

# TopLeaf Server

2010-10-07

**IMPORTANT NOTE:** Turn-Key Systems is committed to the quality and completeness of the TopLeaf documentation. If you notice an error or omission, or wish to have some point more fully covered, please contact us at [support@turnkey.com.au](mailto:support@turnkey.com.au). Let us know the location and nature of the problem and we will do our best to address the issue in the next release.

## 1. Introduction

TopLeaf Server provides client applications with a simple and flexible method for delivering XML and related data to TopLeaf for rendering into a number of published formats.

When the server is running, it listens for connections from clients on a TCP/IP socket. The client sends a **request** to the socket and receives a **response** from the server. The protocols used for the request and response are extensible, and allow for multiple data objects in a single transmission. [Chapter 4](#) contains a description of the protocols.

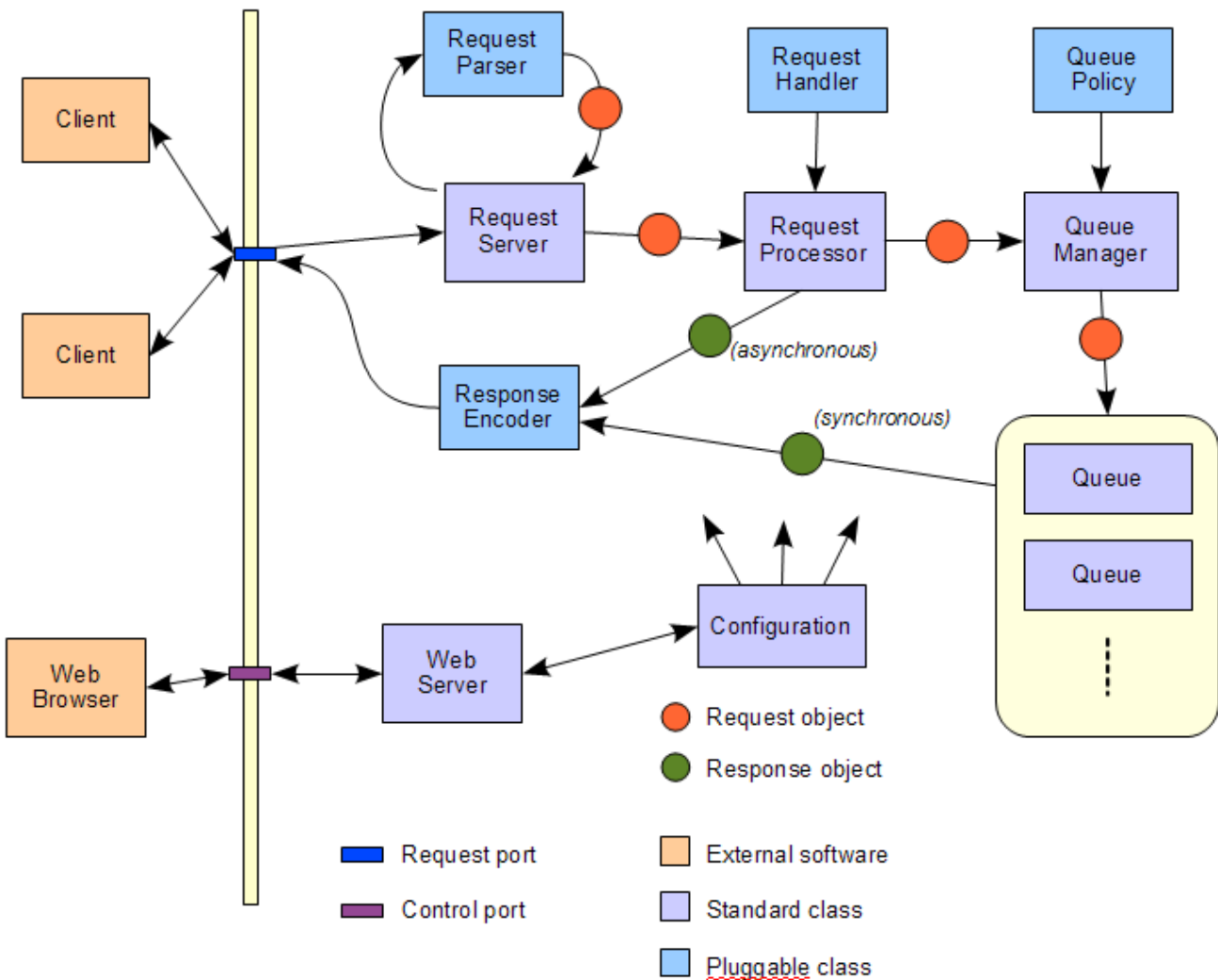
[Chapter 2](#) describes how to install and configure the server. [Chapter 3](#) contains information on administering the server.

For information on writing client software that communicates with the server, see [Chapter 6](#).

See [Chapter 7](#) on how the server can be extended and customised to meet your specific needs.

### 1.1 Server Architecture

The following diagram shows the structure of TopLeaf Server.



The following is a brief description of the operation of TopLeaf Server.

1. A client initiates a request by connecting to the **request port**. The data for the request is written as a stream of bytes and the client waits for a response.
2. The **request server** reads the data and passes it to a Request parser. The parser creates a corresponding **request object** which encapsulates the data.
3. The request object is passed to a **request processor** which decides how to process it. The request processor creates a **request handler** object and attaches it to the request; the request handler is responsible for actually executing the request.
4. If the request is **asynchronous**, the request processor will send a response to the client before processing begins. A **response encoder** is used to generate the stream of bytes sent back to the client.
5. The request object is passed to the **queue manager** to determine the **execution queue** on which it will be run. The **queue policy** decides on which execution queue(s) a particular request can be processed.
6. If the request is **synchronous**, the response will be sent when processing is complete. In this case the response may contain any or all of the generated data.

The operation of TopLeaf Server can be monitored and controlled by connecting a standard web browser to the **control port**. The interface to the server appears as a collection of web pages.

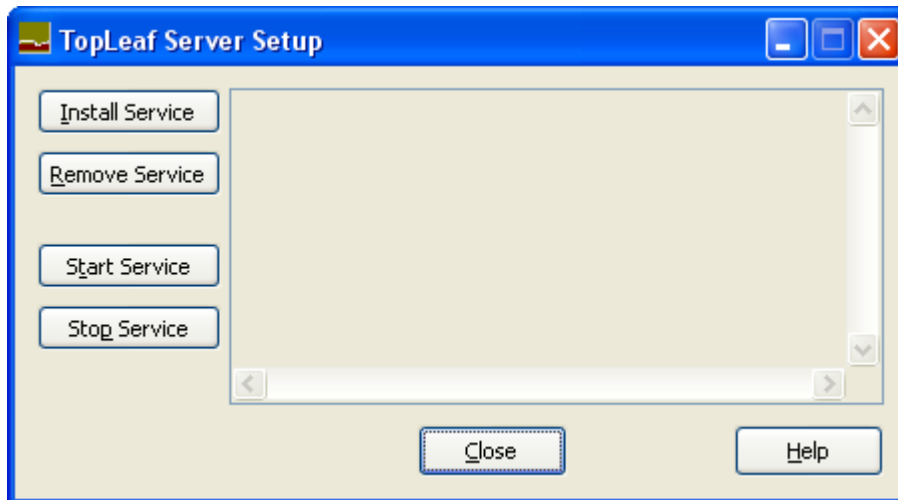
Configuration information is stored in a configuration object which is read from and written to an XML document.

The architecture of the server allows special behavior to be added at various points throughout the use of **pluggable classes**. See [Chapter 7](#) for more information.

## 2. Installation and Configuration

### 2.1 Windows Service Management

After installing the server software, run the server setup program located in the Start menu. This program can also be run as the final step of the installer.



**Warning:**

**Do not attempt to set up the server until TopLeaf has been installed and initialized.**

**TopLeaf must have been started at least once in order to set parameters required to configure the server.**

**Note:**

If the setup program cannot determine the java runtime selected for TopLeaf, it will open a dialog showing all of the currently-installed runtimes. Select one and press OK to continue.

Press the **Install Service** button to install TopLeaf Server as a Windows service. The messages displayed will indicate if this was successful.

Once the Windows service is installed you can use the standard Windows application to start and stop it. You can also use the **Start Service** and **Stop Service** buttons on this application.

Note that after installing the service you need to start it manually the first time. The service will start automatically when the system is restarted.

#### 2.1.1 Advanced Service Configuration

When the service is first installed, a file called **wrapper.conf** is created. This contains configuration parameters for the service. Consult the comments in the file for information on the meaning of each parameter.

This file contains the locations of all java classes that are used by the server (the classpath). If you need to access additional classes, add them here.

If your application requires more memory than the default, you can change the maximum memory parameter.

If you change any of the parameters in this file you must restart the service before they take effect.

### 2.1.2 Service Account

By default the service runs using the **LocalSystem** account. This account has access to most resources on the machine on which it is running, but in general does not have access to network resources. There are two ways to change the account under which the service runs:

1. In the **wrapper.conf** file, set the **wrapper.ntservice.account** and **wrapper.ntservice.password** parameters to the account name and password, respectively. The service must be removed and reinstalled before these parameters take effect.

This method has the advantage that the account is applied automatically when the service is reinstalled, but has the obvious drawback that the account password is stored in clear text.

2. After installing the service, start the Windows Services manager (in Windows, select Control Panel/Administrative Tools/Services). Right-click the TopLeaf Server entry and select Properties. Select the Log On tab and enter the account name and password.

This method is more secure with respect to the account password, but must be done each time the service is installed.

Note that the account name is of the form **Domain\Account**, and accounts on the local machine should be entered in the form **.\Account**.

## 2.2 Server Console

The server implements a console interface that can be accessed with a standard web browser. Use a URL that specifies the appropriate port on the computer running TopLeaf Server; for example:

**`http://server3.company.com:61578`**

You may nominate a user name and password which must be entered in order to access the console. These are specified by the `<login>` element in the configuration file (see [Section 2.3](#)).

#### **Warning:**

**The console login does not provide strong security, because the user name and password are transmitted in a form that can easily be decoded. If you need to make the server console available outside of a secure environment use an appropriate method to access it (for example, tunnelling over SSH).**

#### Note:

You can add your own customised pages to the server console as described in [Section 7.2](#).

### 2.2.1 Server Status Page

This page shows status information for the server.

The status appears as “Accepting requests” if the server is running normally, or “Not accepting requests” if it is running but not accepting connections from clients. The latter mode can be useful for performing maintenance by preventing any new requests from being processed. The button below the message is used to switch from one state to the other — its label shows “Pause” or “Resume” as appropriate.

The status can also appear as “Shutting down” if the server has been instructed to stop.

The request count fields show the following:

**Requests received** The total number of requests that have been received since the server started.

**Requests executing** The number of requests that are currently being processed. This is limited by the number of [queues](#) available.

**Requests pending** The number of requests that have been received and are waiting to start processing.

The server log field shows the location of the file that contains log messages from the server. Note that the log is “rotated” to a new file when it gets to a certain size. Versions of the log file are identified by adding a single numeric digit suffix.

The temporary directory field shows the path to the directory in which temporary files are placed while processing a request. Each request has its own subdirectory in this directory.

### 2.2.2 TopLeaf Defaults Page

This page contains defaults for TopLeaf processing. These can be overridden for a specific request by setting values in the [request descriptor](#).

The repository fields are used to set the path to the TopLeaf repository containing the publications. A number of **named repository locations** can be created. These allow requests to specify repositories by abstract names such as “development” or “production” instead of actual locations. This can assist the server administrator to manage repositories without having to change the requests sent by clients.

To change the location of a named repository, enter a new value in its location text field and press [Update](#). To remove a named repository, tick the Remove checkbox and press [Update](#). To create a new named location click the **Add named location** link and fill in the form that is displayed.

The default publication field is a path to a publication relative to the repository root. Use slash characters to separate path components (for example: “Manuals/UserGuide”). The default publication may be left empty if requests always nominate the publication to use.

The default partition field is also a path relative to the repository root, and uses the same format as the publication field. Note that a request does not need to specify a specific partition; if there is no partition the request processor will create a temporary partition in the publication. Temporary partitions are created by copying the partition called **Template** which must exist in the publication.

The **Action on composition warning** and **Action on composition error** fields allow you to specify how the request processor reacts to warnings and errors during TopLeaf composition. Note that if both warnings and errors occur then the error action is taken. The choices are:

- ignore** This is only available for warnings. The processor will act as if there were no warnings. This can affect actions such as copying the composition log.
- continue** The processor will continue with the request and create composition output as appropriate.
- fail** The processor will stop processing the request and not produce any composition output.

Note:

The above only applies to an [output](#) with **content** equal to “compose”. The composition log can always be processed.

Note:




There is no way to configure the action taken if a fatal error occurs during composition. This always prevents composition output from being created.

The [condition](#) attribute can be used to give more explicit control of when an output is produced.


### 2.2.3 Queues Page

This page allows you to maintain the [execution queues](#) that process requests.

Each queue is displayed with a colour that indicates its state, as follows:

-  The queue is idle and ready to accept a request.
-  The queue is currently processing a request.
-  The queue is idle and not accepting requests.

A queue is identified by its **Name**. If the queue is currently processing a request, the request identifier will appear in the **Executing** column.

Press the  button to display information about the request that is currently running (if any). The information page also allow you to terminate the request prematurely (use with care!).

**Warning:**

**Terminating a request may not stop the associated processes, such as the TopLeaf composition engine.**

The **Users** and **Request Types** fields determine the requests that a queue can process. See [Section 5.1](#) for more information on the queue policy.

Check the **Disable** box to prevent the queue from processing any further requests.

The **Remove** box is only turned on if the queue is disabled and not processing a request. The procedure for removing a queue is as follows:


- Check the **Disable** box and press .
- If a request identifier is shown under Executing you need to wait until the request is processed. Use the reload button of the browser periodically to check.
- When the queue is disabled and not busy you will be able to check the **Remove** box. Press  to complete the removal.

The  and  buttons provide a quick way to modify all queues. These buttons don't appear if there is only one queue. You still need to press Update to change the queues.

The  button allows you to create a new queue.

#### **Pending requests**

Any requests that are waiting for a queue to become available are displayed at the bottom of the page. The order of this list is the same order they will be searched when a queue is looking for a request to execute.

Press the  button to display information about the request. The information page also provides buttons for the following actions:

**Move to start** Moves the request to the start of the list. This will make it the first request a queue examines when looking for a request to execute.

**Move to end** Moves the request to the end of the list.

**Remove** Removes the request from the list. Once a request has been removed it cannot be restored.

### 2.2.4 Email Configuration

These settings are only required if requests will be sending email messages.

The fields under [SMTP Host](#) are used to connect to the host that will be used to send the email. The

Address field is mandatory. The Port Number can be left blank if the default port (25) is to be used. The User Name and Password fields may be left blank if the host does not require authentication.

The remaining fields are optional, and are used only if the request does not supply the relevant information. The From field identifies the sender of the email and the Reply To field can be used to control where replies are sent. The Default subject line appears on the email if the request does not provide a subject.

Note:

Email messages must have a "from" address, so it is recommended that you set a default value in case the request does not specify one.

### 2.2.5 Mapped Drives Page

This page allows you to create and maintain mapped drives that are available to requests. For example, using a mapped drive allows a destination to be specified as **D:\manual.pdf** instead of **\\server31\data\manual.pdf**.

To create a mapped drive, click the Add mapped drive link and fill in the form. The drive must be a single upper case letter (a trailing colon is optional). The location must be the UNC path of a share that is visible on the network. The user name and password are used to connect to the share and must have suitable access rights.

Note:

The user name and password are held in clear text in the configuration file. You may wish to create a special account with restricted privileges that is used to connect mapped drives.

To remove a mapped drive tick the **Remove** checkbox and press Update. Changes to mapped drives that are already connected, including removal, will not take effect until the server is restarted.

### 2.2.6 Debugging Page

This page allows you to control options that may assist with diagnosing and resolving problems.

Tick the **Keep temporary partitions box** to prevent the request processor from removing temporary partitions when a request is complete. This will allow you to investigate the partition using the standard TopLeaf interface.

Tick the **Keep temporary data box** to prevent the request processor from deleting temporary data used by the request.

Note that the above two settings stay in force only until the server is restarted, at which time they are both turned off.

The **Lowest level for summary log** and **Lowest level for detail log** fields allow you to control the amount of information written to the summary and detail logs, respectively. The following table shows the types of messages selected by each value (note that error messages always appear in both logs).

Level	Messages
WARNING	Warning messages only.
INFO	Warnings and informative messages.
FINE	Warnings and informative messages, plus some internal messages.
FINER	Warnings and informative messages, plus more internal messages.

Level	Messages
FINEST	Warnings and informative messages, plus all internal messages.

**Note:**

Changing the message level for a log only affects how it records messages in the future. It has no effect on the messages already in the log.

### 2.2.7 Log Pages

The summary page shows recent log messages, including the request (if any) that gave rise to the message. Warning messages appear with a yellow background; severe messages are bold and shaded in red.

The detail page shows a more detailed view of the server log, including informative messages. This view also contains the composition log for any request that generated composition warnings or errors. At the bottom of the page is a link that allows you to download the entire log, which usually contains more than what is displayed on this page. The download contains a single file in a ZIP archive. The amount contained in the detail log can be controlled by changing the `<log>` element in the [configuration file](#).

## 2.3 Server Configuration

The server stores its configuration information in an XML document. The location of the file containing this document is determined by the `wrapper.conf` file (see [Section 2.1.1](#)). By default this file is called `config.xml` in the directory where you installed the server.

**Note:**

Most of the information in the configuration file can be viewed and modified using the [server console](#).

The server reads information from the XML document by looking for elements with specific names, as described below. The document must contain at most one instance of these elements. You may add other content to this document (other elements, comments, etc.). When the server updates the document any addition content will be preserved, although the format (e.g. whitespace) may be modified.

**Warning:**

**Do not modify the file containing the XML document while the server is running, since it can rewrite the file at any time. Stop the service before making any changes.**

The root element of the document must be called `<config>`. This may contain the following attributes:

**request-port** The port number used for making requests. Defaults to 61577.

**control-port** The port number used for accessing the control console. Defaults to 61578.

**max-connections** The maximum number of simultaneous connections to the server that will be permitted. (If this number is reached subsequent connection attempts will block until one of the current connections is closed.) Default value is 3.

**timeout** The timeout value for reading the request in milliseconds. A value of zero means that there is no timeout. The default value is 20000 (20 seconds).

**temp-dir** The path to the directory used for storing temporary files created while processing requests. If this attribute is not present, the temporary directory will be called **temp** and will be located in the same directory that contains the configuration file.

The following describe the other elements that contain configuration information. Note that they may occur anywhere and in any order in the document.

**<log>** This is used to set options for the logs produced by the server. Note that the summary log is held in memory and can only be accessed by the [server console](#). The detail log is stored in a file, so it can be inspected even if the server is not running. The location of the detail log is set in the [service configuration](#) file.

The **size** and **count** attributes are used to limit the total size of the detail log. When the current log file reaches the number of bytes specified by **size** it is “rolled” and a new file is started. Versions of the log file are identified by a numeric suffix, where “.0” is the current file. The number of versions kept is determined by the value of **count**. The size value may be specified as a number of kilobytes by using the “k” suffix or megabytes by using “m”.

The **summary** and **detail** attributes are used to control the amount of information written to the logs. If present they must have one of the values WARNING, INFO, FINE, FINER or FINEST, where WARNING produces the least information and FINEST produces the most. See [Section 2.2.6](#) for a description of message levels.

**<topleaf>** This is used to set default values for TopLeaf composition runs. These can be overridden for a specific request by values in its [descriptor](#).

The **repository** attribute contains the path to the repository to be used if the request does not specify a repository.

The **publication** attribute contains the path to the publication used to compose the data if the request does not specify a publication. Note that this is a path relative to the repository root, and should use '/' to separate the components of the path.

The **partition** attribute contains the path to the partition used to compose the data if the request does not specify a partition. Note that this is a complete path relative to the repository root (including the publication), and should use '/' to separate the components of the path.

The **compose-warning** attribute specifies the default behavior when TopLeaf composition generates a warning. A value of **ignore** means that warnings are ignored. A value of **continue** means that processing will continue, but the warning will be reported. A value of **fail** means that processing will terminate and no outputs will be produced.

The **compose-error** attribute specifies the default behavior when TopLeaf composition generates an error. The values are the same as for **compose-warning** except that the **ignore** value is not allowed.

The **<topleaf>** element can also contain a number of **<repository>** elements, each of which specifies a named repository location. Each one of these must have a **name** and **location** attribute, containing the name and absolute path to the repository, respectively.

**<login>** If this element is present, access to the server console will be password protected.

The **user** and **password** attributes contain the user name and password, respectively.

**<queues>** This element, if present, is a container for the **<queue>** elements.

If there is no **<queues>** element a single queue named “main” will be created.

- <queue>** Each queue element defines a request processing queue.
- The **name** attribute is mandatory and must contain a non-empty name for the queue.
- The optional **user** attribute contains a space-separated list of user names. These can be used by the [queue policy](#) to determine which requests may be executed on the queue.
- The optional **type** attribute contains a space-separated list of request types. A request type is an arbitrary string that describes the nature of the request (e.g. “urgent” or “draft”). These can be used by the [queue policy](#) to determine which requests may be executed on the queue.
- The **enabled** attribute indicates whether the queue can execute requests. If this has the value “no” the queue is disabled.
- <email>** This is used to configure the sending of email messages. This can be ignored if your requests do not send email.
- The **smtp-host** attribute contains the address of the host used to send mail. This is required.
- The **smtp-port** attribute can be used to override the default port number (25) for the SMTP server.
- The **user** and **password** attributes provide login information for the SMTP host. These may be omitted if the host does not require authentication.
- The **from** and **reply-to** attributes set the corresponding email headers. These provide default values if the request does not specify them. Note that a “from” address is required in order to send a message.
- The **subject** attribute provides a default value for the email subject if the request does not specify one.
- <database>** This is used to enable the embedded database use to collect statistics of the server operation. Some additional third-party libraries are required to use this facility. Contact Turn-Key support for more information.
- The **type** attribute controls the database configuration. The possible values are:
- **none** (the default) disables the database
  - **internal** starts a database that can only be accessed by the server
  - **public** starts a database that can be accessed by other applications
- The **path** attribute determines where the database files are created. If this is a relative path, the files are created relative to the directory containing the configuration file. The default value is “statistics/requests”.
- The **port** attribute specifies the port used to access the database if the public configuration is chosen. The default is “9001”.
- The **checkpoint-count** attribute controls how often a checkpoint operation is performed on the database. A checkpoint clears the transaction log and compacts the database files. The checkpoint is performed when the specified number of requests have been processed. A value of zero prevents checkpointing.
- <cleanup>** This controls the [temporary file clean-up process](#).
- The **delay** attribute specifies the number of *minutes* to wait after the server starts to initiate the clean-up. If this value is less than or equal to zero the clean-up will not run.

The **repeat** attribute specifies the number of *minutes* to wait before running the clean-up again. If this value is less than or equal to zero the clean-up will only run once. Regardless of the attribute value, the clean-up will never run more often than once every ten minutes.

The **age** attribute specifies the minimum age in *days* of files to be deleted by the clean-up. A value less than or equal to zero will be interpreted as one day.

**<classes>** This element is used to specify the names of user-supplied classes which override standard behavior of the server. See [Chapter 7](#) for more information.

Note that all class names must include any relevant package names.

The **request-parser** attribute contains the name of a class used to create RequestParser objects.

The **request-handler** attribute contains the name of a class used to create RequestHandler objects.

The **response-encoder** attribute contains the name of a class used to create ResponseEncoder objects.

The **queue-policy** attribute contains the name of the class that implements the [queue policy](#).

**<console>** This element is used to control the behavior of the [server console](#).

Each instance of a **<user-page>** element will result in a new user-defined page being added to the console. See [Section 7.2](#) for more information. This element can contain the following attributes:

- The **class** attribute is mandatory and specifies the name of the class used to create the page handler object. It is possible for the same class to implement multiple pages by examining the URL in the HTTP request. See [Chapter 7](#) for information about making your own classes available to the server.
- The **url** attribute is optional and specifies the path component of the URL used to access the user page. The prefix `"/user/"` is added automatically, so an attribute value of `"page1"` will result in a URL path of `"/user/page1"`. If this attribute is omitted the page handler class must override the **urlPathMatch** method.
- The **menu** attribute is optional and specifies the name of the link to this page displayed in the navigation menu. The name can also be specified by overriding the **getMenu** method in the page handler class. If no menu name is specified the page will only be accessible by linking to it from another user-defined page.

**<windows>** This element contains information specific to the Windows operating system.

The **<mapped-drive>** element contains information about a mapped network drive in the following attributes:

- **drive** contains the (single letter) drive name
- **location** contains the UNC path to the network resource
- **user** and **password** contain the account details used to establish a connection.

## 3. Server Administration

The following describes administrative procedures for the server.

### 3.1 Updating the Server

The server must be stopped when you need to make any of the following changes:

- Updating the software (either the server itself or TopLeaf).
- Adding new java classes or modifying the [classpath](#) used to locate classes.
- Adding or changing server [customisations](#).

To stop the server you can use the Services utility which is a part of Windows, or you can use the [Server Setup](#) program. This will attempt to shut down the server cleanly without losing any request as described [below](#). Note however, that this procedure may abort synchronous requests.

To make sure that no requests are executing or waiting to execute, follow this procedure:


- Open the [server console](#) and select the [Server Status](#) page.
- Press the  button to prevent the server from accepting any more requests.
- When the **Requests executing** and **Requests pending** fields are both zero the server is idle and can be safely stopped. Use the reload button of the browser to update the page if necessary.

### 3.2 Updating TopLeaf Stylesheets

If you need to update any of the TopLeaf stylesheets, you must ensure that there are no requests executing that may be affected.

The first step is to determine the queues that may be affected by the update. Updating a stylesheet may only affect some queues, depending on which requests can be executed on them.

To make sure that one or more queues are idle, follow this procedure:

- Open the [server console](#) and select the [Queues](#) page.
- Check the **Disabled** box for all of the affected queues. The  button is a quick way to disable all of the queues.
- Press the  button to apply the changes to the queues.
- When all of the affected queues are shown as idle and not accepting requests (  ) it is safe to apply the TopLeaf updates. Use the reload button of the browser to update the page if necessary.

After making the update, enable all queues and press  to activate them.

### 3.3 Shutdown Procedure

When a shutdown is initiated the server does the following:

1. Stop accepting any further requests.
2. Disable all queues, so they will not process any further requests. Requests that are currently being processed will continue normally.
3. Any synchronous requests that are waiting to start processing are aborted. (This is necessary because the connection to the client will be lost when the server stops.)
4. Any asynchronous requests that are waiting to start processing are saved and removed. When the server next starts it will restore these requests before it accepts any new requests.
5. Wait for all requests that are currently being processed to complete.

6. Write the message “Shutdown completed normally” to the log.

### 3.3.1 Restoring Requests After a Shutdown

The requests that have been saved are stored in a directory called **restart** in the same directory that contains the [configuration file](#).

Each request is stored in a separate file, the name of which is a six-digit sequence number followed by the request identifier. The sequence number determines the order in which the requests will be restored.

These files can safely be renamed or removed before starting the server. This allows you to control the requests that will be restored when the server starts.

### 3.4 Temporary File Clean-up

The server can periodically check for temporary files older than a specified number of days and delete them. This check is made in the temporary directory (see the **temp-dir** attribute in the [server configuration](#)) and the [restart directory](#).

The clean-up process checks the modification time of each file or directory in the checked directories. If it is older than the specified number of days, it is removed. If it is a directory, the entire sub-tree is removed (note that the ages of descendent files and directories are not checked).

By default the clean-up process runs one hour after the server starts, and then repeats on a 24-hour cycle. It removes all files older than 7 days at the time of checking.

The parameters for the clean-up process can be modified by changing the **cleanup** element in the [server configuration](#). To disable the clean-up process, set the **delay** attribute value to “0”.

### 3.5 Troubleshooting

The summary and detail logs in the console should provide sufficient information to allow you to diagnose and rectify any problems that occur.

The detail log is stored in the directory in which the application was installed. A new version of the log file is created once it reaches a certain size. Only the most recent version of the log can be accessed via the console.

The detail log files have names beginning with **server.log**. A suffix is appended to indicate the version. The most recent version normally has suffix “.0”.

Note that the console allows you to control the level of detail in the logs on the [debugging page](#).

It is possible that low-level errors will not be recorded in the detail log. This is particularly true when the server application is first starting. Low-level errors are written to a separate log; these are stored in files with names beginning with **wrapper.log**.

## 4. Protocol Descriptions

The following describes the protocols that client applications use to communicate with TopLeaf Server. This covers both the *request protocol* (used to send information to the server) and the *response protocol* (used to receive information from the server).

Note:

The following describes the “standard” protocols implemented by the default java classes. You can implement variations to these protocols, or completely different protocols, by installing customised java classes as described in [Chapter 7](#).

### 4.1 Request Protocol

A request sent to the server is a stream of bytes as shown in the following diagram:



The start byte is the first in the request and must have the value 57 hexadecimal.

The second byte indicates the version of the protocol being used and currently must have the value 1.

Next is a sequence of 8 bytes that indicates the length (in bytes) of the first **section**. The most significant byte occurs first. The value is constructed by concatenating the 8 bytes to form a 64-bit integer. This integer must not be negative when interpreted as a twos-complement binary value (i.e. the most significant bit must be zero).

The section data follows the section length. A section contains data to be passed to Toplevel for processing. The way sections are interpreted is determined by the [request descriptor](#).

The request can contain zero or more sections after the first section. Each section starts with an 8-byte length field followed by the section data.

The request is terminated by the **terminator**, which is a sequence of 8 bytes all with value 0.

The server assumes that the first section contains XML data and will always attempt to parse it.

The default request parser will inspect the XML data to determine if it contains a [request descriptor](#) (by examining the namespace of the root element).

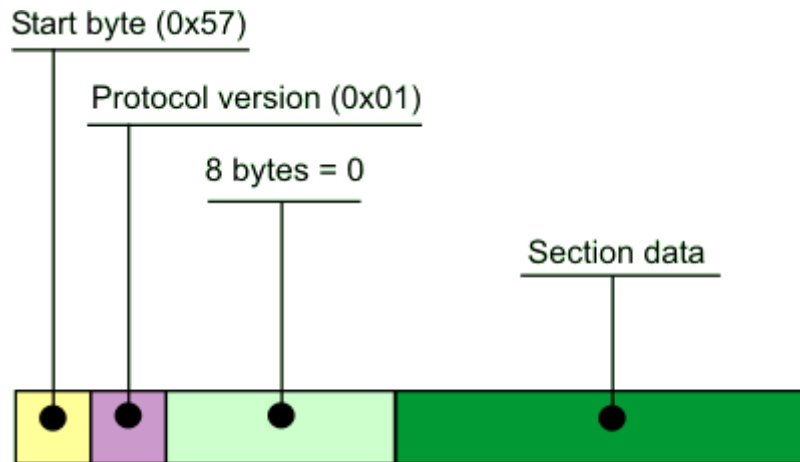
#### 4.1.1 Abbreviated Requests

An abbreviated form of the request may be used if:

- The data to be processed is an XML document that contains all of the information required for TopLeaf to process it; and

- The processing options such as the repository and publication to use can be determined from the server defaults or by inspecting the data in the request.

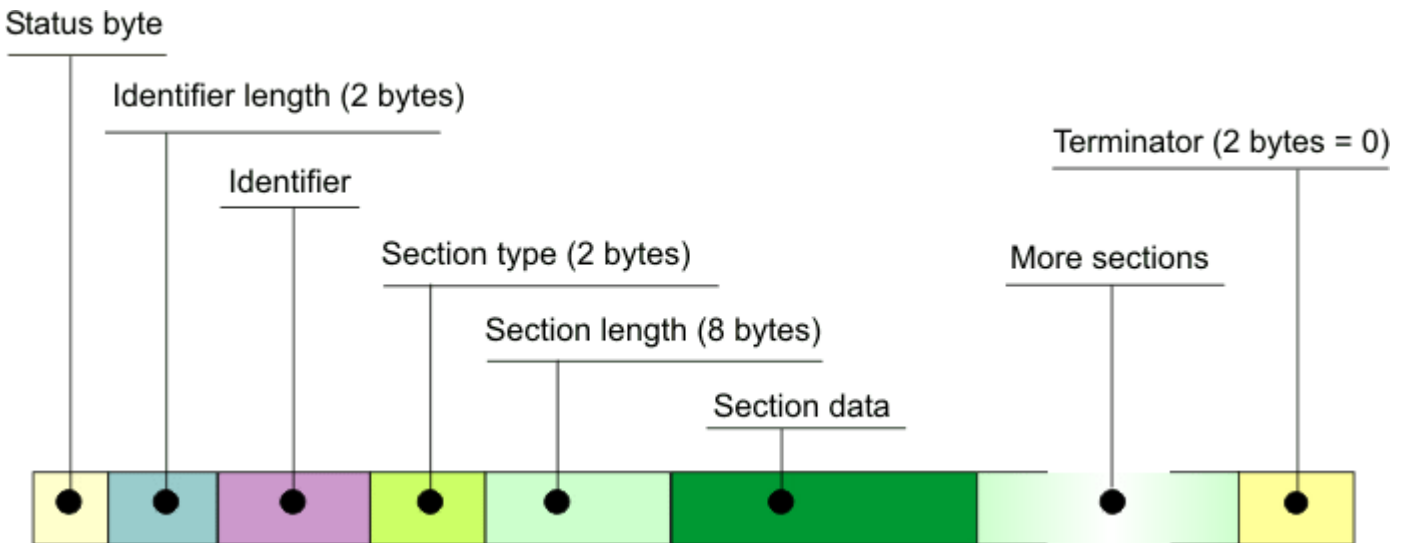
The abbreviated request is shown in the following diagram:



In an abbreviated request the field which normally contains the first section length has the value zero. The section data consists of all of the following bytes to the end of the stream.

## 4.2 Response Protocol

A response received from the server is a stream of bytes as shown in the following diagram:



The status byte contains the following bit fields:

Mask	Description
0x80	This bit is set if an error occurred when processing the request.
0x03	These two bits show the status of the TopLeaf composition run: <ul style="list-style-type: none"> <li>• 0 = no errors or warnings</li> <li>• 1 = at least one warning</li> <li>• 2 = at least one error</li> <li>• 3 = fatal error (composition did not complete)</li> </ul>

The identifier field contains the unique identifier assigned to this request. The number of bytes in the field is contained in the identifier length field (2 bytes, most significant byte first). The identifier is encoded using utf-8 and does not have a null terminator.

Following the identifier are zero or more sections, each containing data returned by the server. Sections are similar to those in the request protocol, except that they also contain an indication of the type of data they contain.

A section starts with a 2 byte field for the data type followed by an 8 byte field containing the number of bytes in the data. Both of these fields have the most significant byte first.

The data type field may contain the following values:

<b>0x0001</b>	PDF data
<b>0x0002</b>	HTML data
<b>0x0003</b>	RTF data
<b>0x0004</b>	text data

### 4.3 Request Descriptor

The request descriptor is an XML document that contains information about the request. When the descriptor is present it must be in the first section of the request.

The root element of the descriptor must use the following namespace:

<http://www.turnkey.com.au/topserve/request>

If the root element uses a different namespace (or has no namespace) it is assumed to be an XML document to be composed by TopLeaf.

The default request parser extracts information from the descriptor by looking for the elements and attributes with the names listed below. Apart from the restrictions stated below, the location of these elements in the document is not important. For a recommended but not mandatory document structure for the descriptor, see [Section Appendix A](#).

#### 4.3.1 The `request` element

This element is normally the root element of the document. It may contain the following attributes:

<b>id</b>	If this attribute is present it contains a string to be used as the unique identifier for this request. If it is not present the server will generate a unique identifier.
<b>type</b>	Indicates the type of request. This can be used by the queuing policy as described <a href="#">below</a> .
<b>user</b>	Identifies the source of the request. This can be used by the queuing policy as described <a href="#">below</a> .

#### 4.3.2 The `process` element

This element contains information about how the request is to be processed in the following attributes:

<b>mode</b>	This attribute must have the value <b>sync</b> (for a synchronous request) or <b>async</b> (for asynchronous). The difference between these two modes is explained in <a href="#">Chapter 1</a> . If this attribute is not present the request is processed synchronously.
<b>queue</b>	This attribute is intended to provide information that the <a href="#">queuing policy</a> may use to decide the queue used to execute this request.

### 4.3.3 The `topleaf` element

This element contains information used to control the operation of TopLeaf. This information overrides the default TopLeaf settings in [Section 2.2.2](#). The attributes allowed in this element are:

**repository** This identifies the TopLeaf repository to be used. It may be either the path to the root of the repository, or one of the named repository locations set in [Section 2.2.2](#).

**publication** This identifies the TopLeaf publication to use. It must be the path to the publication relative to the repository root. Use the slash character to separate path components (e.g. "Manuals/UserGuide").

**partition** This identifies the TopLeaf partition to use. It must be the path to the partition relative to the repository root. Use the slash character to separate path components (e.g. "Manuals/Administrator/Current").

If the partition path is missing or is the empty string a temporary partition will be created in the nominated publication. The publication must contain a partition called **Template** which is copied to create the temporary partition.

**compose-warning** This specifies the action to take if a composition warning occurs. If both warnings and errors occur the error action is taken. The possible values are **ignore**, **continue** and **fail**. This overrides, and has the same meaning as, the **Action on composition warning** setting in [Section 2.2.2](#).

**compose-error** This specifies the action to take if a composition error occurs. The possible values are **continue** and **fail**. This overrides, and has the same meaning as, the **Action on composition error** setting in [Section 2.2.2](#).

The `topleaf` element may also contain any number of `<set>` elements. Each one of these is used to set an initial value for a TopLeaf variable. The **var** attribute contains the name of the variable to set and the **string** attribute contains its value. The normal rules for variable names apply (consult the TopLeaf Mapping Guide for details). In particular, note that variable names must begin with an upper case letter.

The following is an example that sets the publication to use and the initial value of the "Distribution" variable:

```
<topleaf publication="Client/UserGuide">
  <set var="Distribution" string="draft"/>
</topleaf>
```

### 4.3.4 The `output` element

This element indicates how to process output from TopLeaf. It may have the following attributes:

**content** This identifies the source of the output data as follows:

Value	Default format	Description
compose	pdf	Page data produced by the composition engine.
log	text	The composition log.
toc	xml	The generated table of contents. See the TopLeaf User Guide for information about the format of this data.
index	xml	The generated index. See the TopLeaf User Guide for information about the format of this data.
xref	xml	The generated cross reference. See the TopLeaf User Guide for information about the format of this data.

Value	Default format	Description
pageinfo	xml	Information about the generated pages (the “live pages” list). See the TopLeaf User Guide for information about the format of this data.

The default value is **compose**.

The page data is not used to create the output if the composition fails. Warnings and errors can be treated as a failure; this can be controlled by a [configuration parameter](#) or with the `<topleaf>` element. A fatal composition error is always treated as a failure.

**format** This identifies the type of output to create. The legal values are **pdf**, **html**, **rtf**, **text** or **xml**.

This attribute is optional. If omitted, the default value depends on the value of **content** (see table above). Some content values may only allow a subset of the possible format values.

See [below](#) for information about setting options for output formats.

**condition** This determines whether the output will be processed. The table [below](#) describes its possible values. If this attribute is omitted the output is always processed.

**fail-content** This determines what happens if the output data source is not available. It effectively replaces the content attribute in the event of failure. Its value may be **log** (the default) to use the composition log or **none**. In the latter case no output is created in the event of failure.

This attribute is only used when the output format is **pdf**.

**dest-path** This specifies the complete path (including the file name) for the output file.

**dest-dir** This specifies the path to the directory in which the output file will be created. The name of the file will be taken from **dest-name** if present, or generated if not.

**dest-name** The name of the output file. This value is ignored unless **dest-dir** is specified.

**profile** If the output format is **pdf**, this can be used to select the pdf profile to use. (See the TopLeaf User Guide for information on PDF profiles.)

**properties** A space-separated list of name/value pairs used to control the output creation. Each property is specified in the form **name="value"**.

See the description of the **Set Transform Property** method in the *TopLeaf API Manual* for more information. The current implementation does not support the use of properties for PDF creation.

**changed-only** If the output format is **pdf**, this attribute can be set to “yes” to indicate that the PDF should contain only those pages that are different from the previous release. Note that this will have no effect unless at least one release has been made for the partition. See “Release and page management” in the *TopLeaf User Guide* for more information.

**appendlog** If the output format is **pdf**, this attribute controls whether a rendition of the composition log is appended to the PDF output. The value “never” (the default) prevents the log from being appended, while “always” appends it unconditionally. To append the log conditionally use one of the values from the table [below](#).

This feature does not work if the PDF profile specifies an owner password.

**response** This optional attribute is either “yes” to include the output in the response sent back to the client, or “no” to exclude it. The default is “yes”. Note that this attribute is ignored for an asynchronous request.

Any occurrence of the string “{request-id}” in the **dest-path**, **dest-dir** or **dest-name** attributes is replaced by the request identifier. The identifier generated by the server is guaranteed to be suitable to use in a file path; if you assign your own identifier make sure it will not cause the output destination to be invalid.

Output data can be attached to email messages as described [below](#).

See [below](#) for some examples of output elements.

Note:

An output element with no attributes will generate a PDF rendition of the composition data and (for a synchronous request) include it in the response.

### Output Format Options

To control options for PDF output, create an appropriate **PDF profile** and invoke it using the **profile** attribute. See the TopLeaf User Guide for information on pdf profiles.

Other output formats are controlled by the default options set for the request partition. See “Secondary output formats” in the TopLeaf User Guide for more information. Note that the TopLeaf GUI can only be used to set default options for “permanent” partitions. Requests that use temporary partitions (i.e. if no partition is named in the request descriptor or TopLeaf defaults) cannot have default options set for them. Contact Turn-Key support for more information regarding this restriction.

Note:

Images are not currently supported in HTML output.

#### 4.3.5 The **print** element

This element controls printing of the TopLeaf output. Any number of print elements may appear. To produce multiple copies of the output, add several print elements each specifying the same printer.

The only attribute allowed is **printer**, which must contain the network address of the printer to use.

#### 4.3.6 The **email** element

The **<email>** element may be added to the descriptor in order to send an email message. The email messages can have output data attached to them as described [below](#).

Note:

In order to send email, you must provide [information about the email host](#). An email message must have at least one “to” address and a “from” address before it can be sent. The from address can be set in the configuration or in the **<email>** element of the descriptor.

The recipient address or addresses can be set either as attributes of the **<email>** element, or as the content of child elements contained in the **<email>** element. The attribute and element names of the recipients are **to** (main recipient), **cc** (carbon copy) and **bcc** (blind carbon copy). Address values may contain comma-separated lists.

The following two **<email>** elements are equivalent:

```
<email to="fred@foo.org, jim@foo.org"/>
```

```
<email>  
  <to>fred@foo.org</to>  
  <to>jim@foo.org</to>  
</email>
```

The **from** and **reply-to** attributes are used to set the corresponding header values. Note that an email message must have a “from” address, but the “reply to” address is optional.

The **subject** attribute can be used to set the subject text for the message. A default subject can also be set in the [email configuration](#). Any occurrence of the string “{request-id}” in the subject is replaced by the request identifier.

The **id** and **ref** attributes are used for [attaching output data](#) to the message. If the **ref** element is present the element is treated as a placeholder for another `<email>` element (the one whose **id** attribute value matches the **ref** attribute value). The latter element controls the email message (anything in the placeholder element relating to the message is ignored).

The **filename** attribute is used to set the file name associated with an attachment. If this not present, the name of the temporary file is used.

### ***Setting the email message content***

To set the text content of the email message, add a `<body>` element as a child of the `<email>` element. The content of the `<body>` element is used as the text of the message.

The **format** attribute of the `<body>` element can be either **plain** (the default) or **html**. This determines whether the content is to be interpreted as plain text or HTML data.

#### **Warning:**

**The content of the `<body>` element must follow the rules of well-formed XML data. Characters such as ‘&’ and ‘<’ must be escaped. Alternatively, you can use a CDATA section for all or part of the content.**

Any occurrence of the string “{request-id}” in the text is replaced by the request identifier.

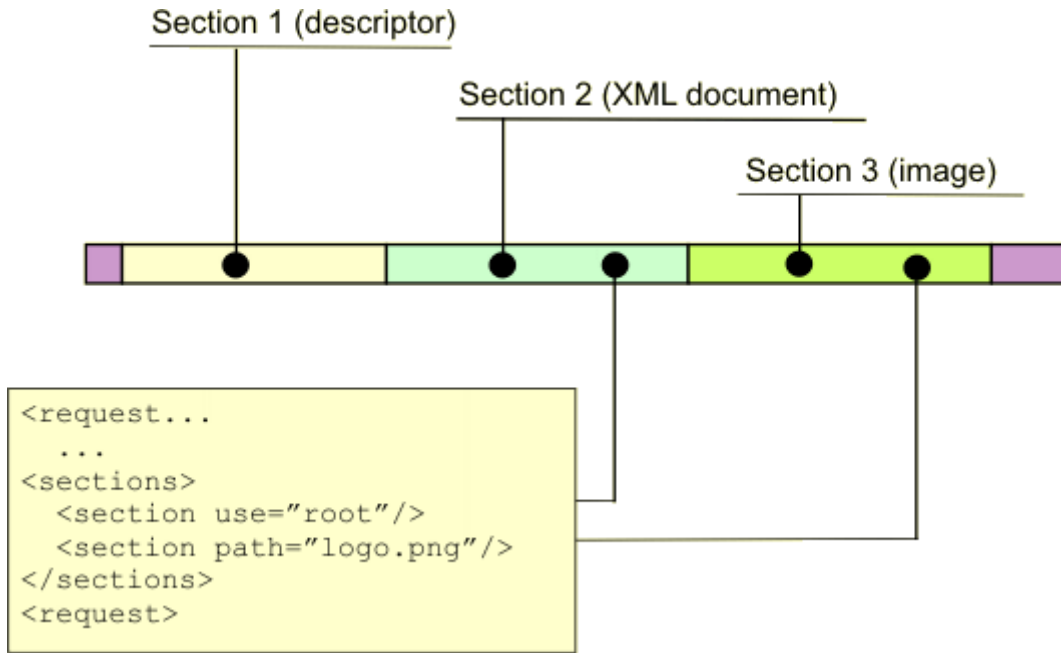
### **4.3.7 The `sections` element**

The only purpose of this element is to contain the **section** elements. It does not contain any additional information.

### **4.3.8 The `section` element**

Each section element corresponds to a section in the request as described in [Section 4.1](#). The only request section that does not have a corresponding section element is the first, which contains the request descriptor.

The following diagram shows the relationship between the request sections and the descriptor for a request that contains an XML document and an image file:



Note that the order of the section elements in the descriptor must match the order in which the sections occur in the request.

The section element may contain the following attributes:

- use** This indicates the way the section data will be used.  
 A value of **root** indicates that the data contains an XML document to be used as the input to TopLeaf (either directly, or within an archive).  
 A value of **stylesheet** indicates that the data should be copied into the TopLeaf publication before composition. A value for the **path** attribute must be present and indicates the destination for the data relative to the publication directory.
- format** This indicates the format of the section data. It is only important if the data must be processed in a specific way before being passed to TopLeaf.  
 A value of **zip** indicates that the data contains a zip archive.
- path** This indicates the destination to which the section data is to be copied. It must be a relative path.
- archive-root** This is required if the data is a multi-file archive and contains the “root” XML document to be processed by TopLeaf. It must contain the relative path to the root file within the archive.  
 If this attribute is present it implies **use="root"**.

**Locating the root XML file**

The section that contains the “root” XML document is determined as follows:

- if any section elements have **use="root"** or the **archive-root** attribute, then the first one found contains the root data
- otherwise, the first data section (i.e. the one after the descriptor) is assumed to contain the root data.

**4.3.9 Condition values**

This table describes the effect of the possible values for the **condition** attribute of the **output** element. If the condition is not satisfied the output element is ignored.

Attribute value	Output is processed...
<b>always</b>	always
<b>ifwarn</b>	if there were warnings or errors in the composition
<b>iferr</b>	if there were errors in the composition
<b>iffail</b>	if the composition failed to complete (abnormal exit)
<b>nowarn</b>	if there were no composition errors or warnings
<b>noerr</b>	if there were no composition errors
<b>nofail</b>	if the composition terminated normally

If the condition attribute is missing, or has a value that does not match any of the above, the output is processed.

Any attribute value that has one of the above as a prefix will match. For example, “ifwarnings” will produce the same result as “ifwarn”.

Note that setting **compose-warning** to **ignore** in the `<topleaf>` element will effectively prevent warnings from occurring.

#### 4.3.10 Output examples

The following are examples of `<output>` elements and the result of each.

<pre>&lt;output dest-dir="C:\Temp"/&gt;</pre> <p>This creates a PDF file in C:\Temp using a generated name.</p>
<pre>&lt;output format="html" dest-path="C:\Temp\{request-id}.htm"/&gt;</pre> <p>This creates an HTML file using the request identifier as the file name.</p>
<pre>&lt;output content="log" condition="ifwarning" dest-dir="C:\Log"/&gt;</pre> <p>This creates a copy of the composition log if there were any composition warnings or errors.</p>
<pre>&lt;output format="pdf" condition="noerrors" dest-dir="S:\Out"/&gt;</pre> <p>This will create a PDF file as long as there are no composition errors (warnings will not prevent the file from being created).</p>

#### 4.3.11 Attaching output to email messages

The output data created by the request can be attached to an email message by adding an `<email>` element as a child of the `<output>` element. For example, the following sends an email message with an attached PDF:

```
<output format="pdf">
  <email to="joe@foo.org" subject="TopLeaf output"/>
</output>
```

(see [above](#) for more information about the `<email>` element).

The `<email>` element can be a placeholder that refers to another element by using a **ref** attribute that matches the **id** attribute in the target `<email>`. This can be used to add several attachments to a single message.

When using placeholder elements, the `<email>` elements do not need to be contained in `<output>` elements. The following sends a message with two attachments:

```
<output format="pdf">
  <email ref="M1"/>
</output>

<output content="log">
  <email ref="M1"/>
</output>

<email id="M1" to="Jim@foo.org"/>
```

The two techniques of creating attachments can be combined. In the following example, two messages are sent. The one sent to Jim contains both the PDF and the composition log. Fred's message contains only the log.

```
<output format="pdf">
  <email ref="M1" filename="userguide.pdf"/>
</output>

<output content="log">
  <email ref="M1"/>
  <email to="Fred@bar.org" filename="topleft.log"/>
</output>

<email id="M1" to="Jim@foo.org"/>
```

(Note the use of the **filename** attribute to associate a specific file name with the attachment. This is typically the default name used by the email client if the user saves the attachment. If it is not present the temporary file name used to generate the file on the server will be used.)

#### 4.3.12 An example descriptor

The following is an example of a complete descriptor:

```
<?xml version="1.0"?>
<request xmlns="http://www.turnkey.com.au/topserve/request">
  <process mode="sync">
    <topleft repository="C:\Test\TopLeaf" publication="UserGuide"/>
    <output/>
    <output format="rtf" dest-path="C:\Output" response="no"/>
    <output content="log" condition="ifwarning"/>
    <print printer="LPT1:"/>
  </process>

  <sections>
    <section format="zip" archive-root="doc/chap1.xml"/>
    <section path="logo/cover.png"/>
  </sections>
</request>
```

This example is intended to show some of the possible options in a descriptor; your application may not need to use all of these options.

The effect of this descriptor is as follows:

- The `<process>` element sets the mode of the request to synchronous.
- The `<topleft>` element sets values for the repository and the publication within the repository.

- The first `<output>` element uses the default options, namely to create a PDF rendition of the composition data and returns it in the response.
- The second `<output>` element creates the composition data in RTF format and writes it to a directory; this data is not returned in the response.
- The final `<output>` element includes the composition log in the response if there were any warnings or errors. When a condition attribute is used the client needs to examine the response to determine the number of sections that have been returned. (In this example, a successful request will return either 1 or 2 sections.)
- The `<print>` element sends the composition data to the nominated printer.
- The first `<section>` element says that the first section after the descriptor contains a multi-file archive. The server will extract this, preserving its directory hierarchy and pass the file whose relative path is specified by **archive-root** to TopLeaf.
- The second `<section>` element says that the next section contains an image file which is copied to the relative location specified by the **path** attribute. If the archive already contained a file at this location, the new file would replace it.

## 5. Queueing

When a request is received it is placed in a first-in first-out list of pending requests. There are a number of **execution queues** managed by a single **queue manager**.

When an execution queue is idle it asks the queue manager for a request to execute. The manager will select the first pending request which matches the selection criteria for the queue.

The **queue policy** is responsible for deciding whether a request can be executed on a particular queue. A customised queue policy can be used by creating a class that implements the **au.com.turnkey.toplevel.server.queue.QueuePolicy** interface. See [Chapter 7](#) for more information.

The following section describes the default queue policy.

### 5.1 Default Queue Policy

The default queue policy determines if a request can be executed on a queue as follows:

- If the request specifies a **queue** attribute in the **<process>** element of its descriptor, it can only be executed on the queue with name matching the attribute value.
- If the queue defines a list of user names the request must specify one of the names in the list. The request user name is specified by the **user** attribute in the **<request>** element of the descriptor.
- If the queue defines a list of request types the request must specify one of the types in the list. The request type is specified by the **type** attribute in the **<request>** element of the descriptor.

If none of the above tests apply for a specific request and queue, then the queue can execute the request.

In the default case, where requests do not nominate a specific queue and queues don't specify users or types, a request can be run on any queue.

## 6. Writing a Client

A number of sample clients are included in the installation. You can find a link to a web page describing them in the Windows Start menu. All of the sample code may be freely copied and modified for use in any application that communicates with TopLeaf Server.

If you are writing client code in java there are some convenience classes you can use. These classes are available in both **server.jar** and **server-client.jar**.

The following is a brief summary of the classes that you may find useful. For more information, consult the API documentation. A link to this can be found in the Windows Start menu.

### *Package au.com.turnkey.toplevel.server.util*

**RequestConstants** contains constant values used when sending a request.

**RequestOutputStream** extends the **java.io.BufferedOutputStream** class to provide methods for writing components of a request.

**RequestInputStream** extends **java.io.BufferedInputStream** to add methods that read components from a server response stream.

**FileUtil** contains methods for creating a ZIP archive that mirrors a directory hierarchy.

### *Package au.com.turnkey.toplevel.server.response*

**DefaultResponseParser** can be used to create a **Response** object using the protocol described in [Section 4.2](#).

**Response** is a class that encapsulates the information returned by the server.

**ResponseParser** is an interface you can use to construct a custom response parser.

### *Package au.com.turnkey.toplevel.server.metadata*

**Descriptor** is a class that abstracts the request descriptor described in [Section 4.3](#). It contains a number of methods which allow you to access parts of the underlying XML document as properties. If you are creating the XML directly, you can use the constants defined by the class for element and attribute names.

### *Package au.com.turnkey.toplevel.server.client*

The classes in this package provide a slightly higher-level model of a request. If you are unfamiliar with java programming [Section Appendix B](#) contains a tutorial guide to using this class.

**ClientRequest** contains methods for constructing and sending a request. See the sample code for information on how it is used.

**ClientDescriptor** extends **Descriptor** to provide methods for creating and modifying a request descriptor.

**SectionData** is an interface you can use for constructing objects that can be used to supply data for a request section.

**SectionFile** implements **SectionData** for a file.

**SectionBytes** implements **SectionData** for an array of bytes.

## 7. Customising the Server

The server can be customised by adding your own java classes to override standard behavior.

To add your own java classes, package them in **.jar** files and place them in the **custom** subdirectory of the server directory. You can also add other JAR files, such as third-party libraries. Note that the JAR files are only read when the server starts, so you will need to restart the service if you have added or changed any files.

Tip:

You can add references to JAR files stored in other locations by changing the [wrapper.conf](#) file.

The [server configuration](#) determines the name of the class to use for particular objects. The `<classes>` element is used to specify class names.

The classes that are designed to be customised each have an interface and a default implementing class. The name of the default class is always the name of the interface with “Default” prepended. Thus the default implementation of the **RequestHandler** interface is the **DefaultRequestHandler** class.

The customisation interfaces are:

**RequestParser** This is used to read the stream of bytes from the client and create a **Request** object.

The request protocol implemented by **DefaultRequestParser** is very general and extensible, so it's rare that you will need to customise this class in order to implement a different protocol (unless the server needs to use an already established protocol).

A more common reason to customise this class is to allow the request to be altered in some way before it is processed. For example a custom RequestParser could be used to select the RequestHandler class to be used based on the Request properties.

**ResponseEncoder** This is used to send the response back to the client.

As for the RequestParser interface, the only time you are likely to customise this is when you must support an existing protocol.

**RequestHandler** This is used to control the processing of the request.

This is the most likely candidate for customisation. If, for example, you have added your own elements to the [request descriptor](#), then you will need to provide a customised RequestHandler to process them.

**QueuePolicy** This is used to control the queue or queues on which a request can run.

The default implementation allows the queue to be determined by matching the user name and/or request type properties of the request. Providing a customised implementation allows a more sophisticated selection method to be used.

**ResponseParser** This is not part of the server code; it is provided for use by client code when interpreting a response from the server. You probably only need to customise this if you have also customised ResponseEncoder.

Since this is not used by server code, it has no corresponding attribute in the `<classes>` configuration element.

## 7.1 An Example Server Customisation

The following is an example of a (not terribly useful) customised request handler. It modifies the request identifier by adding “X” to the beginning. Some examples of more realistic things a customised handler could be used for are:

- dynamically changing the request properties
- pre-processing the XML data before passing it to TopLeaf
- adding new methods of distributing the data produced by TopLeaf.

To create a custom RequestHandler, create a class that either implements the **RequestHandler** interface or extends **DefaultRequestHandler**. For this example we will do the latter:

```
package example.custom;

public class UselessRequestHandler
    extends DefaultRequestHandler
{
    public Response process(Request req)
    {
        // Make changes to the request
        req.setId("X"+req.getId());

        // Call the superclass to process it
        return super.process(req);
    }
}
```

Compile the class and arrange for it to be visible to the server. The simplest way of doing this is to put it in a **.jar** file and add this to the **custom** subdirectory of the server directory (usually **C:\TopLeafServer**).

Stop the service so **config.xml** can be safely changed.

Add the following to **config.xml**:

```
<classes request-handler="example.custom.UselessRequestHandler"/>
```

When the server is restarted it should use the new class when a request is processed. You can verify that your class is being used by setting the level of messages in the detail log to “FINEST”. This will produce messages that include the name of the classes used to instantiate the relevant objects.

## 7.2 Customising the Server Console

The server console can be customised by adding additional pages to display and/or maintain data for your application.

To add custom pages, create one or more classes that extend the **UserPage** class. These need to be registered by adding **<user-page>** elements to **<console>** as described in [Section 2.3](#).

Links to custom pages may appear in the navigation menu at the end of the **Configuration** section. Specify the string to appear in the menu by using the **menu** attribute in **<user-page>** or by overriding the **getMenu** method in your class.

The default for custom pages is to be “embedded” in the server console page framework. This means that a standard heading and navigation menu is automatically created, and the **sendContent** method only needs to send the HTML for the body of the page. You can override the **isEmbedded** method to disable this behaviour in certain cases (e.g. for returning an image).

## Appendix A Request Descriptor DTD

The following DTD is intended as a quick reference for the elements and attributes that may be used in a request descriptor document. Note that a descriptor does not need to be valid with respect to this DTD in order to work correctly — in general, the only requirement is that the correct element and attribute names are used.

Some of the names may relate to features that will be implemented in future releases of the software.

```

<!ELEMENT request (process, sections)>
<!ATTLIST request
  xmlns      CDATA #IMPLIED
  id         CDATA #IMPLIED
  type       CDATA #IMPLIED
  user       CDATA #IMPLIED
>

<!ELEMENT process (toplevel?, (output | print | email)*)>
<!ATTLIST process
  mode       (sync|async) #REQUIRED
  queue      CDATA #IMPLIED
>

<!ELEMENT topleaf (set*)>
<!ATTLIST topleaf
  repository      CDATA #IMPLIED
  publication      CDATA #IMPLIED
  partition        CDATA #IMPLIED
  compose-warning  (ignore|continue|fail) #IMPLIED
  compose-error    (continue|fail) #IMPLIED
  stylesheet       CDATA #IMPLIED
  publication-profile CDATA #IMPLIED
>

<!ELEMENT set EMPTY>
<!ATTLIST set
  var      CDATA #REQUIRED
  string   CDATA #REQUIRED
>

<!ELEMENT output (email*)>
<!ATTLIST output
  response      (yes|no) "yes"
  format        (pdf|html|rtf|text) #IMPLIED
  content       (compose|log) "compose"
  profile       CDATA #IMPLIED
  condition     (always|ifwarning|iferror) "always"
  fail-content  (none|log) "log"
  dest-path     CDATA #IMPLIED
  dest-dir      CDATA #IMPLIED
  dest-name     CDATA #IMPLIED
>

<!ELEMENT print EMPTY>
<!ATTLIST print

```

```
    printer    CDATA    #REQUIRED
>

<!ELEMENT email ((to|cc|bcc)*, body?)>
<!ATTLIST email
    to        CDATA    #REQUIRED
    cc        CDATA    #IMPLIED
    bcc       CDATA    #IMPLIED
    from      CDATA    #IMPLIED
    reply-to  CDATA    #IMPLIED
    subject   CDATA    #IMPLIED
    id        ID       #IMPLIED
    ref       IDREF    #IMPLIED
>

<!ELEMENT to (#PCDATA)>

<!ELEMENT cc (#PCDATA)>

<!ELEMENT bcc (#PCDATA)>

<!ELEMENT body (#PCDATA)>
<!ATTLIST body
    format    (text|html) "text"
>

<!ELEMENT sections (section+)>

<!ELEMENT section EMPTY>
<!ATTLIST section
    use        CDATA    #IMPLIED
    path       CDATA    #IMPLIED
    format     CDATA    #IMPLIED
    archive-root CDATA    #IMPLIED
>
```

## Appendix B Client Tutorial

This is a tutorial guide to writing a client program using the **ClientRequest** class.

**Warning:**

**This tutorial is for informative purposes only, and may not correspond to the latest version of the software.**

**Consult the API reference documentation if you are in any doubt.**

### Creating a ClientRequest object

The first step in executing a TopLeaf Server request is creating the **ClientRequest** object. This is done as follows:

```
ClientRequest req = new ClientRequest();
```

In most cases you will need to add some information about the request. The following section shows how to set this information.

### Accessing the request descriptor

A request can contain a **descriptor** in the form of an XML document as described in [Section 4.3](#).

You can access the descriptor using the following method:

```
ClientDescriptor desc = req.getDescriptor();
```

The **ClientDescriptor** class contains an XML document which can be modified using DOM classes and methods. It also provides methods for accessing some of this information as properties.

For example, the following sets the identifier for a request:

```
Element reqElt = desc.getRequestElement();  
reqElt.setAttribute(Descriptor.ATT_REQUEST_ID, "REQ001");
```

The following has the same effect:

```
desc.setIdentifier("REQ001");
```

The **addCustomElement** method can be used to add arbitrary elements to the descriptor. This is probably only useful if you have [customised](#) the request handler to use them. The following adds an account element to the descriptor and sets some attributes on it:

```
Element act = req.addCustomElement("account");  
act.setAttribute("user", "fred");  
act.setAttribute("dept", "sales");
```

Adding output and print elements

A request can contain any number of `<output>` and `<print>` elements to determine what is produced. These can be added like this:

```
Element outElt = desc.addOutputElement();  
Element prtElt = desc.addPrintElement();
```

These calls can be made any number of times, creating a new element each time.

Attributes can be added to these elements; for example:

```
outElt.setAttribute(ATT_OUTPUT_FORMAT, "rtf");
```

## ***Adding data sections to the request***

Having set up your descriptor, the next step is to add one or more data sections to the request. These sections contain source material (eg. XML, graphics, ZIP archives) plus any other content (eg. schemas, scripts) needed for the request.

A new section is added with the following call:

```
Element secElt = req.addSection(data);
```

where *data* is an object implementing the **SectionData** interface. The **SectionFile** and **SectionBytes** classes can be used to construct suitable objects for files and byte arrays, respectively.

Either call returns a `<section>` element which is automatically added to the descriptor and can have attributes set in the normal manner. For example:

```
secElt.setAttribute(ATT_SECTION_USE, "root");
```

## ***Sending the request***

The final step is to send your request to the server and await a response:

```
Response resp = req.send(SERVER, PORT);
```

where:

- **SERVER** is the address of the server machine.
- **PORT** is the port number on which the server is listening for requests.

The returned **Response** object can be interrogated to determine the status of the request and any files that were returned.

Note:

Data contained in the response will only be copied to files if you call the **setTempDir** method on the request object.

## Appendix C Acknowledgements and Licences

Portions of TopLeaf Server reference software that is used under the following terms:  
Java Service Wrapper

Website: <http://wrapper.tanukisoftware.org>.

Copyright (c) 1999, 2006 Tanuki Software, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of the Java Service Wrapper and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sub-license, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Portions of the Software have been derived from source code developed by Silver Egg Technology under the following license:

BEGIN Silver Egg Technology License -----  
-

Copyright (c) 2001 Silver Egg Technology

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sub-license, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,

WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

END Silver Egg Technology License -----  
-

Xerces XML parser

Website: <http://xerces.apache.org>.

Apache License  
Version 2.0, January 2004  
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by

a

copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of,

publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

- (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
- (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
- (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
- (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one

of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or

implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[ ]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on

the

same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License. Pygmy Web Server

Web site: <http://pygmy-httpd.sourceforge.net>.

The Artistic License

Preamble

The intent of this document is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the Package in a more-or-less customary fashion, plus the right to make reasonable modifications.

Definitions:

"Package" refers to the collection of files distributed by the Copyright Holder, and derivatives of that collection of files created through textual modification.

"Standard Version" refers to such a Package if it has not been modified, or has been modified in accordance with the wishes of the Copyright Holder.

"Copyright Holder" is whoever is named in the copyright or copyrights for the package.

"You" is you, if you're thinking about copying or distributing this Package.

"Reasonable copying fee" is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)

"Freely Available" means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that recipients of the item may redistribute it under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.

2. You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.

3. You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:

a) place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as ftp.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.

b) use the modified Package only within your corporation or organization.

c) rename any non-standard executables so the names do not conflict with standard executables, which must also be provided, and provide a separate manual page for each non-standard executable that clearly documents how it differs from the Standard Version.

d) make other distribution arrangements with the Copyright Holder.

4. You may distribute the programs of this Package in object code or

executable form, provided that you do at least ONE of the following:

- a) distribute a Standard Version of the executables and library files, together with instructions (in the manual page or equivalent) on where to get the Standard Version.
- b) accompany the distribution with the machine-readable source of the Package with your modifications.
- c) accompany any non-standard executables with their corresponding Standard Version executables, giving the non-standard executables non-standard names, and clearly documenting the differences in manual pages (or equivalent), together with instructions on where to get the Standard Version.
- d) make other distribution arrangements with the Copyright Holder.

5. You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution provided that you do not advertise this Package as a product of your own.

6. The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whomever generated them, and may be sold commercially, and may be aggregated with this Package.

7. C or Java subroutines supplied by you and linked into this Package shall not be considered part of this Package.

8. The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.

9. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The End

The version of pygmy included in TopLeaf Server is a very slightly modified version of the standard library (available from SourceForge). The modification consists of a single bug fix to prevent

IOExceptions after a client closes a socket that was kept open by the server to honour HTTP keep-alive headers.

The details can be found on the pygmy bug tracker at:

[http://sourceforge.net/tracker/index.php?func=detail&aid=1667604&group\\_id=87807&atid=584457](http://sourceforge.net/tracker/index.php?func=detail&aid=1667604&group_id=87807&atid=584457).