer Look At the Methods

`public String queryName()`

Returns the module name. This is shown both in the list of available modules when adding a module and in the module list for the site after adding the module.

`public String info()`

Returns a description of the module, shown in the list of available modules.

`public SimpleTagCaller[] querySimpleTagCallers()`

Returns an array of objects that implement one RXML-tag each. In our simple example, the array consists of only one object, which happens to be the class itself since it also implements the interface `SimpleTagCaller`. If one module would contain more RXML tag implementations, they would have to be divided into several classes that are packed into a jar-file (more about how to install such a module later).

`public String queryTagName()`

Returns the name of the actual tag. For example, 'uppercase' means that the tag will be written as `<uppercase>content</uppercase>`. Tag names are case sensitive.

`public int queryTagFlags()`

The interface `SimpleTagCaller` contains a number of flags in the form of int constants. At least one of these shall be returned. Several might be added, and the sum, which represents a bit pattern showing which flags have been set, is returned. There is, for example, one flag showing that the element has no contents, and another showing that the element is a processing instruction. If no other flag is set, then `FLAG_NONE` shall be returned.

`public String tagCalled(String tag, Map args, String contents, RoxenRequest id, Frame frame)`

This method evaluates the element (the tag with its parameters and contents) and returns the result. The first parameter is the tag name, for example "uppercase". The second parameter is a map with all pairs of attributes/values from the tag. The third parameter is

the content of the element. The last two parameters are
Roxen specific environment objects that may provide,
for example, the IP number of the client.

More module examples

A simple tag without contents

In this example we will create an RXML tag that outputs the company name. By updating the company name in the module, and then reloading the module, the change will occur everywhere that the tag has been used in the site. So this could be one way of implementing a string constant. (Another way would be to define it in an XSLT template, of course.) The module is installed exactly the same way as the Uppercase Tag module from the first example.

Notice the difference in the method `queryTagFlags()`, where we tell the parser that there should be no contents.

Source code for `CompanyName.java`

```
import com.roxen.roxen.*;
import java.util.Map;

public class CompanyNameTag extends Module
    implements ParserModule, SimpleTagCaller {

    public String info() {
        return "RXML empty tag that returns the company name.";
    }

    public String queryName() {
        return "Company Name";
    }

    public SimpleTagCaller[] querySimpleTagCallers(
    ) {
        return new SimpleTagCaller[] { this };
    }

    public String queryTagName() {
        return "companyname";
    }

    public int queryTagFlags() {
        return FLAG_EMPTY_ELEMENT;
    }

    public String tagCalled(String tag,
        Map args,
        String contents,
        RoxenRequest req,
        Frame frame) {

        return "Roxen Internet Software AB";
    }
}
```

Adding and using an admin variable

To improve the `companyname` tag, we would like to be able to change the string. One way of doing this is by introducing a variable in the administrative interface. This is accomplished by adding a constructor. There you put tasks to be performed when the class is loaded, for example the `defvar()` method. More information is in the code comments.

There are a number of constants in the class `Module` to set the type of the variable. All of these are not yet mapped

to Java classes, but they are there for future compatibility. For the moment, we only recommend the use of `TYPE_STRING` and `TYPE_FLAG`.

Source code for `VariableCompanyNameTag.java`

```
import com.roxen.roxen.*;
import java.util.Map;

public class VariableCompanyNameTag extends Module
    implements ParserModule, SimpleTagCaller {

    public VariableCompanyNameTag() {
        /* Parameters to the defvar function:
        String var, Object value, String name, int
        type, String doc
        The parameter name has the format "tabname:
        label"
        If there is no colon, the default tab 'settings'
        is used */
        defvar("companyName", "Roxen IS AB",
            "Company Name ", TYPE_STRING,
            "Company name -
            Default value: \"Roxen IS AB\".");
    }

    public String info() {
        return "RXML tag that returns a variable company name.";
    }

    public String queryName() {
        return "Variable Company Name";
    }

    public SimpleTagCaller[] querySimpleTagCallers(
    ) {
        return new SimpleTagCaller[] { this };
    }

    public String queryTagName() {
        return "var-companyname";
    }

    public int queryTagFlags() {
        return FLAG_EMPTY_ELEMENT;
    }

    public String tagCalled(String tag, Map args,
        String contents,
        RoxenRequest req, Frame
        frame) {

        String company = (String)query("companyName
        ");
        if (company != null) return company;
        /* else */ return "";
    }
}
```

Addressing variables from other modules

At times you might need to get the value from a variable belonging to another module. You reach other modules by using a provider module. In this example we create a module whose sole purpose is to provide common parameters,

where the class is called ParameterProvider. We then show the code lines needed to access the parameter from another module

Source code for ParameterProvider.java

```
import com.roxen.roxen.*;

public class ParameterProvider
    extends Module implements ProviderModule {

    /* *****
     *      Constructor
     * ***** */

    public ParameterProvider () {
        defvar("productString", "Roxen CMS", "productString",
            TYPE_STRING, "Product name");
    }

    /* *****
     *      Methods from Module
     * ***** */

    /** Returns the name of this module */
    public String queryName() {
        return "Parameter Provider";
    }

    /

    /** Returns a short description of this module */
    public String info() {
        return "Contains parameters reachable from
other modules.";
    }

    /* *****
     *      Methods from ProviderModule
     * ***** */

    /** Returns the identifier of this provider */
    public String queryProvides() {
        return "commonParameters";
    }
}
```

Code to reach an external variable in ParameterProvider from other modules

```
Module params = myConfiguration().getProvider("commonParameters");
String product = params.query("productString");
```

A useful tag showing incoming parameters

To understand the incoming parameters to the tagCalled-function, and for debugging purposes, it can be useful with a module that outputs these parameters. In addition to the previous examples, this module contains one new method:

- The method start(), with tasks to be performed when the module is started.

Source code for JavaDebugLightTag.java

```
import com.roxen.roxen.*;
import java.util.*;

public class JavaDebugLightTag
    extends Module implements ParserModule, SimpleTagCaller {

    int callCount;
```

```
StringBuffer resp;

/* *****
 *      Constructor
 * ***** */

/

** The constructor is initially called when the module is loaded.
    For example, register variables for the admin interface */
    public JavaDebugLightTag() {

        /* parameters to the defvar function:
        String var, Object value, String name, int type, String doc
        The parameter name has the format "tablename:label"
        If no colon is present, default tab 'settings' is used
        There are 19 valid types, but here we show use only two:
        TYPE_STRING and TYPE_FLAG */

        /

        * The first variable will appear under default tab 'Settings' */
        defvar("myString", "Roxen", "myString", TYPE_STRING,
            "String admin variable. " +
            "Default value: \"Roxen\".");

        /

        * The second variable will appear under a tab called 'Flags'.
        A flag has the value "1" when set and null when unset. */
        defvar("myFlag", null, "Flags: myFlag", TYPE_FLAG,
            "An example admin variable in the form of a flag.");
    }

    /* *****
     *      Methods from Module
     * ***** */

    /** Returns the name of this module */
    public String queryName() {
        return "Java Debug Light Tag";
    }

    /

    /** Returns a short description of this module */
    public String info() {
        return "A demo Java module for outputting some parameters.";
    }

    /

    /** Initial things to do when the module is started.
        For example, initiate variables or write to the log */
    protected void start() {
        callCount = 0;
    }

    /* *****
     *      Methods from ParserModule
     * ***** */

    /

    /** Returns one instance of each RXML tag class (SimpleTagCaller)
        handled by this parser module. */
    public SimpleTagCaller[] querySimpleTagCallers() {
```

```

        return new SimpleTagCaller[] { this };
    }

    /* *****
     * Methods from SimpleTagCaller *
     * ***** */

    /
    ** Returns the name of the tag handled by this tag
    caller. */
    public String queryTagName() {
        return "javadebuglight";
    }

    /
    ** Returns one or more mode flags for this tag call
    er. */
    public int queryTagFlags() {
        /
        * Valid flags: FLAG_NONE, FLAG_EMPTY_ELEMENT,
          FLAG_NO_PREFIX, FLAG_PROC_INSTR, FLAG_DO
        NT_PREPARSE,
          FLAG_POSTPARSE, FLAG_STREAM_RESULT, FLAG
        _STREAM_CONTENT,
          FLAG_DEBUG */
        return FLAG_NONE;
    }

    /
    ** Evaluates a result for the call to this tag call
    er. */
    public String tagCalled(String tag, Map args, S
    tring contents,
                           RoxenRequest id, Frame
    frame) {
        /* tag -
         the name of the tag (javadebuglight)
         args - all attributes/value-
         pairs from the start tag
         contents -
         all contents between the start and the end tag
         id - the Roxen request object
         frame - the RXML parse frame */

        callCount++;
        resp = new StringBuffer();

        /* Class variables */
        resp.append("<b>CLASS VARIABLES</b><br/>");
        resp.append("Call Count=" + callCount + "<b
r/>");

        /* Element variables */
        resp.append("<br/><b>ELEMENT VARIABLES</
b><br/>");
        resp.append("tag=" + tag + "<br/>");
        resp.append("args=" + args + "<br/>");
        resp.append("contents=" + contents + "<br/
>");

        /* Admin variables */
        resp.append("<br/><b>ADMIN VARIABLES</
b><br/>");
        resp.append("myString: " + query("myString"
) + "<br/>");
        if (query("myFlag") != null) {
            resp.append("Admin variable 'myFlag' is
set " +
                        "(Value=" + query("myFlag")
+ ")<br/>");
        } else {
            resp.append("Admin variable 'myFlag' is
unset<br/>");
        }

        /* URL parameter test */
        resp.append("<br/><b>URL PARAMETER TEST</
b><br/>");
        /* Check the URL for parameter ?cmd=save
         if (id.variables().containsKey("cmd")) {
             if (id.variables().get("cmd").toString(
) .equals("save")){
                 resp.append("URL contains 'cmd=save
' <br/>");
             } else {
                 resp.append("URL parameter 'cmd' is
"
                            + "something else than 's
ave'<br/>");
             }
         } else {
             resp.append("URL does not contain param
eter 'cmd'<br/>");
         }

        /* some module class methods */
        resp.append("<br/><b>MODULE CLASS METHODS</
b><br/>");
        resp.append("Internal location: "
                    + queryInternalLocation() + "<br
/>");
        resp.append("Status: " + status() + "<br/
>");

        /* some client request variables */
        resp.append("<br/><b>CLIENT REQUEST VARIABLES</b><br/>");
        resp.append("Client IP: " + id.remoteaddr +
"<br/>");
        resp.append("Raw URL: " + id.rawURL + "<br/
>");
        resp.append("Request variables: " + id.vari
ables()
                    + "<br/>");
        resp.append("Remote address: " + id.remotea
ddr + "<br/>");
        resp.append("Request time: " + new Date(id.
time) + "<br/>");

        resp.append("<br/>Raw request:<br/
>" + id.raw
                    + "<br/><br/>");

        /* some client request methods */
        resp.append("<br/><b>CLIENT REQUEST METHODS</b><br/>");
        resp.append("Cookies:<br/
>" + id.cookies() + "<br/><br/>");
        resp.append("Supported:<br/
>" + id.supports() + "<br/>");

        /* Evaluation is finished -
         return the result */
        return RoxenLib.parseRXML(resp.toString(),
id);
    }
}

```

A module containing more than one class file

So far we have implemented each module as a class file of its own. This is possible, but the preferred way is to install a module as a jar-file, perhaps implementing several tags in one and the same module. As an example we will have one module containing the two tags <uppercase></uppercase> and <companyname>.

What we need:

- Two classes that implement SimpleTagCaller.
- A separate class that implements ParserModule.
- A manifest file.
- To create the jar-file.

We now also have a package structure for our classes. The jar file will be created in the top directory 'myproject' in this example

Directory and file structure

```
- myproject (dir)
- classes (dir)
  Manifest
  - com (dir)
    - roxen (dir)
      - examples (dir)
        JavaTags.class
        CompanyNameTag.class
        UpperCaseTag.class
```

Source of Manifest file

Name: Roxen Java RXML tag module
Main-Class: com.roxen.examples.JavaTags

Note! The Manifest file must end with a line feed, which means you must press Enter at the end of the line defining 'Main Class' before saving the file.

In the directory where you have saved the Manifest file, create the JavaTags.jar file by typing the following on the command line.

```
jar -cvfm ../JavaTags.jar Manifest com
```

Source of CompanyNameTag.java

```
package com.roxen.examples;

import com.roxen.roxen.*;
import java.util.Map;

public class CompanyNameTag implements SimpleTagCaller {

    public String queryTagName() {
        return "companyname";
    }

    public int queryTagFlags() {
        return FLAG_EMPTY_ELEMENT;
    }

    public String tagCalled(String tag, Map args, String contents,
                            RoxenRequest req, Frame
                            frame) {

        return "Roxen Internet Software AB";
    }
}
```

Source of UpperCaseTag.java
package com.roxen.examples;

```
import com.roxen.roxen.*;
import java.util.Map;

public class UppercaseTag implements SimpleTagCaller {

    public String queryTagName() {
```

```
        return "uppercase";
    }

    public int queryTagFlags() {
        return FLAG_NONE;
    }

    public String tagCalled(String tag, Map args, String contents,
                            RoxenRequest req, Frame
                            frame) {

        return contents.toUpperCase();
    }
}

Source of JavaTags.java
package com.roxen.examples;

import com.roxen.roxen.*;
import java.util.Map;

public class JavaTags extends Module implements ParserModule {

    public String info() {
        return "Java module for new RXML tags.";
    }

    public String queryName() {
        return "Java RXML Tags";
    }

    public SimpleTagCaller[] querySimpleTagCallers(
    ) {
        SimpleTagCaller[] tagArray = new SimpleTagCaller[2];
        tagArray[0] = new CompanyNameTag();
        tagArray[1] = new UppercaseTag();
        return tagArray;
    }
}
```

Reference for Roxen Java classes

AbstractLocationModule

```

java.lang.Object
|
+--com.roxen.roxen.Module
|
+--com.roxen.roxen.AbstractLocationModule

```

An abstract adaptor class that provides default implementations for most methods in the LocationModule interface.

A module inheriting this class must either create a module variable location using defvar, or provide a different implementation of the queryLocation method.

```

public AbstractLocationModule()

public java.lang.String queryLocation()

public java.lang.String[] findDir(java.lang.String f,
                                   RoxenRequest id)

public java.lang.String realFile(java.lang.String f,
                                   RoxenRequest id)

public int[] statFile(java.lang.String f,
                      RoxenRequest id)

```

ExperimentalModule

The interface for modules that should be marked as experimental in the module listing. It contains no methods, implementing it just flags the module as experimental.

FileExtensionModule

The interface for modules which handle a specific file extension.

```

public java.lang.String[] queryFileExtensions()

public RoxenResponse handleFileExtension(java.io.File file,
                                           java.lang.String ext,
                                           RoxenRequest id)

```

Frame

```

java.lang.Object
|
+--com.roxen.roxen.Frame

```

An object representing an RXML parse frame

HTTP

```

java.lang.Object
|
+--com.roxen.roxen.HTTP

```

A support class providing HTTP related functionality. Rather than using this class directly, all these functions can be accessed through the RoxenLib class.

```
public static java.lang.String httpEncodeString(java
.lang.String f)
```

The following characters are replaced with % escapes: SP, TAB, LF, CR, %, ', ", NUL.

```
public static RoxenResponse httpLowAnswer(int erro
r,
                                     java.lang
.String data)
```

```
public static RoxenResponse httpLowAnswer(int erro
r)
```

```
public static RoxenResponse httpStringAnswer(java.la
ng.String text,
                                     java.l
ang.String type)
```

```
public static RoxenResponse httpStringAnswer(java.la
ng.String text)
```

```
public static RoxenResponse httpRXMLAnswer(java.
lang.String text,
                                     java.lan
g.String type)
```

```
public static RoxenResponse httpRXMLAnswer(java.
lang.String text)
```

```
public static RoxenResponse httpFileAnswer(java.io.R
eader text,
                                     java.lan
g.String type,
                                     long len
)
```

```
public static RoxenResponse httpFileAnswer(java.io.R
eader text,
                                     java.lan
g.String type)
```

```
public static RoxenResponse httpFileAnswer(java.io.R
eader text)
```

```
public static RoxenResponse httpFileAnswer(java.io.I
nputStream text,
                                     java.lan
g.String type,
                                     long len
)
```

```
public static RoxenResponse httpFileAnswer(java.io.I
nputStream text,
                                     java.lan
g.String type)
```

```
public static RoxenResponse httpFileAnswer(java.io.I
nputStream text)
```

```
public static RoxenResponse httpFileAnswer(java.io.F
ile text,
                                     java.lan
g.String type,
                                     long len
```

```
)
                                     throws java.io.
FileNotFoundException
```

```
public static RoxenResponse httpFileAnswer(java.io.F
ile text,
                                     java.lan
g.String type)
                                     throws java.io.
FileNotFoundException
```

```
public static RoxenResponse httpFileAnswer(java.io.F
ile text)
                                     throws java.io.
FileNotFoundException
```

```
public static RoxenResponse httpRedirect(java.net.UR
L url)
```

```
public static RoxenResponse httpAuthRequired(java.l
ang.String realm,
                                     java.l
ang.String message)
```

```
public static RoxenResponse httpAuthRequired(java.l
ang.String realm)
```

```
public static RoxenResponse httpProxyAuthRequired(j
ava.lang.String realm,
                                     j
ava.lang.String message)
```

```
public static RoxenResponse httpProxyAuthRequired(j
ava.lang.String realm)
```

LastResortModule

```
public RoxenResponse last_resort(RoxenRequest id)
```

LocationModule

The interface for modules which have a specific URL path in the virtual file system.

```
public java.lang.String queryLocation()
```

```
public RoxenResponse findFile(java.lang.String f,
                               RoxenRequest id)
```

```

public java.lang.String[] findDir(java.lang.String f
,
                                RoxenRequest id)

public java.lang.String realFile(java.lang.String f,
                                RoxenRequest id)

public int[] statFile(java.lang.String f,
                     RoxenRequest id)

```

Module

```

java.lang.Object
|
+--com.roxen.roxen.Module

```

The base class for Roxen modules. All modules must inherit this class, directly or indirectly.

Each module should also implement one or more of the specific module type interfaces.

```

public static final int TYPE_STRING

public static final int TYPE_FILE

public static final int TYPE_INT

public static final int TYPE_DIR

public static final int TYPE_STRING_LIST

public static final int TYPE_MULTIPLE_STRING

public static final int TYPE_INT_LIST

public static final int TYPE_MULTIPLE_INT

public static final int TYPE_FLAG

public static final int TYPE_TOGGLE

public static final int TYPE_DIR_LIST

public static final int TYPE_FILE_LIST

public static final int TYPE_LOCATION

public static final int TYPE_TEXT_FIELD

public static final int TYPE_TEXT

```

```

public static final int TYPE_PASSWORD

public static final int TYPE_FLOAT

public static final int TYPE_MODULE

public static final int TYPE_FONT

public static final int VAR_EXPERT

public static final int VAR_MORE

public static final int VAR_DEVELOPER

public static final int VAR_INITIAL

public Module()

public abstract java.lang.String queryName()

public abstract java.lang.String info()

public RoxenConfiguration myConfiguration()

protected java.lang.String queryInternalLocation()

protected RoxenResponse findInternal(java.lang.String
f,
                                     RoxenRequest i
d)

public java.lang.String status()

protected void start()

protected void stop()

protected void defvar(java.lang.String var,
                      java.lang.Object value,
                      java.lang.String name,
                      int type,
                      java.lang.String doc)

protected void defvar(java.lang.String var,
                      java.lang.Object value,
                      java.lang.String name,
                      int type)

public java.lang.Object query(java.lang.String name
)

public int queryInt(java.lang.String name)

public java.lang.String queryString(java.lang.String
name)

protected void set(java.lang.String name,
                   java.lang.Object value)

```

```
protected void set(java.lang.String name,
                  int value)
```

ParserModule

The interface for modules which define RXML tags.

```
public SimpleTagCaller[] querySimpleTagCallers()
```

ProviderModule

The interface for modules providing services to other modules.

```
public java.lang.String queryProvides()
```

RXML

```
java.lang.Object
|
+--com.roxen.roxen.RXML
```

RXML framework

```
public static java.lang.Object getVar(java.lang.String var,
                                     java.lang.String scopeName)
```

```
public static java.lang.Object getVar(java.lang.String var)
```

```
public static java.lang.Object userGetVar(java.lang.String var,
                                          java.lang.String scopeName)
```

```
public static java.lang.Object userGetVar(java.lang.String var)
```

```
public static java.lang.Object setVar(java.lang.String var,
```

```
    java.lang.Object val,
    java.lang.String scopeName)
```

```
public static java.lang.Object setVar(java.lang.String var,
                                       java.lang.Object val)
```

```
public static java.lang.Object userSetVar(java.lang.String var,
                                           java.lang.Object val,
                                           java.lang.String scopeName)
```

```
public static java.lang.Object userSetVar(java.lang.String var,
                                           java.lang.Object val)
```

```
public static void deleteVar(java.lang.String var,
                             java.lang.String scopeName)
```

```
public static void deleteVar(java.lang.String var)
```

```
public static void userDeleteVar(java.lang.String var,
                                 java.lang.String scopeName)
```

```
public static void userDeleteVar(java.lang.String var)
```

```
public static void runError(java.lang.String msg)
    throws RXML.Backtrace
```

```
public static void parseError(java.lang.String msg)
    throws RXML.Backtrace
```

```
public static void tagDebug(java.lang.String msg)
```

RXML.Backtrace

```
java.lang.Object
|
+--java.lang.Throwable
|   |
|   +--java.lang.Error
|       |
|       +--com.roxen.roxen.RXML.Backtrace
```

The object used to throw RXML errors.

```
public java.lang.String getType()
```

RoxenClassLoader

```
java.lang.Object
|
+--java.lang.ClassLoader
|
+--java.security.SecureClassLoader
|
+--java.net.URLClassLoader
|
+--com.roxen.roxen.RoxenClass-
Loader
```

```
public RoxenClassLoader(java.net.URL[] urls)
```

```
public void addJarFile(java.lang.String jarFileName)
    throws java.io.FileNotFoundException,
           java.io.IOException
```

```
public static java.lang.String getModuleClassName(
    java.lang.String jarFileName)
    throws java.io.FileNotFoundException,
           java.io.IOException
```

RoxenConfiguration

```
java.lang.Object
|
+--com.roxen.roxen.RoxenConfiguration
```

A class representing the configuration of a virtual server in the Roxen server.

```
public RoxenConfiguration()
```

```
public java.lang.String getRealPath(java.lang.String filename,
    RoxenRequest id)
)
```

```
public java.lang.String getFileContents(java.lang.String filename,
```

```
t id)
```

```
public java.lang.String getMimeType(java.lang.String filename)
```

```
public java.lang.Object query(java.lang.String name)
```

```
public java.lang.String queryInternalLocation(Module m)
```

```
public java.lang.String queryString(java.lang.String name)
```

```
public java.lang.String queryInternalLocation()
```

```
public Module[] getProviders(java.lang.String provides)
```

```
public Module getProvider(java.lang.String provides)
```

RoxenFileResponse

```
java.lang.Object
|
+--com.roxen.roxen.RoxenResponse
|
+--com.roxen.roxen.RoxenFileResponse
```

A class of responses using a file as their source. Use the methods in the HTTP class to create objects of this class.

RoxenLib

```
java.lang.Object
|
+--com.roxen.roxen.HTTP
|
+--com.roxen.roxen.RoxenLib
```

A support class containing useful methods for interpreting requests and synthesizing responses.

```
public static java.lang.String htmlEncodeString(java.lang.String str)
```

The following characters are replaced with HTML entities: &, <, >, ", ', NUL.

```
public static java.lang.String htmlDecodeString(
    java.lang.String str)
```

All HTML 4.0 entities are replaced with their literal equivalent.

```
public static java.lang.String doOutputTag(
    java.util.Map args,
    java.util.Map[] varArr,
    java.lang.String contents,
    RoxenRequest request id)
```

This method is used to create tags such as database query tags, where zero or more results in the form of variable bindings are applied to a fixed template, and the results of the substitutions are concatenated to form the total result.

```
public static java.lang.String parseRXML(
    java.lang.String what,
    RoxenRequest request id)
```

```
public static java.lang.String makeTagAttributes(
    java.util.Map in)
```

```
public static java.lang.String makeTag(
    java.lang.String s,
    java.util.Map in)
```

```
public static java.lang.String makeEmptyElemTag(
    java.lang.String s,
    java.util.Map in)
```

```
public static java.lang.String makeContainer(
    java.lang.String s,
    java.util.Map in,
    java.lang.String contents)
```

RoxenRXMLResponse

```
java.lang.Object
|
+--com.roxen.roxen.RoxenResponse
    |
    +--com.roxen.roxen.RoxenStringResponse
        |
        +--com.roxen.roxen.RoxenRXMLResponse
```

A class of responses using an RXML parsed string as their source. Use the methods in the HTTP class to create objects of this class.

RoxenRequest

```
java.lang.Object
|
+--com.roxen.roxen.RoxenRequest
```

A class representing requests from clients.

```
public final RoxenConfiguration conf
public final java.lang.String rawURL
public final java.lang.String prot
public final java.lang.String clientprot
public final java.lang.String method
public final java.lang.String realfile
public final java.lang.String virtfile
public final java.lang.String raw
public final java.lang.String query
public final java.lang.String notQuery
public final java.lang.String remoteaddr
public final long time
public final RoxenConfiguration configuration()
public java.util.Map variables()
public java.util.Map requestHeaders()
public java.util.Map cookies()
public java.util.Set supports()
public java.util.Set pragma()
public java.util.Set prestate()
```

```
public void cache(int sec)
```

```
public void noCache()
```

RoxenResponse

```
java.lang.Object
|
+--com.roxen.roxen.RoxenResponse
```

The base class for response objects. Use the methods in the HTTP class to create response objects.

```
public void addHTTPHeader(java.lang.String name,
                          java.lang.String value)
```

RoxenStringResponse

```
java.lang.Object
|
+--com.roxen.roxen.RoxenResponse
|
+--com.roxen.roxen.RoxenStringResponse
```

A class of responses using a string as their source. Use the methods in the HTTP class to create objects of this class.

SecurityModule

The interface for modules that should be marked as security modules in the module listing. It contains no methods, implementing it just flags the module as security related.

SimpleTagCaller

The interface for handling a single specific RXML tag

```
public static final int FLAG_NONE
```

```
public static final int FLAG_EMPTY_ELEMENT
```

```
public static final int FLAG_NO_PREFIX
```

```
public static final int FLAG_PROC_INSTR
```

```
public static final int FLAG_DONT_PREPARSE
```

```
public static final int FLAG_POSTPARSE
```

```
public static final int FLAG_STREAM_RESULT
```

```
public static final int FLAG_STREAM_CONTENT
```

Note: It might be obvious, but using streaming is significantly less effective than nonstreaming, so it should only be done when big delays are expected.

```
public static final int FLAG_STREAM
```

```
public static final int FLAG_DEBUG
```

```
public java.lang.String queryTagName()
```

```
public int queryTagFlags()
```

```
public java.lang.String tagCalled(java.lang.String tag,
                                  java.util.Map args,
                                  java.lang.String contents,
                                  RoxenRequest id,
                                  Frame frame)
```

UniqueModule

The interface for modules that may only have one copy in any given virtual server. It contains no methods, implementing it just prevents multiple copies of the module from being added to a virtual server.