HELIX SECURITY MANAGER DEPLOYMENT GUIDE
Version 11
Revision Date: May 8, 2006 2:36 pm
# CONTENTS

## INTRODUCTION

1. Conventions Used in This Book ................................................................. 1
2. Technical Support .................................................................................... 2

## 1 SECURITY MANAGER OVERVIEW

1. Security Manager Components ................................................................. 3
2. Web Portal ................................................................................................. 3
3. URL Processor ......................................................................................... 3
4. Allowance Plug-ins .................................................................................. 4
5. Licensing ................................................................................................. 5
6. Secure URLs ............................................................................................ 5
7. URL Timeout ............................................................................................. 6
8. Security Key ............................................................................................. 6
9. Tokens ...................................................................................................... 6
10. Optional Security System Features ......................................................... 7
11. IP Address Verification ............................................................................ 7
12. Media Streaming Lifetime ....................................................................... 7
13. External Reallowance ............................................................................. 7
14. Allowance Bypass .................................................................................. 8

## 2 INSTALLING URL PROCESSOR

1. Setting Up the Java Environment .............................................................. 9
2. Installing URL Processor Files ................................................................. 10
3. Setting up Log4j for URL Processor ......................................................... 11
4. Configuring URL Processor ...................................................................... 12
5. Defining the License File Location ........................................................... 13
6. Setting Basic URL Processor Values ....................................................... 13
7. Abbreviating Query String Parameter Names ......................................... 16
8. Replacing URL Keywords ...................................................................... 18
9. Defining URL Processor Tokens .............................................................. 19
10. Starting and Stopping URL Processor .................................................... 21
11. Starting JBoss and URL Processor .......................................................... 21
12. Stopping URL Processor ....................................................................... 21
13. Changing the JBoss Port ....................................................................... 22
3 SETTING UP HELIX SERVERS AND HELIX PROXIES

Installing the Allowance Plug-in ................................................................. 23
Installing and Verifying the Allowance Plug-in License ................................. 23
Installing the Allowance Plug-in ................................................................. 24
Creating a Token File for the Allowance Plug-In ........................................... 24
Configuring the Streaming Server Allowance Plug-in ..................................... 25
Testing the Allowance Plug-in ..................................................................... 26
Setting Basic Allowance Plug-in Parameters ............................................... 27
Enabling the Allowance Plug-in .................................................................... 28
Defining Token Handling ............................................................................ 28
Configuring Allowance Logs ........................................................................ 29
Enabling Error Handling ............................................................................. 31
Setting Allowance Bypass Operation ............................................................ 31
Selecting the IP Address Source .................................................................. 31
Defining URL Parameter Keywords ............................................................ 32

4 SETTING UP DOWNLOAD SERVERS AND PROXIES

Installing the Download Server Allowance Plug-in ................................-------- 33
Installing the Download Server Allowance Plug-in ......................................... 33
Configuring the Download Server Allowance Plug-in .................................... 34
Creating a Token File for the Download Server ............................................. 38

5 CONFIGURING ADVANCED AUTHORIZATION FEATURES

Error Handling for Allowance Plug-ins ......................................................... 41
Error Types and Default Messages .............................................................. 41
Defining Error Handlers ............................................................................. 42
Creating Error Handlers on Helix Server or Helix Proxy ................................. 42
Creating a Customized Error Message ......................................................... 43
Redirecting a URL on an Error ..................................................................... 44
Contacting an External Allowance Server .................................................... 44
Defining External Allowance Error Handling for Streaming Servers ............. 45
Setting Up External Allowance Error Handling for Download Servers .......... 45
Configuring Error Web Service Attribute Values ........................................... 46
Posting Information to the External Allowance Server ................................. 46
Responding to a Reallowance Request ......................................................... 48
Response Code 1 – Use Default Error .......................................................... 48
Response Code 2 – Allow Access ................................................................ 49
Response Code 3 – Custom Error Message .................................................. 49
Response Code 4 – RTSP Redirect ............................................................... 49
Modifying Parameters on URL Redirection .................................................. 51
Resubmitting General Parameter Values ...................................................... 51
Setting Up Allowance Bypass ..................................................................... 53
INTRODUCTION

Welcome to the Helix Security Manager Deployment Guide from RealNetworks®. This document explains the installation and configuration of Helix Security Manager, a set of software components that provides security for streaming and downloaded media.

Conventions Used in This Book

The following table explains the typographic conventions used in this book.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>emphasis</strong></td>
<td>Bold text is used for in-line headings, user-interface elements, URLs, and e-mail addresses.</td>
</tr>
<tr>
<td><strong>terminology</strong></td>
<td>Italic text is used for technical terms being introduced in a given manual or other document, and to lend emphasis to generic English words or phrases.</td>
</tr>
<tr>
<td><strong>syntax</strong></td>
<td>This font is used for file names, directory names, code examples (excerpted or in whole), or command-line instructions.</td>
</tr>
<tr>
<td><strong>syntax emphasis</strong></td>
<td>Bold syntax character formatting is used for program names and to emphasize specific syntax elements.</td>
</tr>
<tr>
<td><strong>variables</strong></td>
<td>Italic text denotes variables. Substitute values appropriate for your system.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Square brackets indicate optional values. As a rule, when you use these optional values, you do not include the brackets themselves.</td>
</tr>
<tr>
<td><strong>choice 1</strong></td>
<td>Vertical pipes separate values you can choose between.</td>
</tr>
<tr>
<td><strong>choice 2</strong></td>
<td>Vertical pipes separate values you can choose between.</td>
</tr>
</tbody>
</table>

(Table Page 1 of 2)
Technical Support

To reach RealNetworks Technical Support, please fill out the form at http://customerrelations.real.com/scripts/rnforms/contact_tech_service.asp

The information you provide in this form will help Technical Support personnel respond promptly. For general information about RealNetworks Technical Support, visit this Web page:

http://service.real.com/help/call.html
CHAPTER 1

SECURITY MANAGER OVERVIEW

This chapter provides an overview of Helix Security Manager. It describes the system components and explains how the system creates and authenticates secure URLs. It also covers some of the advanced features that you can implement using Helix Security Manager.

Security Manager Components

The Helix Security Manager allows you to generate tamper-proof URLs that grant media access to authorized users. This security system consists of three main components: a Web portal, the URL Processor, and an allowance plug-in for each media server. You configure Security Manager components using XML-based configuration files. The system does not include graphical user interfaces. Rather, it provides interfaces that allow you to integrate Security Manager features into an existing Web portal using Java methods or CGI scripts.

Web Portal

Helix Security Manager integrates with any Web portal running on any operating system and Web server software. The portal verifies the identity of each user, determining if the user should receive access to the requested content. For allowed accesses, it transmits the unsecure URL path to URL Processor, receiving back a secure URL that it forwards to the user.

URL Processor

The URL Processor application runs under the JBoss application server, which is included with Helix Security Manager. It implements security features by attaching query string parameters to the URL submitted by the Web portal.
Secure URLs are invalidated if they are altered, and have a timeout feature that guards against URL forwarding and bookmarking.

**URL Processor Requirements**

The following table lists minimum requirements for the machine running the JBoss application server along with URL Processor.

<table>
<thead>
<tr>
<th>Component</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer</td>
<td>Sun v120</td>
</tr>
<tr>
<td></td>
<td>Intel or compatible 2.0GHz or greater</td>
</tr>
<tr>
<td>Operating Systems</td>
<td>Solaris 8, Solaris 9 (SPARC only)</td>
</tr>
<tr>
<td></td>
<td>RedHat Enterprise Linux 4 AS or ES</td>
</tr>
<tr>
<td></td>
<td>Windows Server 2003</td>
</tr>
<tr>
<td>RAM</td>
<td>1 GB</td>
</tr>
<tr>
<td>Disk Storage</td>
<td>100 GB (logging disk)</td>
</tr>
<tr>
<td>Network Interface Card</td>
<td>100 Mbit/sec (minimum)</td>
</tr>
<tr>
<td>Java Environment (J2SE)</td>
<td>Sun Java 1.4.2 available at:</td>
</tr>
<tr>
<td></td>
<td><a href="http://java.sun.com/j2se/1.4.2/download.html">http://java.sun.com/j2se/1.4.2/download.html</a></td>
</tr>
<tr>
<td>Application Server</td>
<td>JBoss Application Server 4.0.22 (included with Helix Security Manager)</td>
</tr>
</tbody>
</table>

**Allowance Plug-ins**

Each Helix Server, Helix Proxy, download server, or download proxy that hosts your content uses an allowance plug-in. This plug-in evaluates the security parameters of each URL request, allowing the access only if the URL has not been tampered with and has not timed out. The plug-in keeps a log file of the actions it took on each URL request.

**For More Information:** Chapter 3 and Chapter 4 provide allowance plug-in configuration instructions for streaming and download servers, respectively. See also Chapter 5 for information about advanced allowance features.
Allowance Plug-in Requirements

Allowance plug-ins are installed on your existing streaming and download servers. The following table lists server software versions compatible with Helix Security Manager allowance plug-ins.

<table>
<thead>
<tr>
<th>Server</th>
<th>Supported Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Streaming Server</td>
<td>Helix Server 11 or later</td>
</tr>
<tr>
<td>Streaming Proxy</td>
<td>Helix Proxy 11 or later</td>
</tr>
<tr>
<td>Download Server</td>
<td>Apache v2.0 (earlier and later versions not supported); Covalent Enterprise Ready Server</td>
</tr>
<tr>
<td>Download Proxy</td>
<td>Apache v2.0 (earlier and later versions not supported); Covalent Enterprise Ready Server</td>
</tr>
</tbody>
</table>

Licensing

A valid license file is required to activate URL Processor and the allowance plug-ins. The license uses the file extension of .lic. The default license filename is secmgr-license.lic, and is used for all Helix Security Manager components. The license file is not included with the program files. It is available separately from your RealNetworks representative.

Secure URLs

URL Processor secures URLs by means of query string parameters. The following example illustrates the essential security parameters in a URL for streaming media (line breaks have been added for clarity):

http://helixserver.example.com/ramgen/video.rm
?token ttl=120
&gt ime=1052261021
&tokenname=key5
&key=D832AC16CC54615E313723538AF6F278

For More Information: Chapter 6 explains how to use all of the URL Processor security features, options, and overrides.
URL Timeout

Secure URLs include a lifetime such as 120 seconds. If a user accesses the URL after this lifetime expires, the server’s allowance plug-in denies the access attempt. The `tokenttl` value sets the URL lifetime in seconds. (Allowance plug-ins are also configured with a default lifetime value in case `tokenttl` is not present.) The `gt` value indicates the time when URL Processor generated the URL. The allowance plug-in evaluating the access uses the `gt` value to determine the start of the lifetime.

**Note:** To ensure that URL timeout works as expected, the URL Processor machine and all content servers should use a time synchronizing program.

Security Key

The `key` parameter secures the URL against tampering. URL processor generates the `key` value by hashing a private token value with *everything* in the URL that precedes the `key` parameter. When evaluating the secure URL, an allowance plug-in regenerates the `key` value from the requested URL and the same token value. If the regenerated `key` value does not match the `key` value in the requested URL exactly, the allowance plug-in rejects the request.

**Note:** In generating the key value, URL Processor can use the MD5 or SHA-1 hash algorithm.

Tokens

URL Processor generates the security hash key using a token, which consists of a user-defined name, such as `key5`, along with a user-defined value similar to a password, such as `rt238yg9`. The security system can use a single, default token value defined in the configuration files for URL Processor and each allowance plug-in. In this case, a token name is not needed and the `tokenname` parameter is omitted from the secure URL.

 Optionally, you can define a set of tokens for URL Processor and each plug-in. When generating the secure URL, the Web portal instructs URL Processor to use one of the token values. URL Processor then includes the token name as the value of the `tokenname` parameter. When it evaluates an access request, an allowance plug-in looks up the value associated with the token name and uses the associated value to regenerate the secure URL.
Optional Security System Features

The following sections describe optional features that you can use with Helix Security Manager.

**IP Address Verification**

URL Processor can include the client’s IPv4 address (with or without a netmask) as a value when generating the URL’s secure hash key. Allowance plug-ins then verify that the media player requesting the media from a streaming or download server uses the same IP address as the browser client that requested the media URL from the Web portal.

*For More Information:* Refer to “Verifying Client IP Addresses” on page 68 for details.

**Media Streaming Lifetime**

URL Processor can set a time limit for viewing streaming media. It might set a three-minute total viewing time for a clip, for instance. This allows you to stream a preview or implement a pay-per-view service without encoding separate clips. Once the viewer’s time limit has expired, the allowance plug-in terminates access to the stream.

*For More Information:* See the section “Setting a Clip Lifetime” on page 65.

**External Reallowance**

Helix Security Manager allows you to configure the actions that occur on the URL timeout and stream timeout conditions. You can, for example, have the allowance plug-in forward the URL to an *external allowance server*. This is any Web-based portal that has access to the user information. The portal can then instruct the allowance plug-in to allow access, deny access, or redirect the user to different content.

*For More Information:* For information about allowance error conditions and external reallowance, see “Error Handling for Allowance Plug-ins” on page 41.
Allowance Bypass

In most content delivery systems, access to the content is secured. However, you may also want to host some unsecured content for general access. Free content might include movie trailers or video introductions to how your subscription system works. Helix Security Manager allows you to define which content is available without security restrictions. You can do this on a general level, such as any content in a specific directory, or with fine granularity, such as URL-by-URL.

For More Information: For an explanation of using this feature, refer to “Setting Up Allowance Bypass” on page 53.
This chapter explains how to install and configure the URL Processor Component of Security Manager. The URL Processor runs as an application under the JBoss application server.

**For More Information:** Chapter 6 explains URL Processor features.

## Setting Up the Java Environment

The JBoss application server requires a Java environment. You can install a J2SE Runtime Environment (JRE) or a full Java Development Kit (JDK). Security Manager supports version 1.4.2 or the JRE or JDK from Sun Microsystems.

**Note:** Other Java environments may work with Security Manager, but are not supported by RealNetworks.

- **To install a Java Environment:**
  1. Download the Sun JRE or JDK from the following Web page:
     
     [http://java.sun.com/j2se/1.4.2/download.html](http://java.sun.com/j2se/1.4.2/download.html)

     2. Install the JRE or JDK files.
     
     a. On Windows, double-click the installation file to launch the installer. Follow the installer prompts to perform a typical installation.

     b. On UNIX, run the self-extracting binary file (.bin), or run the RPM file as the root user.

     **For More Information:** Detailed installation instructions for the JRE are available at [http://java.sun.com/j2se/1.4.2/jre/install.html](http://java.sun.com/j2se/1.4.2/jre/install.html). For JDK installation instructions, refer to [http://java.sun.com/j2se/1.4.2/install.html](http://java.sun.com/j2se/1.4.2/install.html).
3. Add the JAVA_HOME environmental variable to your environment.
   a. On UNIX, edit the shell profile, such as .bash_profile, of the user running the Java environment. Include the following line:

   ```
   JAVA_HOME=Java_location
   export JAVA_HOME
   
   For example:
   ```
   ```
   JAVA_HOME=/usr/local/java/j2sdk1.4.2_11
   export JAVA_HOME
   ```
   b. On Windows, set the JAVA_HOME variable to the directory where you installed the Java Environment. You add variables using the command

   ```
   Start>Settings>Control Panel>System>Advanced>Environment Variables. For example:
   ```

   ```
   Variable name: JAVA_HOME
   Variable value: C:\Program Files\Java\j2re1.4.2_11
   ```

### Installing URL Processor Files

The following procedure explains how to install JBoss and Security Manager files. The installation procedure is similar on Windows and UNIX computers. All files are contained within a compressed archive. Security Manager does not use an installation application.

**Note:** On Windows, you need an application capable of reading Tar archives that have been compressed using Gzip (.tar.gz). Many Zip utilities on Windows can read these formats.

- **To install URL Processor:**
  
  1. Uncompress the Security Manager URL Processor package until you reach the jboss-version folder.
  
  2. Move the jboss-version folder to the location where you want to run the URL Processor. For example:

   ```
   UNIX: /usr/local/jboss-4.0.2
   Windows: C:\jboss-4.0.2
   ```

  3. Add the JBOSS_HOME environmental variable to your environment.
a. On UNIX, edit the shell profile, such as .bash_profile, of the user running URL Processor. Include the following line:

```
JBOSS_HOME=JBOSS_location
export JBOSS_HOME
```

For example:

```
JBOSS_HOME=/usr/local/jboss-4.0.2
export JBOSS_HOME
```

b. On Windows, set the JBOSS_HOME variable to the directory where you installed JBoss. You add variables using the command

Start>Settings>Control Panel>System>Advanced>Environment Variables. For example:

```
Variable name: JBOSS_HOME
Variable value: C:\jboss-4.0.2
```

4. Copy the Security Manager license file, typically named secmgr-license.lic, to the secmgr subdirectory under the JBoss installation directory.

Setting up Log4j for URL Processor

You can set up Log4j logging to capture information useful for monitoring URL Processor performance and troubleshooting problems. URL Processor uses the Log4j Java API, an open source logging API for Java applications that enables monitoring and debugging. This logging information is entirely separate from the authorization, access, and error logs created by the Security Manager security system.

**For More Information:** For background on Log4j, refer to [http://logging.apache.org/log4j/docs/index.html](http://logging.apache.org/log4j/docs/index.html).

To use Log4j with URL Processor:

1. Using any text editor, open the file secmgr-log4j.properties located in the secmgr subdirectory under the JBoss installation directory.

2. In the file secmgr-log4j.properties, change every sample path to the path relevant to your environment.

3. Save and close the file secmgr-log4j.properties.

**Note:** The script used to start JBoss must refer to the location of the Log4j properties file. The scripts secmgrstart.sh and
Helix Security Manager Deployment Guide

secmgrstart.bat, which the section “Starting JBoss and URL Processor” on page 21 describes, are preconfigured to point to the default location of the properties file.

Configuring URL Processor

To configure URL Processor, you use any text editor to edit the file secmgr-config.xml located in the secmgr subdirectory under the JBoss installation directory. The values in this file allow you to define basic URL Processor operation:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<SDSConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
<License LicenseFileName="../secmgr/secmgr-license.lic" />

<URLProcessor UseDefaultToken="true" UseFullURL="false" UseBase64="false"
MACAddress="address" DefaultToken="default_token_value"
EncryptionType="0">
<URLKeywords>
...Keywords for query string parameters passed to URL Processor...
</URLKeywords>
<ReplaceKeywordsInURL>
...Keywords to replace in a requested URL for authorizing the access...
</ReplaceKeywordsInURL>
<Keys>
<Key Name="tokenname1" Value="tokenvalue1" />
<Key Name="tokenname2" Value="tokenvalue2" />
...additional name and value pairs...
</Keys>
<URLProcessorEncryptionType="0|1"/>
</URLProcessor>
</SDSConfig>

Tip: The majority of default values are adequate for most installations. Minimally, verify the license location and set the appropriate values for MACAddress and DefaultToken.

Note: If you want to do an advanced configuration of URL Processor, refer to the schema file secmgr-config.xsd.
Defining the License File Location

The attribute LicenseFileName provides URL Processor with the location of the Security Manager license file. Verify that the value is the correct relative or absolute path, including the file name. For example:

```
<License LicenseFileName="..\secmgr\secmgr-license.lic" />
```

Setting Basic URL Processor Values

The `<URLProcessor>` tag contains the basic attributes that you use to configure the URL Processor:

```
<URLProcessor UseDefaultToken="true\|false" UseFullURL="true\|false"
    UseBase64="true\|false" MACAddress="address" DefaultToken="value"
    EncryptionType="0|1"
    RequiredURLAttributes="parameter1,parameter2,..."/>

...Other URL Processor configuration variables...
</URLProcessor>
```

UseDefaultToken and DefaultToken

URL Processor uses the default token value to generate a URL’s secure hash key if no token name is included as an argument in the method used to submit the URL for processing. For UseDefaultToken, set a true or false value that specifies whether or not to use the default token. If you turn off the default token with a value of false, URLs that do not include a token name value are rejected by URL Processor. If you set UseDefaultToken to true, set DefaultToken to any value.

**Tip:** To override the default token, you encode secure URLs using a token name and value pair selected from a token file, which is described in the section “Defining URL Processor Tokens” on page 19. See “tokenname” on page 74 for information on how to specify which token URL Processor uses.

**For More Information:** The section “Defining Token Handling” on page 28 explains the DefaultToken variable for allowance plug-ins. The value for each plug-in must match the value you set for the URL Processor.
UseFullURL

This property takes a true or false value that indicates whether or not to use the full URL for creating the hash. The false value is recommended. In this case, URL Processor ignores the URL keywords defined in the <ReplaceKeywordsInURL> list when creating the secure hash. For more information about this list, refer to “Replacing URL Keywords” on page 18.

UseBase64

Use a true value for the UseBase64 parameter to turn Base64 encoding on for the values of the URL parameters key and guid. A false value causes URL Processor to use Hex encoding for these values. Base64 encoding shortens the length of the key and guid parameters by 1/4 when used with the default MD5 algorithm. For example, the 16-character key and guid values are 32 characters when Hex-encoded versus 24 characters when Base64 encoded.

Base64 encoding also works with the optional SHA-1 algorithm, which you can choose between as described in the section “Overriding the Default Encryption” on page 67. When using the SHA-1 algorithm, the key and guid values are 20 alphanumeric characters instead of 16. With Base64 encoding, the transmitted values are approximately 2/3 the size of the Hex-encoded values.

Tip: When verifying a URL, allowance plug-ins automatically determine if Base64 encoding was used based on the length of the secure hash key value. Allowance plug-ins therefore require no configuration to use this feature.

For More Information: Base64 encoding in the URL Processor adheres to the standards defined in RFC 3548. Specifically, only characters from the Base 64 alphabet are included in the Base64-encoded parameters. For more information, refer to the RFC at http://www.faqs.org/rfcs/rfc3548.html.

MACAddress

This required attribute provides the network adapter’s Media Access Control (MAC) address associated with the server hosting the URL Processor. URL Processor uses this address to ensure that the GUID associated with a secure URL is globally unique. Do not include dashes or colons in the MAC address. For example:

MACAddress="00342398EFAC"
**Tip:** On UNIX machines, the root user can give the command `ifconfig -a` to report the MAC address, which is typically listed as the hardware address under the eth0 device. On Windows, enter `getmac` at the command prompt.

**Default Encryption Algorithm**

The `EncryptionType` attribute defines the default encryption algorithm used to generate the secure URL’s hash key:

`EncryptionType="0|1"`

The attribute uses a value of 0 or 1:

- 0 MD5 algorithm
- 1 SHA-1 algorithm

The SHA-1 algorithm is more secure than MD5, but takes up to 50% more time to run. You will therefore affect the performance of URL Processor if you use SHA-1 for all URLs. The secure key generated with SHA-1 is longer than that generated with MD5 (20 versus 16 characters), but you can shorten the transmitted value using Base64 encoding rather than Hex encoding, as described in the section “UseBase64” on page 14.

**For More Information:** You can override the default algorithm on a case-by-case basis, as described in the section “Overriding the Default Encryption” on page 67.

**RequiredURLAttributes**

The optional `RequiredURLAttributes` attribute lists query string parameters that must be present in URLs submitted to URL Processor. Because URL Processor includes these parameters when generating the secure hash key, removing the parameters from the secure URL invalidates the URL.

To use the `RequiredURLAttributes` attribute, add it to the `URLProcessor` tag and indicate the required parameters in a comma-delimited list, as demonstrated in the following example:

```
<URLProcessor ...other attributes... RequiredURLAttributes="userid,subplan">
```

With the preceding configuration example, URL Processor rejects URLs submitted to it if they do not contain the `userid` and `subplan` query parameters:

```
http://helixserver.example.com/ramgen/video.rm?userid=value&subplan=value
```
Tip: It is not necessary to define the query parameters your system uses as values of RequiredURLAttributes. URL Processor accepts all parameters in URLs submitted to it. Use this feature only if you want to ensure that all secure URLs always contain specific parameters that invalidate the URL if removed.

Abbreviating Query String Parameter Names

By using URL abbreviations, you can shorten parameter names contained in URL query strings. For example, you can abbreviate `tokenname=value` in a query string to `t=value`. This is useful for media players or devices that cannot accept long query string parameters. The URLKeywords list defines the keyword abbreviations to use. For example:

```xml
<URLProcessor ...>
    ...
    <URLKeywords>
        <DigestAlgorithmKeyword>da</DigestAlgorithmKeyword>
        <GTimeKeyword>gtime</GTimeKeyword>
        <IPAddressKeyword>ip</IPAddressKeyword>
        <HashKeyKeyword>key</HashKeyKeyword>
        <LifeKeyword>life</LifeKeyword>
        <NetmaskKeyword>nm</NetmaskKeyword>
        <TokenNameKeyword>t</TokenNameKeyword>
        <TokenTimeoutKeyword>ttl</TokenTimeoutKeyword>
    </URLKeywords>
    ...
</URLProcessor>
```

The following table indicates the URL parameters that you can abbreviate. To shorten the token name parameter, for example, define an abbreviation, such as “t,” for the `<TokenNameKeyword>` entry in the URLKeywords list. Abbreviations in the following table are provided as examples only. If you already use a query
string parameter named “t,” for instance, you can specify a different abbreviation for the tokenname parameter.

### Configurable Keywords for Secure URL Query Parameters

<table>
<thead>
<tr>
<th>Default URL Keyword</th>
<th>Suggested Value</th>
<th>Configuration Variable</th>
<th>Parameter Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>gtime</td>
<td>g</td>
<td>GTimeKeyword</td>
<td>Indicates the time the URL was generated. Used with the URL Processor, streaming servers, and download servers. See “Generation Time” on page 65.</td>
</tr>
<tr>
<td>ip</td>
<td>ip</td>
<td>IPAddressKeyword</td>
<td>Causes the allowance plug-in to use IP address validation for the access. Used with the URL Processor, streaming servers, and download servers. See “Verifying Client IP Addresses” on page 68.</td>
</tr>
<tr>
<td>key</td>
<td>key</td>
<td>HashKeyKeyword</td>
<td>Provides the secure hash key. Used with the URL Processor, streaming servers, and download servers. See “Security Key” on page 64.</td>
</tr>
<tr>
<td>life</td>
<td>l</td>
<td>LifeKeyword</td>
<td>Sets the lifetime for a stream. Not used with download servers. See “Setting a Clip Lifetime” on page 65.</td>
</tr>
<tr>
<td>nm</td>
<td>nm</td>
<td>NetmaskKeyword</td>
<td>Applies a netmask to the IP address value. Used with the URL Processor, streaming servers, and download servers.</td>
</tr>
</tbody>
</table>

(Table Page 1 of 2)
Configurable Keywords for Secure URL Query Parameters (continued)

<table>
<thead>
<tr>
<th>Default URL Keyword</th>
<th>Suggested Value</th>
<th>Configuration Variable</th>
<th>Parameter Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>tokenname</td>
<td>t</td>
<td>TokenNameKeyword</td>
<td>Indicates the token to use to recreate the secure hash key. Used with the URL Processor, streaming servers, and download servers. See “tokenname” on page 74.</td>
</tr>
<tr>
<td>tokenttl</td>
<td>ttl</td>
<td>TokenTimeoutKeyword</td>
<td>Sets the life of the URL. Not used with the URL Processor. See “Specifying the URL Timeout” on page 67.</td>
</tr>
</tbody>
</table>

The configuration file for each allowance plug-in must also define the keyword abbreviations. When it evaluates a secure URL, an allowance plug-in first looks for an abbreviated keyword. If it does not find this, it matches URL values to default parameter names. As noted in the preceding table, however, some plug-ins do not need to define certain keywords. Because the lifetime parameter is not relevant for a download server, for example, you do not need to include this abbreviation in a download server’s allowance plug-in file.

**Tip:** Do not use lt or gt as keyword abbreviations. Microsoft Internet Explorer incorrectly interprets the strings &lt or &gt as less-than and greater-than signs (< and >), respectively, even though the strings lack a semi-colon at the end.

**For More Information:** For information about configuring the allowance plug-in, refer to “Defining URL Parameter Keywords” on page 32.

**Replacing URL Keywords**

Using the ReplaceKeywordsInURL list, you can indicate values that the URL Processor removes from the submitted URL before it generates the secure URL’s hash key. You should remove any value that will not appear in the secure URL used to request media from servers. Place each value between <Keyword> and </Keyword> tags, as shown here:
<URLProcessor ...”>
    ...
<ReplaceKeywordsInURL>
    <Keyword>/sdpgen</Keyword>
    <Keyword>/ramgen</Keyword>
    <Keyword>/asxgen</Keyword>
    ...
</ReplaceKeywordsInURL>

Note: To remove the keywords from hash key generation, you must also set the UseFullURL parameter to false, as described in the section “Setting Basic URL Processor Values” on page 13.

Client Launch Utility Keywords

RealNetworks recommends that you retain the keywords /sdpgen, /ramgen, and /asxgen in the list. These keywords are for utilities that launch media players using a Web page URL. The mount points are not present in the URLs that the media players use to request media from Helix Server, however. For example, a Web page link that launches RealPlayer may look like this:
http://helixserver.example.com/ramgen/video.rm

The /ramgen mount point causes Helix Server to send the user’s browser a MIME string that launches RealPlayer. In turn, RealPlayer issues a second request for the media in the form of a URL that uses RTSP and omits the /ramgen mount point:
rtsp://helixserver.example.com/video.rm

If you include /ramgen when creating the hash key for the secure URL, the Helix Server allowance plug-in rejects the request because the second, RTSP URL it receives does not include /ramgen.

For More Information: For additional information about any of the media player launch utilities (Ramgen, SDPgen, or ASXgen), refer to the chapter on clip delivery in Helix Server Administration Guide.

Defining URL Processor Tokens

The optional keys list defines token name and value pairs that URL Processor uses to generate the secure URL. The token name identifies the value that
URL Processor uses to compute the secure hash key. The token name is passed to URL Processor by the method that generates the secure URL, as explained in the section “tokenname” on page 74. The token name is also included in the URL to inform the allowance plug-in which value to use to verify the secure hash key. You can create any number of token name and value pairs in the following format:

```xml
<URLProcessor ...
  ...
</URLProcessor>
```

For example, you might define a token and value such as the following:

```xml
<Key Name="token1" Value="ert349HZK"/>
```

Note the following about token names and values:

- If you do not include a token name as an argument in the method used to generate the secure URL, URL Processor generates the hash key using the value of its DefaultToken parameter. For more information, refer to “UseDefaultToken and DefaultToken” on page 13.
- Each allowance plug-in must have a list of tokens that is the same as, or a subset of, the token list used by URL Processor. For more information, refer to “Creating a Token File for the Allowance Plug-In” on page 24.
- Token names and values are case-sensitive. The names and values used by URL Processor and referenced by allowance plug-ins must be identical in case.
- Token names and values can be any alphanumeric characters.
- Because token names appear in the URL as the value of the tokenname query parameter, it is best to keep the name short.
- Because of the XML-based configuration of the URL Processor configuration file, you must represent XML delimiter characters through
codes, as noted in the following table. Because allowance plug-ins use a simple text file for tokens, these codes are not necessary.

<table>
<thead>
<tr>
<th>Character</th>
<th>URL Processor Representation</th>
<th>Allowance Plug-in Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>&amp;</td>
<td>&amp;</td>
<td>&amp;</td>
</tr>
<tr>
<td>'</td>
<td>'</td>
<td>’</td>
</tr>
<tr>
<td>“</td>
<td>&quot;</td>
<td>“</td>
</tr>
</tbody>
</table>

**Starting and Stopping URL Processor**

The following sections explain how to start and stop JBoss and the URL Processor application. Optionally, you can change the port that URL Processor uses to listen for secure URL requests.

**Tip:** After you start URL Processor the first time, follow the instructions in the section “Testing the URL Processor” on page 76 to verify URL Processor operation.

**Starting JBoss and URL Processor**

You start JBoss and the URL Processor application using the startup scripts located in the directory $JB0SS_HOME/bin. On UNIX, navigate to this directory and enter the following at the command line prompt:

```
> ./secmgrstart.sh
```

On Windows, double-click the secmgrstart.bat file, or start it from the command line. For example:

```
C:\jboss-4.0.2\bin> secmgrstart.bat
```

**Tip:** Optionally, you can list start-up options by including the help flag (-h) after the script name.

**Stopping URL Processor**

To stop URL Processor on UNIX, issue a kill command with the process ID (pid), or run the shutdown.sh script in the $JB0SS_HOME/bin directory.
On Windows, you can enter `ctrl+C` at the command window you used to start URL Processor, or run the shutdown.bat batch file in the `$JBOS_HOME/bin` directory.

**Changing the JBoss Port**

By default, the JBoss application server uses port 8080 as its listen port. As explained in the section “Using the URL Processor Servlet” on page 71, a process can generate a secure URL by submitting an HTTP POST request to this port. If port 8080 is already in use, you can change the listen port to any other open port.

**Tip:** To ensure access to URL Processor by appropriate processes only, use a firewall to manage access to the selected port.

▷ **To change the JBoss port:**

1. Using any text editor, open the file `server.xml` in the following path under the JBoss installation directory:
   
   `$JBOS_HOME/server/secmgr/deploy/jbossweb-tomcat55.sar`

2. Change the Connector port value to any unused port. For example:
   
   `<Connector port="9045" address="${jboss.bind.address}"`

This chapter explains how to set up Helix Server and Helix Proxy for use with Security Manager. It describes how to install and configure the allowance plug-in.

**For More Information:** For instructions on installing and configuring Helix Server and Helix Proxy, see the installation chapters of *Helix Server Administration Guide* and *Helix Proxy Administration Guide*, respectively.

### Installing the Allowance Plug-in

The following sections explain how to install the allowance plug-ins on each Helix Server or Helix Proxy.

**For More Information:** Refer to “Submitting URLs to URL Processor” on page 59 for background information about how secure URLs are generated and validated.

### Installing and Verifying the Allowance Plug-in License

Using the allowance plug-in requires a valid Security Manager license file for Helix Server or Helix Proxy. Follow the next procedure to provide a license file for each streaming server and proxy.

► **To install and verify the allowance plug-in license:**

1. Copy the Security Manager license file (`secmgr-license.lic`) to the Helix Server or Helix Proxy license directory, which is under the main installation directory. For example, use this command to copy the license file on UNIX:

   ```
cp secmgr-license.lic /usr/local/Real/HelixServer/License
   ```

   On Windows, the default license directory for Helix Server is the following (Helix Proxy is similar):
C:\Program Files\Real\Helix Server\License

2. Using a text editor, open the license file, secmgr-license.lic.

3. Check that the license file has the EnableAggregation variable set to true. If
   EnableAggregation is not set to true, contact your RealNetworks
   representative.

   **Warning!** Do not make any changes to the license file. Doing so
   disables Helix Server or Helix Proxy.

### Installing the Allowance Plug-in

Perform the following procedure to install the allowance plug-in.

- **To install the allowance plug-in:**
  1. Uncompress the server adapter package in the Security Manager archive.
     The package contains separate subdirectories holding the plug-ins for the
     supported operating systems.
  2. Copy the appropriate allowance plug-in from the Security Manager server
     adaptor package to the Plugins directory of Helix Server or Helix Proxy:
     - Helix Server: daucplin.so or daucplin.dll
     - Helix Proxy: pdaucplin.so or pdaucplin.dll
  3. On UNIX systems, grant execute permission to the allowance plug-in file.
     For example:
     ```
     chmod 755 daucplin.so
     ```
     - or -
     ```
     chmod 755 pdaucplin.so
     ```

### Creating a Token File for the Allowance Plug-In

Tokens provide the means by which the URL Processor generates secure hash
keys and the allowance plug-in verifies the secure URL. The token file is not
necessary if you plan to use just a single, default token value. In this case,
define the default token in the URL Processor and allowance plug-in
configuration files. If you want to set up multiple tokens, however, you can
create a token file for each allowance plug-in as described in the following
procedure.
For More Information: Refer to “Setting Basic URL Processor Values” on page 13 for information about defining the default token on URL Processor. The section “Defining Token Handling” on page 28 explains the allowance plug-in default token.

To create a token file for use by the allowance plug-in:

1. In the Helix Server or Helix Proxy main installation directory, open a text file in which you will create a list of tokens.

2. In the text file, create any number of token name and token value pairs in the following format:

   `tokenname1:tokenvalue1
   tokenname2:tokenvalue2
   tokenname3:tokenvalue3
   ...additional name and value pairs...

   Warning! The list of tokens must be identical to the tokens defined under the `<keys>` element of the URL Processor configuration file. For more information, refer to “Defining URL Processor Tokens” on page 19.

3. Save the token file under any file name.

4. If you are using multiple Helix Servers or Helix Proxies that will handle URLs using the same token names, you can copy the token file to each Helix Server or Helix Proxy installation directory.

   Tip: If you use a different directory to hold the token file, you must update the allowance configuration information, as explained in “Defining Token Handling” on page 28.

Configuring the Streaming Server Allowance Plug-in

Follow the next procedure to configure the allowance plug-in. Before you attempt to configure the plug-in, decide which features you wish to use with the plug-in, and how you want to configure these features. The following sections explain the major plug-in features:

- “Setting Basic Allowance Plug-in Parameters” on page 27.
- “Error Handling for Allowance Plug-ins” on page 41.
- “Setting Up Allowance Bypass” on page 53.
To configure the allowance plug-in:

1. Navigate to the Helix Server or Helix Proxy installation directory. On UNIX, for example:
   
   `/usr/local/Real/HelixServer`
   
   On Windows, the default location is the following:
   
   `C:\Program Files\Real\Helix Server`
   
2. Back up the configuration file for Helix Server (`rmserver.cfg`) or Helix Proxy (`rmproxy.cfg`), and then open it in a text editor.

3. To the end of the configuration file, add a `DAUCPlugin` list:
   
   `<!-- DAUC Plugin -->
   <List Name="DAUCPlugin">
   ... allowance plug-in lists and variables ...
   </List>`

4. In the `DAUCPlugin` list, add the variables described in the section “Setting Basic Allowance Plug-in Parameters” on page 27.

5. Refer to the section “Error Handling for Allowance Plug-ins” on page 41 if you want to define which actions occur on error conditions, such as when a secure URL’s lifetime expires. Defining custom error actions requires that you create error handlers, then define the actions that each handler performs.

6. Optionally, you can enable allowance bypass, which allows you to define certain content that is streamed without authorization. Refer to the section “Setting Up Allowance Bypass” on page 53 for set-up instructions.

7. Save the configuration file.

8. Restart Helix Server or Helix Proxy to put the new settings into effect.

Testing the Allowance Plug-in

Follow the next procedure to test the allowance plug-in by generating and streaming a secure URL, then verifying that a log file has been created.

To test the allowance plug-in:

1. Test the allowance plug-in at the URLProcessor testing page, as described in “Testing the URL Processor” on page 76.

2. Click the media link generated by the URLProcessor testing page. If the system configuration is correct, the clip plays in a media player.
3. After you’ve successfully streamed a clip, navigate to the Helix Server logs directory under the main installation directory, such as the following on UNIX:

```
/usr/local/Real/HelixServer/logs
```

or Windows:

```
C:\Program Files\Real\Helix Server\logs
```

Verify that the specified log file, such as dauc.log, has been created. This file contains all of the allowance information from the server.

### Setting Basic Allowance Plug-in Parameters

The following sections describe the basic configuration parameters for the allowance plug-in. Most parameter are standalone variables that you add to the DAUCPlugin list of the Helix Server or Helix Proxy configuration files, as described in the section “Configuring the Streaming Server Allowance Plug-in” on page 25. Some features require the addition of sublists, however. The following provides an example of the basic allowance plug-in parameters:

```xml
<List Name="DAUCPlugin">
  <Var Enabled="1"/>
  <Var TokenFileName="filename"/>
  <Var DefaultToken="value"/>
  <Var KeyExpiryTime="30"/>
  <Var UseFullURL="0"/>
  <Var AuthLogging="1"/>
  <Var AuthLogStyle="7"/>
  <Var AuthLogFile="Logs/dauc.log"/>
  <Var LogRollSize="2"/>
  <Var UseUserAlerts="1"/>
  <Var AllowanceDefault="Deny"/>
  <Var SourceAddressHeaderValue="ADDRESS_HEADER"/>
</List>
```

...Abbreviations for query string parameter keywords...

...Other allowance plug-in lists and variables...
Enabling the Allowance Plug-in

To enable the allowance plug-in to authorize URL requests, set the Enabled variable to the value 1. If you set this attribute to 0 (zero), the plug-in passes through all requests without validating them.

Defining Token Handling

The following variables define how the allowance plug-in handles the token names and values that create secure URLs:

<Var TokenFileName="file_name"/>
<Var DefaultToken="value"/>
<Var KeyExpiration="seconds"/>
<Var UseFullURL="0"/>

TokenFileName

The optional TokenFileName parameter provides the name of the file that stores the token name and value pairs. The section “Creating a Token File for the Allowance Plug-In” on page 24 explains how to create this file. You can omit this variable from the configuration if you plan to use only the default token value.

Tip: Token files typically reside in the main Helix Server or Helix Proxy installation directory, which is also the directory from which you start Helix Server or Helix Proxy. If the file and the startup directory differ, however, the TokenFileName value must include a relative path from the startup directory to the file.

DefaultToken

The DefaultToken value is used to evaluate the secure hash key if no token name parameter is present in the secure URL. If URL Processor enables this feature, the allowance plug-in’s DefaultToken value must match the DefaultToken value defined in the URL Processor configuration, as described in the section “Setting Basic URL Processor Values” on page 13.

KeyExpirationTime

The KeyExpirationTime parameter sets the valid duration in seconds for the secure hash key generated by URL Processor. The default value is 30, meaning that the plug-in must receive the URL request within 30 seconds of when the URL
Processor generated the secure URL. If that time has expired, the allowance plug-in issues an URL Timed Out condition. The plug-in can then either reject the access attempt or forward the URL to an external allowance server, as described in the section “Error Handling for Allowance Plug-ins” on page 41. A value of 0 prevents secure hash keys from expiring.

**Note:** The plug-in determines when the URL was generated by evaluating the gtime parameter in the URL. To ensure that URL expiration functions correctly, the system times on the media server and the application server running URL Processor should be synchronized through a time-syncing program. For more on gtime, refer to “Generation Time” on page 65.

**Tip:** If the token_ttl argument is included in the URL submitted to URL Processor, its value overrides the KeyExpireTime parameter. See “Specifying the URL Timeout” on page 67.

**UseFullURL**

This parameter determines whether the allowance plug-in includes the URL’s protocol, server address, and port when verifying the secure hash key. The setting should be the same as the URL Processor UseFullURL configuration parameter, which the section “Setting Basic URL Processor Values” on page 13 explains. The allowance plug-in’s UseFullURL parameter takes a Boolean value:

- 0  Allowance plug-in uses just the path and query string from the URL to verify the key. This is the default value.
- 1  Allowance plug-in uses the complete URL, including the protocol, server IP address, and port, to verify the key.

**Tip:** RealNetworks recommends that you use the default setting of 0. If UseFullURL is enabled, the allowance plug-in may erroneously reject URLs that include a /ramgen, /sdpgen, or /asxgen mount point.

**Configuring Allowance Logs**

The following variables define logging features for the allowance plug-in:

```xml
<Var AuthLogging="0|1"/>
<Var AuthLogStyle="1-7"/>
<Var AuthLogFileName="path/file_name"/>
<Var LogRollSize="MB"/>
```
For More Information: Refer to Appendix A for information about the fields included in each authorization log entry.

AuthLogging
The AuthLogging parameter turns logging on or off for the plug-in. The default value is 1, which enables logging. Set this parameter to 0 to disable logging entirely.

AuthLogStyle
The AuthLogStyle parameter specifies which authorization request results are logged. The default value of 7 means to log all granted, rejected, and skipped authorization requests. The following table lists the possible values you can choose.

<table>
<thead>
<tr>
<th>Style</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Log pass results (access granted) only.</td>
</tr>
<tr>
<td>2</td>
<td>Log fail results (access denied) only.</td>
</tr>
<tr>
<td>3</td>
<td>Log both pass and fail results.</td>
</tr>
<tr>
<td>4</td>
<td>Log skipped validations (requests for which validation was not required).</td>
</tr>
<tr>
<td>5</td>
<td>Log both pass results and skipped validations.</td>
</tr>
<tr>
<td>6</td>
<td>Log both fail results and skipped validations.</td>
</tr>
<tr>
<td>7</td>
<td>Log all.</td>
</tr>
</tbody>
</table>

For More Information: Refer to “Setting Up Allowance Bypass” on page 53 for information about how to skip certain URL validations.

AuthLogFile
The AuthLogFile parameter sets the path and file name of the log created by the allowance plug-in. The file is a text file typically stored in the Logs subdirectory beneath the Helix Server or Helix Proxy installation directory. You can specify an absolute path, or a relative path from the directory used to start Helix Server or Helix Proxy. The following is an example of a relative path from the Helix Server or Helix Proxy installation directory:

<Var AuthLogFile="Logs/dauc.log"/>
LogRollSize

The LogRollSize parameter sets the maximum size of the allowance plug-in log file in Megabytes (MB). Helix Server or Helix Proxy creates a new log file once the current file reaches this size. It archives the old file by appending a timestamp in the format YYYYMMDDHHMMSS to the end of the file name. For example, an archive log created on January 24, 2006 at 4:56.13 P.M. looks like this:

dauc20060124165613.log

Enabling Error Handling

If set to 1, the UseUserAlerts variable enables the allowance plug-in to deliver error messages or perform other actions on an error, such as when a URL’s secure key expires. If you set this to 0, the allowance plug-in takes no actions other than denying access to content when errors occur.

Note: Activating user alerts delivers the default error messages described in the section “Error Types and Default Messages” on page 41. You can define other error actions as well, as the section “Error Handling for Allowance Plug-ins” on page 41 explains.

Setting Allowance Bypass Operation

The AllowanceDefault variable is used with the bypass feature described in “Setting Up Allowance Bypass” on page 53. Its setting determines whether URLs that meet the allowance bypass criteria are allowed or denied access:

Allow  Allow unsecured accesses. Enforce security only on URLs that meet the specified allowance bypass criteria. This is the setting you typically use if the majority of your content does not require authorization.

Deny   Deny unsecured accesses. Enforce security on all URLs except those that meet the bypass criteria. This is the default setting, which is the typical choice if most of your content requires authorization.

Selecting the IP Address Source

If you are using the client IP address validation feature described in the section “Verifying Client IP Addresses” on page 68, you can add the following line to the allowance plug-in configuration file. This parameter instructs the
plug-in to extract the client IP address from the specified header of the RTSP or HTTP request:

```xml
<Var SourceAddressHeaderName="ADDRESS_HEADER"/>
```

If you leave this line out of the configuration file, the plug-in determines the IP address from the client IP packets, which may cause IP address validation to fail if the server resides behind a certain type of firewall.

For More Information: Refer to the section “Selecting the IP Address Source” on page 71 for more about the two different methods for obtaining the client IP address.

### Defining URL Parameter Keywords

As the section “Abbreviating Query String Parameter Names” on page 16 explains, you can configure the keywords for most query string parameters in the secure URL. This allows you to keep the URL as short as possible and to avoid name collisions with parameters used for business needs outside of the scope of Security Manager. To enable configurable parameter, add the URLKeywords list to the DAUCPlugin list:

```xml
<List Name="DAUCPlugin">
    ...other allowance plug-in variables...
<List Name="URLKeywords">
    <Var DigestAlgorithmKeyword="da"/>
    <Var GTimeKeyword="gtime"/>
    <Var IPAddressKeyword="ip"/>
    <Var HashKeyKeyword="key"/>
    <Var LifeKeyword="life"/>
    <Var NetmaskKeyword="nm"/>
    <Var TokennameKeyword="t"/>
    <Var TokenTimeoutKeyword="ttl"/>
</List>
</List>
```

Note: The keyword abbreviations in the preceding example (ttl, key, life, and so on) are suggested values only. However, the values you specify must be identical in the configuration files of the URL Processor, streaming servers, and download servers.
This chapter explains how to configure the allowance plug-ins on download servers and proxies, which may be Apache 2.0 servers or Covalent Enterprise Ready Servers (ERS). This chapter assumes that the download servers and proxies have already been installed.


Installing the Download Server Allowance Plug-in

This section describes how to install and configure the download server allowance plug-in. To perform the installation, you need access to the server configuration file, httpsd.conf.

Installing the Download Server Allowance Plug-in

Carry out the following procedure to install the allowance plug-in on a download server or proxy.

To install the download server allowance plug-in:

1. The download server allowance plug-in requires a valid Helix Security Manager license to function. Copy the license file (secmgr-license.lic) to any download server directory. For example:

   cp secmgr-license.lic /usr/local/apache/servers/server_name/licenses

2. Uncompress the server adapter package in the Security Manager archive. The package contains separate subdirectories holding the plug-ins for the supported operating systems.
3. Into a convenient directory on the download server, copy the allowance plug-in from the Security Manager server adapter package:

UNIX:       moddauc.so
Windows:    moddauc.dll

4. On the download server, create a directory named moddauc in the modules directory.

5. Into the directory moddauc, copy the download server allowance plug-in.

6. On Windows only, copy the files msvc71.dll and msvcr71.dll from the apache subdirectory in the server adapter package to the download server’s bin directory.

Configuring the Download Server Allowance Plug-in

The next procedure explains how to configure the download server allowance plug-in by editing the httpd.conf file. The editing procedure adds a configuration section like that shown in the following example. Depending on the features you use, you may not need to add all of the following sections:

...existing configuration information...
<DAUCPlugin>
   Enabled 1
   DefaultToken value
   TokenFileName /usr/local/apache/servers/server_name/conf/tokenfile
   AuthLogging 7
   AuthLogFile /usr/local/apache/servers/server_name/logs/dauc.log
   AllowanceDefault Deny
   KeyExpiryTime 1
   LicenseFileName /usr/local/apache/servers/server_name/licenses/secmgr-license.lic
   Version 2.0
   SourceAddressHeaderName ADDRESS_HEADER
<URLKeywords>
   DigestAlgorithmKeyword da
   GTimeKeyword gtime
   IPAddressKeyword ip
   HashKeyKeyword key
   LifeKeyword life
   NetmaskKeyword nm
   TokenNameKeyword t
   TokenTimeoutKeyword ttl
</URLKeywords>
MatchedExpressionMask 4
<MatchExpressions>
  Var Expr_1 "^\?access=free"
</MatchExpressions>
ErrorAction "2"
</ErrorWebService>
</DAUCPlugin>

➤ **To configure the allowance plug-in:**

1. On the download server, navigate to the directory conf.
2. Using a text editor with search capability, open the file httpsd.conf.
3. Search for the section “Dynamic Shared Object (DSO) Support” and add the following to the list, indicating the path to the allowance plug-in on your server machine:

   LoadModule dauc_module "path/moddauc.so"

   For example:

   LoadModule dauc_module "/usr/local/apache/servers/server_name/modules/moddauc.so"

4. On a new line, add the following text, replacing server_name with the name of your download server’s main directory:

   <DAUCPlugin>
     Enabled 1
     TokenFileName token_file
     AuthLogging 1
     AuthLogFile /usr/local/apache/servers/server_name/logs/dauc.log
     AllowanceDefault Allow|Deny
     KeyExpiryTime 30
     LicenseFileName license_file
     Version 2.0
     DefaultToken default_token
   </DAUCPlugin>
The following table describes the basic configuration file for the allowance plug-in.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowanceDefault</td>
<td>This property is used with the allowance bypass feature described in the section “Setting Up Allowance Bypass” on page 53. The value can be Allow or Deny. For information about choosing a setting, refer to “Setting Allowance Bypass Operation” on page 31.</td>
</tr>
<tr>
<td>AuthLogFileName</td>
<td>This property indicates the file name of the log created by the allowance plug-in. Use the absolute path to the log file.</td>
</tr>
<tr>
<td>AuthLogging</td>
<td>Use this property to determine if the validation result of each request is logged. For details about possible values, refer to the section “AuthLogging” on page 30.</td>
</tr>
<tr>
<td>DefaultToken</td>
<td>Default token value. For more information, refer to “DefaultToken” on page 28.</td>
</tr>
<tr>
<td>Enabled</td>
<td>This property enables or disables the plug-in. The value 1 enables system security. If you set this to 0 (zero), the plug-in passes all requests through to the server.</td>
</tr>
<tr>
<td>KeyExpiryTime</td>
<td>This parameter sets the valid duration in seconds for the key generated by the portal. The default value is 30, meaning that the key generated by URL Processor is valid for 30 seconds. For KeyExpiryTime to function correctly, the system times on Helix Server and the application server must be synchronized. To simplify this, it is recommended that you use a time-syncing program. Note that if the parameter Token Time to Live (token_ttl) is included in the URL, it overrides the TokenKeyExpiryTime parameter.</td>
</tr>
<tr>
<td>LicenseFileName</td>
<td>Name and path of the Security Manager license file.</td>
</tr>
<tr>
<td>SourceAddressHeaderName</td>
<td>Determines how the plug-in extracts the client IP address for address verification. For more information, refer to “Selecting the IP Address Source” on page 31.</td>
</tr>
<tr>
<td>TokenFileName</td>
<td>Name, along with full or relative path, of the text file that contains the list of tokens, as explained in the section “Creating a Token File for the Download Server” on page 38.</td>
</tr>
</tbody>
</table>

5. To enable configurable parameter names as described in “Abbreviating Query String Parameter Names” on page 16, add text such as the following to the end of the DAUCPlugin list:
<DAUCPlugin>
...plug-in configuration information...
  <URLKeywords>
    DigestAlgorithmKeyword da
    GTimeKeyword gtime
    IPAddressKeyword ip
    HashKeyKeyword key
    LifeKeyword life
    NetmaskKeyword nm
    TokennameKeyword t
    TokenTimeoutKeyword ttl
  </URLKeywords>
</DAUCPlugin>

Note: The keyword abbreviations in the preceding example (key, life, ttl, and so on) are suggested values only. The abbreviations are configurable to prevent name collision issues. However, the values you specify must be identical in the configuration files of the URL Processor, streaming servers, and download servers.

6. To enable allowance bypass, which is described in the section “Setting Up Allowance Bypass” on page 53, you add a MatchedExpressionVariable and a MatchExpressions list to the main DAUCPlugin section as shown in the following example:

<DAUCPlugin>
...other plug-in configuration information...
  MatchedExpressionMask value
  <MatchExpressions>
    Var Expr_1 "MatchExpression_1"
    Var Expr_2 "MatchExpression_2"
  </MatchExpressions>
</DAUCPlugin>

7. To contact an external allowance server on a URL timeout, add a section such as the following to the configuration section:

<DAUCPlugin>
...other plug-in configuration information...
  ErrorAction "2"
  <ErrorWebService>
    Host "errorserver.example.com"
    Port "8080"
    URL "/errorPages/handleError.jsp"
  </ErrorWebService>
</DAUCPlugin>
ConnectionTimeout "30"
ResponseTimeout "15"
</ErrorWebService>
</DAUCPlugin>

For More Information: The section “Error Handling for Allowance Plug-ins” on page 41 explains external allowance. The download server’s capabilities are more limited than those of Helix Server or Helix Proxy, and the section “Contacting an External Allowance Server” on page 44 explains the meaning of the preceding syntax.

8. Save the httpsd.conf file.

9. Restart the download server.

Tip: You can test the download server Allowance Plug-in as described in the section “Testing the URL Processor” on page 76.

Creating a Token File for the Download Server

Tokens provide the means by which the URL Processor generates secure hash keys and the allowance plug-in verifies the secure URL. The token file is not necessary if you plan to use just a single, default token value. In this case, define the default token in the download server configuration files as the value of the DefaultToken parameter. If you want to set up multiple tokens, however, you can create a token file for each download server as described in the following procedure.

➢ To create a token file for use by the download server allowance plug-in:

1. In the server conf directory, open a text file in which you will create a list of tokens.

2. In the text file, create any number of token name and token value pairs in the following format:

tokenname1:tokenvalue1
tokenname2:tokenvalue2
tokenname3:tokenvalue3
...additional name and value pairs...

Warning! The list of tokens must be identical to the tokens defined under the <Keys> element of the URL Processor
configuration file. For more information, refer to “Defining URL Processor Tokens” on page 19.

3. Save the token file under any file name.

4. In the configuration file (httpsd.conf), set the value of the TokenFileName parameter to the path and file name of the token file.

5. Save the httpsd.conf file.

6. Restart the download server.
This chapter covers advanced features that you can configure for allowance plug-ins on streaming and download servers and proxies. The first several sections explain how to define error handlers to carry out specific actions on authorization errors. The last section covers allowance bypass, which enables you to host both free and secured content on the same server.

Error Handling for Allowance Plug-ins

By default, an allowance plug-in detects server and access errors, sending error messages to the media player. These error messages are not part of the RTSP standard, however, and are ignored by all media players except the desktop version of RealPlayer. The allowance plug-in therefore allows you to customize error actions for all players. If a URL has timed out, for example, you can redirect the user to an explanatory Web page.

Error Types and Default Messages

The allowance plug-in detects several types of errors, which are described in the section “Allowance Codes” on page 84. The following table lists the error conditions (along with the default messages) for which you can define customized error actions.

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Condition</th>
<th>Default Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>InternalError</td>
<td>The plug-in did not receive necessary values from the server. The plug-in may have also received an out-of-memory error. Or, if an external allowance server is used, that server may be unavailable.</td>
<td>Internal error. Please try again.</td>
</tr>
</tbody>
</table>

(Table Page 1 of 2)
### Error Condition Default Error Messages (continued)

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Condition</th>
<th>Default Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvalidURL</td>
<td>The URL validation failed because the URL was altered or had no secure hash key. The error may also have resulted from a plug-in configuration error.</td>
<td>Invalid URL. Please try again.</td>
</tr>
<tr>
<td>IPAddressMismatch</td>
<td>The IP address of the media player does not match the address of the client that requested the URL. This condition is reported explicitly only if the original IP address is included in the URL as the value of the &amp;ip query string parameter. See “Verifying Client IP Addresses” on page 68 for details.</td>
<td>IP address mismatch. Please report this problem to your system administrator.</td>
</tr>
<tr>
<td>URLTimedOut</td>
<td>The secure URL has reached the end of its life. This is the only condition reported on the download server and proxy. For more information, refer to “Specifying the URL Timeout” on page 67.</td>
<td>This URL has timed out.</td>
</tr>
</tbody>
</table>

(Permission Page 2 of 2)

### Defining Error Handlers

For each of the customizable error conditions, you can configure the allowance plug-in to take a specific error action. You do this by defining an error handlers list in the allowance plug-in section of the configuration file for the streaming or download server or proxy. This list associates each error type with an named error handler, then defines the action that each error handler performs.

**Note:** Download servers and proxies are limited to URLTimedOut handlers. For configuration information, refer to “Contacting an External Allowance Server” on page 44.

### Creating Error Handlers on Helix Server or Helix Proxy

To create the customized error handlers for Helix Server or Helix Proxy, you set up an ErrorHandlers list within the allowance plug-in section of the server or proxy configuration file. This list provides user-defined names for the error handlers to use for any of the customizable error conditions. Following the
error handlers list, you create a list for each of the error handlers. The following example shows the structure of the error handler section:

```xml
<List Name="DAUCPlugin">
...
<List Name="ErrorHandlers">
  <Var InternalError="HandleInternalError"/>
  <Var InvalidURL="HandleInvalidURL"/>
  <Var IPAddressMismatch="HandleInvalidIPAddress"/>
  <Var URLTimedOut="HandleURLTimedOut"/>
</List>
</List>

<!-- Lists that define the actions performed by each error handler -->
<List Name="HandleInternalError">
  ...variables that define the actions to perform on an internal error...
</List>
<List Name="HandleInvalidURL">
  ...variables that define the actions to perform on an invalid URL...
</List>
<List Name="HandleInvalidIPAddress">
  ...variables that define the actions to perform on an invalid IP address...
</List>
<List Name="HandleURLTimedOut">
  ...variables that define the actions to perform on a URL timeout...
</List>
</List>
```

**Note:** You also need to set the UseUserAlerts parameter to 1 in the server or proxy configuration file. See the section “Enabling Error Handling” on page 31.

### Creating a Customized Error Message

To create a customized message for an error condition on Helix Server or Helix Proxy, you define the error handler as described in the section “Defining Error Handlers” on page 42. You then create an ErrorAction attribute with a value of 0, and define an error message up to 1024 bytes in any character set as the value of an ErrorMsg attribute. The following is an example of a customized error message for an error handler defined by the name HandleInternalError:

```xml
<List Name="HandleInternalError">
  <Var ErrorAction="0"/>
  <Var ErrorMsg="Your request could not be fulfilled due to an internal error."/>
</List>
```
Tip: Keep in mind that only the desktop RealPlayer can display this error message to the user. All other media players ignore the message.

Redirecting a URL on an Error

URL redirection is useful for any of the configurable error conditions (InvalidURL, IPAddressMismatch, or URLTimedOut) that occur on Helix Server or Helix Proxy. To define it, you create an error handler as described in the section “Defining Error Handlers” on page 42. You then set an ErrorAction attribute with a value of 1, and define the redirection URI as the value of an ErrorRedirectURL attribute. Here is an example:

```xml
<List Name="HandleInvalidURL">
  <Var ErrorAction="1"/>
  <Var ErrorRedirectURL="rts://helixserver.example.com/InvalidErr.rm"/>
</List>
```

Note: The redirected content must be a RTSP URL. For URL redirection to work, the media player must respect the RTSP REDIRECT command at any point in the session. Some media players may not respect the command at all, or may respect it only during the RTSP session setup, not during the session itself.

Tip: When redirecting the user to an unsecure URL such as that shown in the preceding example, you must allow access to the content as described in the section “Setting Up Allowance Bypass” on page 53. To redirect the user to a secure URL, use an external allowance server (rather than the simple URL redirection shown here) to generate a new, secure URL for each redirected request.

Contacting an External Allowance Server

On any configurable error condition, the allowance plug-in can contact an external allowance server that instructs the plug-in on the action to take. The external server may allow or disallow access, as well as redirect the client to a different URL. This is particularly useful for the IPAddressMismatch and URLTimedOut conditions.
Defining External Allowance Error Handling for Streaming Servers

To define this error action, you create an error handler as described in the section “Defining Error Handlers” on page 42. You then set an ErrorAction attribute with a value of 2, and define an error service using the parameters of an ErrorWebService sublist, as shown in the following example. The table “ErrorWebService List Variables” on page 46 explains these variable values:

```xml
<List Name="HandleURLTimedOut">
  <Var ErrorAction="2"/>
  <List Name="ErrorWebService">
    <Var Host="errorserver.example.com"/>
    <Var Port="8080"/>
    <Var URL="/errorPages/handleError.cgi"/>
    <Var ConnectionTimeout="30"/>
    <Var ResponseTimeout="15"/>
  </List>
</List>
```

Setting Up External Allowance Error Handling for Download Servers

For a download server or proxy, you can create an external allowance error handler only for the URLTimedOut error condition. The following example shows the structure of the error handler section:

```xml
<DAUCPlugin>
  ...additional allowance plug-in parameters...
  ErrorAction "2"
  <ErrorWebService>
    Host "http://errorserver.example.com"
    Port "8080"
    URL "/errorPages/handleError.jsp"
    ConnectionTimeout "30"
    ResponseTimeout "15"
  </ErrorWebService>
</DAUCPlugin>
```
Configuring Error Web Service Attribute Values

The ErrorWebService sublist provides the information needed by the allowance plug-in to establish a connection to an external allowance server. It contains the attributes described in the following table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Host</strong></td>
<td>IP Address</td>
<td>Address of the external allowance server to contact to when the error occurs. If the plug-in must go through a proxy, set the Host value to the proxy server address. Do <strong>not</strong> include a protocol value such as http://.</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>0-65535</td>
<td>The port corresponding to the address of the host server or proxy. The default is 80.</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td>path</td>
<td>The path needed to execute the appropriate logic for obtaining an alternate error action. This information is appended to the host and port information.</td>
</tr>
<tr>
<td><strong>Connection Timeout</strong></td>
<td>seconds</td>
<td>The number of seconds that the allowance plug-in waits to establish a connection to the external allowance server. If the plug-in cannot open a socket, it halts the connection attempt, rejects the request, sends the default error response, and logs an error. The default value of 0 applies the operating system timeout.</td>
</tr>
<tr>
<td><strong>Response Timeout</strong></td>
<td>seconds</td>
<td>The number of seconds that the allowance plug-in waits for the external allowance server to respond after the connection has been established. If there is no response after this time elapses, the plug-in rejects the request, sends the default error response, and logs an error. The default is 10 seconds.</td>
</tr>
</tbody>
</table>

Posting Information to the External Allowance Server

The allowance plug-in sends the external allowance server the path and query string information in an HTTP POST such as the following, which allows the request to be proxied by a standard HTTP proxy:
POST URL HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length:

ClientIP=IP_Address
Url=URL
bytes=byte_range

**Note:** The URL value includes only the path and query string parameters. It does not include the protocol, server, and port information.

**Client IP**

The allowance plug-in extracts the ClientIP value from the IP packets or the RTSP headers, as described in the section “Selecting the IP Address Source” on page 31. Because this may be, in some cases, the address for a firewall, the ClientIP value is not a reliable means for the external allowance server to validate the client. RealNetworks recommends that the external allowance server identify clients through URL parameters added to the request URL, such as the guid value.

**For More Information:** See “GUID” on page 64.

**Byte Range**

The bytes value defines the range of bytes requested in the HTTP header. It provides support for the progressive download feature of HTTP, which allows the viewer to download a file in segments, pausing the download as needed or resuming a download that was interrupted. The bytes value indicates which segment of the file is being requested. This information is applicable only to download servers and proxies, and the bytes value is not included in posts sent by Helix Server or Helix Proxy.

When a reallocation request includes a bytes value, the external allowance server can use the information to evaluate the request, comparing it to the byte ranges in other requests for the same file from the same user. To do this, the server can retrieve the number of bytes already downloaded, as recorded in the download server or proxy access logs.
Byte Range Format

A byte range is typically not included in the first HTTP request. In subsequent requests, the bytes value may be reported as a range separated by a hyphen. Each subsequent range is separated from the preceding range by a comma:

bytes=first_byte-last_byte, first_byte-last_byte, ...other byte ranges...

If the first_byte value is omitted, it is assumed to be 0. If the last_byte value is omitted, it is assumed to be the last byte in the file. The following are sample bytes ranges:

Bytes 0 through 499: bytes=0-499
First 5000 bytes: bytes=0-5000
Byte 5000 to the end of the file: bytes=5000-
Bytes 100 through 200, and 400 through 500: bytes=100-200,400-500

For More Information: For more on HTTP pause support, refer to http://www.w3.org/Protocols/rfc2616/rfc2616.html.

Responding to a Reallowance Request

When the external allowance server receives a request from the allowance plug-in, it should evaluate the path and query parameters. It must return a message that contains a response code and, depending on the code, other values that instruct the plug-in to perform one of the following actions.

For More Information: The section “Reallowance Codes” on page 85 lists all of the codes that the allowance plug-in logs based on the response of the external allowance server.

Response Code 1 – Use Default Error

If the allowance server returns 1 as the response code, the allowance plug-in carries out the default action for the error condition, as described in the section “Error Types and Default Messages” on page 41. Here is a sample response:

HTTP/0K
Content-Type: text/plain
Content-Length: length

Response=1
CHAPTER 5: Configuring Advanced Authorization Features

Response Code 2 – Allow Access

The response code 2 instructs the allowance plug-in to grant the user access to the requested media without redirection. For example:

HTTP/OK
Content-Type: text/plain
Content-Length: length

Response=2

Response Code 3 – Custom Error Message

Response code 3 returns the error message specified through an ErrorMessage parameter. For example:

HTTP/OK
Content-Type: text/plain
Content-Length: length

Response=3
ErrorMessage=Your request cannot be fulfilled. Please try again later.

Note: This response code functions only with streaming servers and proxies. For Helix Server and Helix Proxy, define the error message handler described in the section “Creating a Customized Error Message” on page 43.

Response Code 4 – RTSP Redirect

Response code 4 includes a Location parameter that redirects the client to a second server if the URL is a valid RTSP or HTTP URL. For content on streaming servers, but not download servers, you can also perform an unvalidated, internal redirection to different content on the same server. The response HTTP looks like the following:

HTTP/OK
Content-Type: text/plain
Content-Length: length

Response=4
Location=URL
External Redirection

To redirect the client to a different streaming or download server, the external allowance server returns a fully-qualified RTSP or HTTP URL for the new content. The allowance plug-in on the server that receives the redirected client validates the new URL. The allowance server must therefore submit the new URL and any necessary query string parameters to URL Processor as described in Chapter 6.

Typically, an external allowance server does not resubmit any query string parameters related to security to URL Processor. For example, it must remove the key parameter before regenerating a secure hash key. As explained in “Modifying Parameters on URL Redirection” on page 51, however, you may resubmit the original guid parameter value to record the original access and the redirected access under the same GUID.

Internal Redirection

For content on Helix Server or Helix Proxy, the external allowance server can redirect the client to different content on the same server. This internal redirection does not use the RTSP REDIRECT command. Rather, the allowance plug-in simply directs the server to stream the new content. To cause an internal redirect, the external allowance server returns the full path to the new content, starting with the content mount point, along with any necessary query string parameters. For example:

/media/validated/songs/hit.rm?parameter=value&parameter=value

Because the allowance plug-in does not revalidate the URL, the server does not need to return a secure key parameter. The authorization log records the query string parameters that are resubmitted, however, and some parameters may be necessary for proper Security Manager operation. For example, to set a streaming lifetime for the new content, add the life parameter to the URL.

**Warning!** Redirect users to a clip that has the same encoding settings—such as the same media format, codec, and streaming bandwidth—as the clip initially requested. Otherwise, the media player may experience an error when it receives media in an unexpected format.
CHAPTER 5: Configuring Advanced Authorization Features

Modifying Parameters on URL Redirection

To direct a client to a different streaming or download server, an external allowance server needs to generate a new, secure URL. The allowance server may use new query string parameters in this URL, or it may reuse parameters from the original URL. As well, the allowance server may need to include parameters from the original URL when redirecting the client to new content on the same streaming server (an internal redirection). This section provides guidelines for reusing query string parameter values in these instances.

Resubmitting General Parameter Values

For query string parameters generated automatically, such as gtime, URL Processor always creates a new value. If you submit a URL that already includes a gtime value, the new, secure URL will include two gtime values:

...&gtime=first_time...&gtime=second_time...

In instances of double parameters, allowance plug-ins always read the first, older value and ignore the second, newer value. Depending on the parameter, this may or may not be desirable. The following table lists the parameters included in secure URLs. It explains how to handle each parameter when using external or internal redirection.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>user-defined</td>
<td>You can leave, remove, or update any query string parameter that does not affect the security system.</td>
</tr>
<tr>
<td>da</td>
<td>If the original URL contains the da parameter to override the default algorithm (MD5 or SHA-1), you typically leave the parameter unchanged in the new URL submitted to URL processor. In rare instances, older allowance plug-ins may reject URLs that use the SHA-1 algorithm. In this case, you can use da=0 to re-encode the URL using MD5.</td>
</tr>
<tr>
<td></td>
<td>On an internal redirect, the da parameter is not necessary because it is not included in the secure URL (only the original URL submitted to URL Processor). For more information, refer to “Overiding the Default Encryption” on page 67.</td>
</tr>
</tbody>
</table>

(Table Page 1 of 3)
### Parameter Reuse for External and Internal Redirection (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| gtime     | Always remove the gtime value before resubmitting a URL to URL processor. If you do not, allowance plug-ins use the older gtime value, which may result in the plug-in erroneously timing out the URL.  
On an internal redirect, the gtime value is not required because the allowance plug-in does not revalidate the URL. Include it only if you want to record the value in the authorization logs. For more information, refer to “Generation Time” on page 65. |
| guid      | URL Processor adds a unique guid value any URL that does not contain one. If you leave the existing guid in the URL, URL Processor retains that value. Remove the existing guid to record the access under a new guid value.  
On an internal redirect, the allowance server can use the original guid value or change it, depending on the value it wants to record in the authorization logs. For more information, refer to “GUID” on page 64. |
| ip        | If the URL contains an ip query string value with or without a address value, leave the parameter if you want to preserve IP address validation. In this case, the method used to generate the new, secure URL must pass the client IP address to URL Processor as an argument. If you want to remove IP address validation because of problems with firewalls or proxies, for example, remove the ip query string parameter and value, and regenerate the URL without passing an IP address to URL Processor.  
On an internal redirect, the ip value is not required because the allowance plug-in does not revalidate the URL. Include it only if you want to record the value in the authorization logs. For more information, refer to “Verifying Client IP Addresses” on page 68. |
| key       | Always remove the key value before resubmitting a URL to URL Processor. If you do not, allowance plug-ins use the older key value, which causes the access to fail.  
On an internal redirect, the key value is not required because the allowance plug-in does not revalidate the URL. Include it only if you want to record the value in the authorization logs. For more information, refer to “Security Key” on page 64. |

(Table Page 2 of 3)
### Parameter Reuse for External and Internal Redirection (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| life      | You can change or keep the life value in the new URL. Note that if you regenerate the URL using a different life value, but do not remove the original life parameter from the resubmitted URL, allowance plug-ins use the original life value.  

On an internal redirect, include the life value to set a lifetime for the streaming media. Leaving the value out grants full access to the entire clip or broadcast. For more information, refer to “Setting a Clip Lifetime” on page 65. |
| nm        | On external or internal redirection, leave or remove the nm query string parameter and value based on whether you leave or remove the ip query string parameter. For more information, refer to “Verifying Client IP Addresses” on page 68. |
| tokenname | You can change or keep the tokenname value in the new URL. Note that if you generate the URL using a different token name, but do not remove the original tokenname parameter from the resubmitted URL, the new URL will be invalid.  

On an internal redirect, the tokenname value is not required because the allowance plug-in does not revalidate the URL. For more information, refer to “tokenname” on page 74. |
| tokenttl  | You can add the tokenttl parameter if it was not present in the original URL. If you remove it from the resubmitted URL, the allowance plug-in’s KeyExpireTime value determines the secure URL lifetime. You can also update the parameter’s value in the resubmitted URL. Increasing the value may be necessary if the original URL timed out too quickly.  

On an internal redirect, the tokenttl value is not required because the allowance plug-in does not revalidate the URL. For more information, refer to “Specifying the URL Timeout” on page 67. |

**Warning!** URL Processor generates the secure hash key by hashing all of the query parameters, including duplicate parameters, with the token value. Removing a duplicate parameter from a secured URL invalidates the URL.

### Setting Up Allowance Bypass

Using the bypass feature, you can deliver both secure and unsecured content from the same streaming or download server. For example, you can set the
allowance plug-in to enforce secured access by default, then use allowance bypass to define content that offers unsecured access. Conversely, you can grant unsecured access to all content by default, using allowance bypass to enforce secured access for just some content. You can allow or deny access to content using a variety of criteria, such as the following:

- Content placed in a certain directory or accessed through a specific mount point. An easy way to allow unsecured access to certain clips is to set up the allowance bypass feature to allow access to all content in a directory such as /media/content/free.

- Content corresponding to a specific URL path, or path and file name. If you want to host free preview clips, for example, you can keep both the preview and feature clips together, allowing unrestricted access to the preview clip URLs while maintaining security on the feature clips.

- Content delivered through one IP address of a multihomed server. If your server has multiple IP addresses, you can make one IP address accessible only from behind your firewall, giving all internal users access to content without security restrictions.

**Note:** The value of the AllowanceDefault variable in the allowance plug-in configuration file determines if URLs that meet the allowance bypass conditions are granted or denied access. For more information, see “Setting Allowance Bypass Operation” on page 31.

**Allowance Bypass Syntax**

You define allowance bypass in the allowance plug-in section of the configuration file for the streaming or download server or proxy. The markup defines an expression mask, which determines how much of the URL to consider. It then lists any number of expressions that, when encountered in the URL, trigger the allowance bypass. The following is the syntax used in the Helix Server or Helix Proxy configuration file:

```xml
<List Name="DAUCPlugin">
  ...other allowance plug-in configuration information...
  <Var MatchedExpressionMask="value"/>
  <List Name="MatchExpressions">
    <Var Expr_1="Pattern_1"/>
    <Var Expr_2="Pattern_2"/>
```
...additional match expressions...
</List>
...
</List>

Use the following syntax for the download server and proxy:

```xml
<MatchExpressions>
  <Var Expr_1 "Pattern_1"
  <Var Expr_2 "Pattern_2"
  ...
</MatchExpressions>
```

### Setting the Expression Mask

The expression mask is a global value that determines which parts of request URLs the allowance plug-in considers when evaluating URLs for bypass. For example, consider the following URL:

rtsp://192.168.0.1:554/4minute_video.rm?userid=4321&subplan=12...

The plug-in breaks this URL down into three portions:

- **destination:** rtsp://192.168.0.1:554
- **path:** /4minute_video.rm
- **query string:** ?userid=4321&subplan=12...

The default value (1) for the expression mask causes the plug-in to evaluate only the URL path for bypass. By selecting a different mask value, you can change the pattern matching. The following table explains the mask values that you can choose.

<table>
<thead>
<tr>
<th>Value</th>
<th>URL Portions Evaluated for Pattern Matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>path portion</td>
</tr>
<tr>
<td>2</td>
<td>destination portion</td>
</tr>
<tr>
<td>3</td>
<td>path and destination portions</td>
</tr>
<tr>
<td>4</td>
<td>query string portion</td>
</tr>
<tr>
<td>5</td>
<td>path and query string portions</td>
</tr>
<tr>
<td>6</td>
<td>destination and query string portions</td>
</tr>
<tr>
<td>7</td>
<td>all three portions: destination, path, and query string</td>
</tr>
</tbody>
</table>
Note: Each server can use just one expression mask. For example, you cannot implement bypass allowance for some content based on paths and for other content based on IP address on the same server.

Expression Mask Examples

If you want to implement bypass allowance based solely on the Helix Server IP address, for example, you add the following to the allowance plug-in section of the Helix Server configuration file:

```xml
<Var MATCHED_EXPRESSION_Mask="2"/>
```

To allow free downloads according to matches on the destination, path, and query string, for instance, you add the following line to the download server configuration:

MATCHED_EXPRESSION_Mask 7

Defining Match Expressions

Once you set the expression mask, you define the expressions that trigger bypass allowance. In the MatchExpressions list, you create numbered entries in the form Expr_\_n. Each entry provides an expression that, when matched, triggers allowance. The following example demonstrates how to set up unsecured access to all content under the /free/ mount point on Helix Server or Helix Proxy, using the expression name FreeDirectory:

```xml
<Var MATCHED_EXPRESSION_Mask="1"/>
<List Name="MatchExpressions">
  <Var Expr_1="/free/"/>
</List>
```

The next example illustrates the download server syntax for providing free access to any URL in which the first query string parameter is ?access=free:

```xml
MATCHED_EXPRESSION_Mask 4
<MatchExpressions>
  Var Expr_1 "^\?access=free"
</MatchExpressions>
```

Note: To match a reserved character, use the character’s hex-encoded value. For a list of these characters, refer to “Reserved URL Characters” on page 62. Additionally, escape the percent sign character (%) that indicates a hex-encoded value. To match an ampersand (&), for example, you specify \%26.
Tip: The allowance plug-in evaluates URLs against the expressions in the order in which you list the expressions. To optimize plug-in performance when using multiple expressions, list expressions that are more specific in their match criteria before expressions that are more general.

Potential Security Holes

As a feature, Helix Security Manager allows a Web portal to attach information to a secure URL after the key query string parameter, as explained in the section “Security Key” on page 249. This information is not evaluated by allowance plug-ins for security, but may trigger allowance bypass if bypass criteria are not set correctly.

For example, suppose that your bypass criterion is the path string /free/. If the expression mask evaluates both the path and the query string components, adding /free/ to the end of the query string triggers the bypass. The solution in this case is to set the expression mask to evaluate only the path component.

If you use a query string parameter to trigger allowance bypass, anchor the parameter to the start of the string. For example, if the bypass parameter is access=free, add this parameter to the beginning of the query string, and set the match expression to ^\?access=free, as shown in the preceding example.

Using Regular Expressions

When defining match expressions, you can use the syntax of regular expressions commonly used with mark-up languages such as Perl. This allows you to add operators such as the carat (^), for example. The following is a sample expression that uses regular expression syntax:

<Var Premium = "^[a-z]+/premium"/>

For more information on regular expressions, see the following Web sites:

- http://www-2.cs.cmu.edu/People/rgs/pl-regex.html
- http://virtual.park.uga.edu/humcomp/perl/regex2a.html

Pattern Matching Examples

The following are additional examples of pattern matching:

1. Matching on a single directory:
"/public/demo/"

2. Matching on a per-URL basis, individual URLs can be directly specified in the expressions:
"/trial/sports/highlights\.rm"

3. Matching on a complete URL:
"^rtsp://[^/]*public/demo/"
Chapter 6: Generating Secure URLs

This chapter describes how to use the URL Processor, which generates the secure URLs evaluated by the allowance plug-ins on the streaming and download servers. Access to the URL processor is provided to Web portals by a J2EE Servlet and a Java class.

For More Information: You install the URL Processor as explained in Chapter 2. You define basic URL Processor operating features through the URL Processor configuration file, as described in the section “Configuring URL Processor” on page 12.

Submitting URLs to URL Processor

To distribute secured content to a user, a Web portal first processes all necessary business transactions through the user’s Web browser. It next submits to URL Processor an unsecured URL to the content that the user wants to view. URL Processor then returns to the Web portal a time-limited, secure URL to that content.

The Web portal forwards the secure URL to the user using any available method. For example, it may return an RTSP link to a streaming clip on Helix Server. To cause the user’s browser to launch a media player and pass it the secure URL, the portal formats the return response using the RealMedia MIME type (audio/x-pn-realaudio), SDP MIME type (application/sdp), or one of the Windows Media MIME types (such as video/x-ms-asf).

Warning! URL Processor does not validate the submitted URL or check the authenticity of the incoming request. You must therefore ensure that the Web portal submits a request for existing content. You must also set up appropriate network security measures for accessing URL Processor, such as by
using firewalls or IP address restrictions to prevent outside parties from requesting a URL directly from URL Processor.

**Note:** The Web portal should return to the user an RTSP link for streaming clips. This ensures that the user’s Web browser does not cache the media.

### Adding Query String Parameters to the Unsecured URL

The unsecure URL that the Web portal submits to URL Processor may contain any number of query string parameters. Some parameters may be applicable to your business needs whereas others, such as &token_ttl=30, specify certain security system features. For example, a submitted URL may look like this:

```
rtsp://helixserver.example.com/video.rm?userid=345923&token_ttl=30
```

The following table summarizes the query string parameters that the Web portal can add to the unsecure URL before submitting it to URL Processor. Later sections in this chapter describe the security parameters in detail.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Feature</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>user-defined value</td>
<td>no</td>
<td>Provides for any business needs outside of the scope of Helix Security Manager.</td>
<td>page 59</td>
</tr>
<tr>
<td>da</td>
<td>no</td>
<td>Overrides the default algorithm (MD5 or SHA-1).</td>
<td>page 67</td>
</tr>
<tr>
<td>ip</td>
<td>no</td>
<td>Indicates that explicit IP address verification is used.</td>
<td>page 70</td>
</tr>
<tr>
<td>nm</td>
<td>no</td>
<td>Specifies a netmask for IP address verification.</td>
<td>page 69</td>
</tr>
<tr>
<td>token_ttl</td>
<td>no</td>
<td>Overrides the default URL expiration time set by the allowance plug-in.</td>
<td>page 67</td>
</tr>
</tbody>
</table>

**Note:** By default, URL Processor includes all query string parameters in the submitted URL when creating the URL’s secure hash key. This means that removing or altering any of these parameters, even user-defined parameters, invalidates the secure URL.

**Tip:** By defining the RequiredURLAttributes configuration parameter for URL Processor, you can require that all URLs submitted to URL Processor contain specific query string
parameters. For details, refer to “Setting Basic URL Processor Values” on page 13.

Encoding URLs for Submission to URL Processor

URL Processor supports any valid URI defined in RFC 2396, which is located at http://www.faqs.org/rfcs/rfc2396.html. Specifically, a URI may contain any characters from the ASCII character set. URL Processor also supports hex encoding for some or all characters, such as using the value %20 to represent a space.

Tip: You can write URLs using any character set. However, you need to use hex encoding for any character outside of the valid set of unreserved characters. Browsers typically allow URLs to be typed in any character set, but convert the URL to a valid, encoded URL before transmitting it to the server.

Unreserved URL Characters

The following table lists the unreserved characters you can pass to URL Processor.

<table>
<thead>
<tr>
<th>Unreserved Characters</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>apostrophe</td>
<td>’</td>
</tr>
<tr>
<td>asterisk</td>
<td>*</td>
</tr>
<tr>
<td>exclamation point</td>
<td>!</td>
</tr>
<tr>
<td>hyphen</td>
<td>-</td>
</tr>
<tr>
<td>left parenthesis</td>
<td>(</td>
</tr>
<tr>
<td>lowercase letters</td>
<td>a-z</td>
</tr>
<tr>
<td>numerals</td>
<td>0-9</td>
</tr>
<tr>
<td>period</td>
<td>.</td>
</tr>
<tr>
<td>right parenthesis</td>
<td>)</td>
</tr>
<tr>
<td>tilde</td>
<td>~</td>
</tr>
<tr>
<td>underscore</td>
<td>_</td>
</tr>
<tr>
<td>uppercase letters</td>
<td>A-Z</td>
</tr>
</tbody>
</table>
Reserved URL Characters

The following table lists the reserved characters used only as delimiters, typically for query string parameters. You can pass one of these characters as text by using its hex-encoded value.

<table>
<thead>
<tr>
<th>Description</th>
<th>Character</th>
<th>Hexadecimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>“at” sign</td>
<td>@</td>
<td>%40</td>
</tr>
<tr>
<td>ampersand</td>
<td>&amp;</td>
<td>%26</td>
</tr>
<tr>
<td>colon</td>
<td>:</td>
<td>%3A</td>
</tr>
<tr>
<td>comma</td>
<td>,</td>
<td>%2C</td>
</tr>
<tr>
<td>dollar sign</td>
<td>$</td>
<td>%24</td>
</tr>
<tr>
<td>equals sign</td>
<td>=</td>
<td>%3D</td>
</tr>
<tr>
<td>forward slash</td>
<td>/</td>
<td>%2F</td>
</tr>
<tr>
<td>plus sign</td>
<td>+</td>
<td>%2B</td>
</tr>
<tr>
<td>question mark</td>
<td>?</td>
<td>%3F</td>
</tr>
<tr>
<td>semicolon</td>
<td>;</td>
<td>%3B</td>
</tr>
<tr>
<td>space</td>
<td></td>
<td>%20</td>
</tr>
</tbody>
</table>

**Note:** The Web portal must encode spaces in the URL as the hex value %20 before passing the URL to URL Processor.

**For More Information:** When you use the URL Processor servlet, ampersands that denote parameters must also be encoded. For details, refer to “Encoding Query String Parameters” on page 72.

Passing Arguments to URL Processor

As it submits the unsecure URL to URL Processor, the Web portal may pass several arguments that affect security. Including some of these arguments causes URL Processor to add query string parameters to the secure URL. For example, if you use an argument to specify a maximum viewing lifetime of 180 seconds for a clip, URL Processor automatically adds the query string parameter life=180 to the secure URL.
The following table summarizes the query string parameters in the secure URL that result from arguments passed to URL Processor.

**Query String Parameters Resulting from Arguments Passed to URL Processor**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Feature</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip</td>
<td>no</td>
<td>Instructs the allowance plug-in to enforce IP address verification for the client requesting the media.</td>
<td>page 68</td>
</tr>
<tr>
<td>life</td>
<td>yes</td>
<td>Sets the streaming lifetime for the clip.</td>
<td>page 65</td>
</tr>
<tr>
<td>tokenname</td>
<td>no</td>
<td>Informs the allowance plug-in of the token value to use when validating the secure hash key. (If not included, a default value is used.)</td>
<td>page 74</td>
</tr>
</tbody>
</table>

How you pass an argument to URL Processor depends on the URL Processor implementation you use:

- When using the Java servlet, you pass the arguments as parameters in an HTTP POST or GET method. See “Using the URL Processor Servlet” on page 71 for details.

- If you use the Java class, you specify the arguments as variables in a Java method. For more on this, refer to “Using the URL Processor Class” on page 73.

**Secured URL Returned by URL Processor**

URL Processor returns a secure URL that includes the query string parameters submitted in the original URL, such as tokenttl=30. To these values, URL Processor adds query string parameters, such as life=180, based on arguments passed to it. At the end of the URL, URL Processor adds the query string parameters that secure the URL. The following table summarizes these automatically-generated parameters.

**Query String Parameters Added Automatically to Secure URLs**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Feature</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>gtime</td>
<td>yes</td>
<td>Provides a timestamp that lets the allowance plug-in determine the secure URL’s expiration.</td>
<td>page 65</td>
</tr>
<tr>
<td>guid</td>
<td>yes</td>
<td>Sets a unique ID for each secure URL.</td>
<td>page 64</td>
</tr>
<tr>
<td>key</td>
<td>yes</td>
<td>Provides the secure hash key used by the allowance plug-ins to validate the access.</td>
<td>page 64</td>
</tr>
</tbody>
</table>
The resulting URL looks like the following example, in which line breaks have been added for clarity:

rtsp://helixserver.example.com/video.rm
?tokenttl=30&ip
&nm=255.255.255.0&gtime=1052261021&tokenname=key5&guid=1bc652c06242402588570003ba1871d&life=0&key=D832AC16CC54615E313723538AF6F278

**Note:** The secure URL has a length limit of 1024 characters. URL Processor can shorten the URL by encoding certain numeric values in Base64 and abbreviating query string parameters. For details, refer to “Setting Basic URL Processor Values” on page 13 and “Abbreviating Query String Parameter Names” on page 16.

**GUID**

URL Processor assigns each secure URL a unique identifier as the value of the guid parameter. Because a user might request a stream several times using the same secure URL, the guid value is not unique to each possible stream.

**Tip:** If you include a GUID as a query string parameter (guid=value) in the submitted URL, URL Processor uses that GUID value instead of generating its own. In this case, the Web portal must ensure uniqueness for each GUID.

**Security Key**

The key parameter provides a numeric value that secures the URL. URL Processor creates this value by hashing the value of the chosen token with the submitted URL and all of its query string parameters. It includes the client IP address in the hash if that option is chosen. The hashing algorithm is MD5 by default, but you can also use the SHA-1 algorithm.

When authorizing the URL request, the allowance plug-in regenerates the secure hash key using its default token value, or the value corresponding to the token name specified by the tokenname query string parameter. The plug-in grants access to the URL only if the regenerated hash key exactly matches the hash key passed as the value of the key query string parameter.
Note the following about how the security system handles query string parameters that precede and follow the security key:

- Because key generation by URL Processor and regeneration by an allowance plug-in includes the URL and all query string parameters submitted to URL Processor, changing any value in a secure URL up to and including the key value invalidates the URL.

- When validating a secure URL, allowance plug-ins ignore any query string parameters added to the secure URL after the key parameter and value. The Web portal can therefore add or remove any user-defined parameter after the key value without affecting the security system.

- Allowance plug-ins record all query string parameters for the secure URLs they evaluate, whether the parameters precede or follow the security key. For more information, refer to the section “URL” on page 82.

**Generation Time**

The gtime value indicates the time that URL Processor generated the URL. The allowance plug-in uses this value to determine if the URL is still valid. It compares the gtime value to the current time, taking the valid URL lifetime as either the value of the token_ttl query string parameter or the value of its KeyExpiryTime configuration value.

**For More Information:** For details on setting the URL lifetime, refer to the sections “Specifying the URL Timeout” on page 67 and “KeyExpiryTime” on page 28.

**Setting Basic Security Features**

The following sections explain basic security features that you can specify when submitting a URL to URL Processor.

**Setting a Clip Lifetime**

The life query string parameter sets the maximum number of seconds that the streaming session, whether a clip or live broadcast, may last. This is useful in pay-per-view billing arrangements, for limiting access to live broadcasts, or for streaming part of a clip as a preview. A value of 0 allows the viewer to view or download the entire clip.
Note that the lifetime determines how long Helix Server or Helix Proxy delivers the stream. It does not control how much of the clip timeline the viewer receives. When a clip streams at its encoded bandwidth, the two values are approximately equal. However, the server may take advantage of unused connection bandwidth when streaming to certain media players, such as desktop RealPlayers using the TurboPlay feature.

Suppose that a clip is encoded to stream at 256 Kbps and has a lifetime set to 60 seconds. If Helix Server streams the clip to a TurboPlay-enabled RealPlayer at 512 Kbps, the user will receive approximately 120 seconds of the clip timeline. This oversending of data does not occur with live broadcasts or with mobile devices, which typically lack an overbuffering feature because of memory limitations.

**For More Information:** For instructions on setting the life value by passing an argument to URL Processor, refer to either “Using the URL Processor Servlet” on page 71 or “Using the URL Processor Class” on page 73 depending on how you implement URL Processor.

**Tips for Setting a Clip Lifetime**

Note the following about setting a clip lifetime:

- For a downloadable clip, always use a value of 0 to enable a complete download.
- URL Processor does not validate that the chosen lifetime is valid, or that it is consistent with the size of the media clip.
- The secure URL includes the clip lifetime value as the &life=seconds query string parameter. The allowance plug-in on Helix Server or Helix Proxy then enforces the life query string value, declaring a URL Life Expired condition once the value elapses. At that point, the plug-in issues an error message and ends the session.
- Allowance plug-ins do not count user-initiated pauses against the lifetime. Refer to “Play and Pause Times” on page 83 for information about how allowance plug-ins log viewing time.
- Keep in mind that the lifetime determines how long the media streams, whereas the tokenttl query string parameter determines how quickly the user must request the stream after the secure URL is generated. See “Specifying the URL Timeout” on page 67.
• The stream lifetime resets on each access attempt. If you set the time window for a live broadcast to five minutes, for example, the viewer can view five minutes on the first access attempt, and five additional minutes by reusing the URL. To prevent this, set a lower URL lifetime through the tokenttl parameter. A value such as 30 seconds for tokenttl ensures that the URL has expired by the time the viewer attempts to reuse it. See “Specifying the URL Timeout” on page 67.

Specifying the URL Timeout

Allowance plug-ins on streaming and download servers set a lifetime for a secure URL. For example, each allowance plug-in may reject any access attempt in which 60 seconds or more have passed since the secure URL was generated. URL Processor includes a feature to override the default URL lifetime for a specific URL. To use this override, the Web portal submits the parameter tokenttl=seconds along with the unsecured URL. The maximum lifetime value is 2147483648 seconds. For example:

rtsp://helixserver.example.com/video.rm?tokenttl=30

In the preceding example, URL Processor sets a URL lifetime of 30 seconds. When an allowance plug-in validates the URL, it determines if the URL was generated more than 30 seconds ago by comparing the generation time provided by the gtime parameter to the current time according to its machine clock. If the URL has expired, the plug-in rejects the URL attempt or forwards the URL to an external allowance server.

Note: If you do not include a tokenttl value, the allowance plug-in’s KeyExpiryTime configuration value sets the default lifetime. For details, refer to the section “KeyExpiryTime” on page 28.

For More Information: The section “Generation Time” on page 65 explains the gtime parameter. Refer to “Error Handling for Allowance Plug-ins” on page 41 for information about external allowance servers.

Overriding the Default Encryption

As described in the section “Default Encryption Algorithm” on page 15, you can use either the MD5 or SHA-1 hash algorithm to compute the hash keys for secure URLs. By including the optional da query string in the URL submitted to URL Processor, you can override the algorithm choice on a case-
by-case basis. The parameter and value pair da=0 specifies the MD5 algorithm, whereas da=1 indicates SHA-1:
rtsp://helixserver.example.com/video.rm?da=1

**Note:** The da query string parameter is not included in the secure URL. Allowance plug-ins identify the encryption algorithm used based on the length of the key value.

### Verifying Client IP Addresses

URL Processor can include the client’s IPv4 address (with or without a netmask) as a value when generating the URL’s secure hash key. Allowance plug-ins then verify that the media player requesting the media from a streaming or download server uses the same IP address as the browser client that requested the media URL from the Web portal.

**Note:** IPv6 client addresses are not currently supported.

**Tip:** Regardless of whether you use IP address verification, allowance plug-ins always record in their log files a client IP address gathered from the IP packet source address field. For more information, refer to “Client IP Address” on page 82.

### Supplying the Client’s Address

To use the address verification feature, the Web portal extracts the client’s IP address by any method. For example, it may get the address from the HTTP request headers. It passes the dotted, IPv4 address, such as 207.188.7.188, to URL Processor as an argument. URL Processor then includes the IP address when computing the secure hash key. When the user requests the media, the allowance plug-in on the streaming or download server rejects the request if the requesting client’s IP address is different.

**Tip:** Using a specific IP address works for clients that have public IP addresses or that make both HTTP and RTSP requests through the same NAT firewall. In the latter case, URL Processor and the allowance plug-in both use the firewall’s IP address for verification. In other cases, you may need to apply a netmask.
For More Information: For instructions on passing the IP address as an argument to URL Processor, refer to either “Using the URL Processor Servlet” on page 71 or “Using the URL Processor Class” on page 73 depending on how you implement URL Processor.

Applying a Netmask

Before passing an IP address to URL Processor, a Web portal can apply a netmask in IPv4 dotted decimal format, such as 255.255.0.0. In this case, URL Processor uses the netmaked IP address to compute the secure hash key. The Web portal passes the netmask value to allowance plug-ins as the value of the &nm query string parameter in the URL it submits to URL Processor:

rtsp://helixserver.example.com/video.rm?...&nm=255.255.255.0

Situations Where Netmasks are Required

Including a netmask may be necessary for clients residing behind a corporate firewall. For example, if a browser’s HTTP request to the Web portal passes through an HTTP proxy and a media player’s RTSP request to a streaming server passes through a separate RTSP proxy, the two IP addresses will differ. To prevent authorization from failing, you can often apply a netmask to make the IP address value general to a range of addresses (a netblock or a subnet) rather than specific to one address.

Calculating Netmasks

Netmasks can define large netblocks or small subnets. In a netmask, the octet 255 preserves the corresponding octet in the original address. A netmask octet of 0 sets the corresponding IP address octet to 0. For example, applying a netmask of 255.255.255.0 to the IP address 207.188.7.188 matches the addresses 207.188.7.1 through 207.188.7.255. A netmask of 255.255.255.248, on the other hand, limits the mask to just eight machine addresses starting with the base IP address.

Implicit IP Address Validation

If the Web portal passes the IP address to URL Processor as an argument, but does not also include the address as a query string parameter, URL Processor adds three characters—&ip—to the query string of the secure URL. This parameter instructs the allowance plug-in to determine the media player’s IP address and use that address when regenerating and evaluating the secure hash key. If the URL also contains a netmask passed with the &nm parameter, the allowance plug-in uses the netmask when evaluating the hash key.

With this *implicit* method of validation, the allowance plug-in does not know the expected value of the client’s IP address. It extracts the IP address from RTSP, HTTP, or IP packets, and uses it as one of many values for regenerating the URL’s secure hash key. If the IP address of the media player differs from the address used by URL Processor to generate the secure hash key, the plug-in reports only that the secure key is invalid. In contrast, the *explicit* validation method allows the plug-in to report specifically that it rejected the access request because of an IP address mismatch.

Explicit IP Address Validation

Optionally, the Web portal can include the client’s IP address explicitly in the submitted URL as the value of the ip query string parameter. This enables the allowance plug-in to report that it rejected the access attempt specifically because of an IP address mismatch. The value of the ip query string parameter must be the same as that submitted to URL Processor in the IP address argument:

```
rtsp://helixserver.example.com/video.rm?...&ip=207.188.7.188
```

When evaluating the secure URL, the allowance plug-in gathers the requesting client’s IP address from the HTTP, RTSP, or IP packets. It recomputes the secure hash key to validate the access. If the recomputed key does not match the secure URL’s key, and the client’s actual IP address does not match the address passed through the ip query string parameter, the allowance plug-in logs an error code for an IP address mismatch when denying the request.

*For More Information:* The section “Allowance Codes” on page 84 explains codes logged when the allowance plug-in rejects an access attempt.

**Tips for Using Explicit IP Address Validation**

Note the following about using explicit IP address validation:
• Including the client IP address lengthens the full URL by up to 19 characters, which may cause errors with devices that limit URL length. The section “Configuring URL Processor” on page 12 explains different methods for shortening the secure URL’s length.

• URL validation fails if the original URL contains the &ip query string parameter (with or without an address value), but the Web portal does not pass URL Processor the client’s IP address as an argument. This is because the &ip parameter instructs the allowance plug-in to evaluate the secure hash key using the client IP address, but the URL Processor will have created the hash key without incorporating the address.

• When using explicit address validation, you can create a custom error message or action when the allowance plug-in rejects a request due to an IP address mismatch. For details, refer to “Error Handling for Allowance Plug-ins” on page 41.

Selecting the IP Address Source

By default, an allowance-plug-in determines the client IP address from the client’s IP packets. This may cause address validation to fail if the streaming and download servers reside behind a firewall that translates IP addresses of incoming requests. In this case, you may be able to configure the firewall to add the client’s actual IP address to the a specific header of the RTSP or HTTP request. You then configure the allowance plug-ins to gather the client IP address from the request headers rather than the IP packets.

For More Information: Refer to “Selecting the IP Address Source” on page 31 for information about plug-in configuration.

Using the URL Processor Servlet

Exposed by URLProcessorBean, the URL Processor servlet allows you to submit an unsecure URL to URL Processor using an HTTP GET or POST method. The servlet URL is the following:

http://Application_Server:port/URLProcessor

Tip: The following sections demonstrate HTTP GET methods. For an example of using POST, view the HTML source of the URL Processor test page, described in the section “Testing the URL Processor” on page 76.
Encoding Query String Parameters

As explained in the section “Adding Query String Parameters to the Unsecured URL” on page 60, you can add query string parameters to the unsecure URL. When using a GET method with the servlet, you must encode ampersands in the query string using the value %26. When using a POST method, however, you do not encode the ampersands. For example, the following URL submitted through a GET method includes three query string parameters:

rtsp://helixserver.example.com/video.rm?userid=345923%26t0kenttl=30%26da=1

URL Processor automatically decodes the ampersands in the secure URL it returns to the Web portal:

rtsp://helixserver.example.com/video.rm?userid=345923&token_ttl=30...

Passing Arguments Using the Servlet

The section “Passing Arguments to URL Processor” on page 62 summarizes arguments that you can pass to URL Processor. When using a GET method with the servlet, you pass the argument values as query string parameters appended to the end of the submitted URL. These parameters must use unencoded ampersands. For example, in the following URL, the life value instructs URL Processor to set the clip’s streaming lifetime to two minutes:

rtsp://helixserver.example.com/video.rm?userid=345923%26token_ttl=30%26da=1
&life=120

Because the life value must be passed to the allowance plug-in as well, URL Processor automatically includes the life query string parameter in the secure URL.

Tip: If you change the query string parameter names as described in “Abbreviating Query String Parameter Names” on page 16, use the names you define rather than the default names described here for both query string parameters and arguments passed to URL Processor.

Formatting the Response

By default, the servlet uses the mode debug=false, which returns the URL Processor response as a string formatted in the RealMedia MIME type (audio/x-pn-realaudio). The Web portal can then forward the string to the viewer’s browser to cause the browser to launch RealPlayer and request the content. The portal may also change the MIME type to launch a different player.
Optionally, you can include the debug=true argument with the submitted URL to return the secure URL as a link formatted within an HTML page, just like the URL Processor test page described in the section “Testing the URL Processor” on page 76. For example:

rtsp://helixserver.example.com/video.rm?userid=345923%26token_ttl=30%26da=1&life=120&debug=true

To forward the secure URL returned by URL Processor to a user, the Web portal would extract the URL from the HTML page and reformat it in a delivery means appropriate for the user.

Sending a Client IP Address through the Servlet

To verify the client’s IP address as described in the section “Implicit IP Address Validation” on page 70, you pass the IP address as an argument in the servlet:

rtsp://helixserver.example.com/video.rm?userid=345923%26token_ttl=30%26da=1&life=120&ip=207.188.7.188

To apply a netmask to the address, specify the netmask value using the nm query string parameter within the submitted URL. The parameter must use URL- encoded ampersands (%26) and precede the arguments:

rtsp://helixserver.example.com/video.rm?userid=345923%26token_ttl=30%26da=1%26nm=255.255.255.0&life=120&ip=207.188.7.188

To pass the client the IP address as described in “Explicit IP Address Validation” on page 70, include the IP address as a parameter as well as an argument. The following example includes a netmask, which is not required:

rtsp://helixserver.example.com/video.rm?userid=345923%26token_ttl=30%26da=1%26ip=207.188.7.188%26nm=255.255.255.0&life=120&ip=207.188.7.188

Using the URL Processor Class

The following sections explain the URL Processor arguments and methods that Web portals use to generate secure URLs.

Warning! URL Processor does not authenticate the process requesting the secured URL. The system integrator is responsible for implementing appropriate measures to secure access to URL Processor.
URL Processor Arguments

The following sections explain the arguments used in URL Processor methods. You include some arguments only if you want to use specific URL Processor features, such as IP address verification.

URL

The original URL and any query string parameters, as described in the section “Submitting URLs to URL Processor” on page 59. URL Processor does not verify that the submitted URL is valid.

tokenname

The name of one of the tokens defined in the URL Processor token file, which is described in the section “Defining URL Processor Tokens” on page 19. The URL Processor uses the token’s value to generate the secure hash key. URL Processor adds the token name to the secure URL using the query string tokennamename.

**Note:** The allowance plug-in that validates the URL must have the same list of tokens and values in its own token file, as described in the section “Creating a Token File for the Allowance Plug-In” on page 24.

**Tip:** You do not need to include a token value if URL Processor and allowance plug-ins are configured to use a default token value. For more information, refer to “Setting Basic URL Processor Values” on page 13.

timewindow

An integer value that sets the clip viewing lifetime in seconds. The value 0 means that the viewer can stream or download the entire clip. The secure URL passes the value to the allowance plug-in by appending the life=seconds query string parameter to the URL. For more information, refer to “Setting a Clip Lifetime” on page 65.

IPAddress

The dotted, IPv4 address, such as 207.188.7.188. This is used only for client IP address verification, as explained in the section “Verifying Client IP Addresses” on page 68.
CHAPTER 6: Generating Secure URLs

URL Processor Class

The URLProcessorManager class allows you to generate instances of the URL Processor Manager and to create secure URLs. Most methods throw java.lang.Exception.

generateHashURL

This method generates the secure hash key appended to all URL requests from streaming and download servers. It returns the secure URL:

```java
public generateHashURL(java.lang.String URL,
                      java.lang.String tokenname,
                      long timewindow)
```

Using IP Address Validation

To have allowance plug-ins validate the client IP address on the access attempt, a feature described in the section “Verifying Client IP Addresses” on page 68, you pass the IP address with any necessary netmask already applied in the URL method:

```java
public generateHashURL(java.lang.String URL,
                      java.lang.String tokenname,
                      long timewindow,
                      java.lang.String IPAddress)
```

getConfig

This method gets the URL Processor configuration from memory, returning it in XML format:

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

For More Information: The section “Configuring URL Processor” on page 12 explains the configuration parameters.

getConfigName

This method returns the path and file name of the URL Processor configuration file, secmgr-config.xml, which contains the URL Processor configuration information:

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

getInstance

This method returns the single instance of URLProcessorManager:
public static URLProcessorManager getInstance()

release
Use this method to release resources once all clients disconnect:
public void release()

reload
This method reloads the URLProcessorManager instance:
public void reload()

reloadConfig
This method loads the URL Processor configuration from file:
public void reloadConfig()

Testing the URL Processor

You can test the URL Processor using a testing page running under the application server. Running the test requires that URL Process is configured and operational. To stream clips using the URL returned by the test page, the allowance plug-in must be installed on the appropriate server, as described in “Installing the Allowance Plug-in” on page 23.

► To test the URL Processor:

1. Open the URL Processor page through the application server by entering the following URL in a Web browser:
   http://Application_Server:port/urlprocessor/URLProcessor
   The following example uses the local host address and the default URL Processor listen port:
   http://127.0.0.1:8080/urlprocessor/URLProcessor
   Note: You can change the default port from 8080 as described in the section “Changing the JBoss Port” on page 22.

2. In the URL field, enter the URL to a clip or broadcast. The media does not need to exist if you want only to test URL generation. If you want to request existing media, the allowance plug-in on the streaming or download server must be installed.
a. To request a media clip over RTSP, enter the RTSP URL to a clip on Helix Server, specifying the RTSP port if port 554 is not used. For example:
    rtsp://helixserver.example.com/test.rm

b. To request a Windows Media clip, enter an MMS URL, specifying the MMS port if port 1755 is not used. For example:
    mms://helixserver.example.com/test.wmv

c. To request a media download, enter the HTTP URL to a clip, specifying the HTTP port if port 80 is not used. For example:
    http://download.example.com/test.rm

d. To request a RealMedia clip using Ramgen, enter the HTTP URL to a clip on Helix Server, specifying the HTTP port if port 80 is not used, and include the /ramgen mount point. For example:
    http://helixserver.example.com/ramgen/test.rm

e. To request a streaming media clip using SDPgen, enter the HTTP URL to a clip on Helix Server, specifying the HTTP port if port 80 is not used, and include the /sdpgen mount point. For example:
    http://helixserver.example.com/sdpgen/test.rm

f. To request a Windows Media clip using ASXgen, enter the HTTP URL to a clip on Helix Server, specifying the HTTP port if port 80 is not used, and include the /asxgen mount point. For example:
   http://helixserver.example.com/asxgen/test.wmv

3. To the end of the base URL, append any required parameters that you defined through the RequiredURLAttributes attribute of the URL Processor configuration file. For testing purposes, the parameters can take any value, such as the following:
   rtsp://helixserver.example.com/test.3gp?userid=1234&subplan=1234

   **Note:** Your configuration may not require the inclusion of specific query string parameters in URLs submitted to URL Processor. For more information, refer to “Setting Basic URL Processor Values” on page 13.

4. For **Tokenname**, enter the name of a token defined for the URL Processor, as described in the section “Defining URL Processor Tokens” on page 19. URL Processor uses the token’s value to generate the secure hash key.
Note: If you created a default token as described in the section “Setting Basic URL Processor Values” on page 13, you can leave the token name blank.

5. The value for Life of the streaming session defines the length in seconds of the content streamed from Helix Server. Setting the value to 0 (zero) streams the entire clip. If you set the value to 30, for instance, the clip plays for 30 seconds.

Note: For downloaded clips, leave this value set to 0.

6. To use client IP address validation, enter a client IP address such as 207.188.7.188 in the IP Address field.

7. Set Debug to false to return the response formatted as a string in the RealMedia MIME type (audio/x-pn-realaudio). Set Debug to true to display the URL Processor response as a link within the test page. Choose the true option for any of the following reasons:
   a. The requested content does not exist or you do not want to test streaming or downloading with the content.
   b. The submitted URL contains the /ramgen, /asxgen, or /sdpgen mount point.
   c. You want to request the content using a media player other than RealPlayer.

8. Click the Send to Servlet button.
   a. If you set Debug to false, the browser launches RealPlayer and passes it the secure URL. RealPlayer then requests the content from the server.
   b. If you set Debug to true, the secure URL appears as a link in the browser. The link text provides the path to the content. The inclusion of security parameters such as guid and key indicates that URL Processor is functioning. For example:

   Tip: Click the link to stream or download the clip. If the returned link is an RTSP or MMS URL that does not include the /ramgen, /asxgen, or /sdpgen mount point, copy the URL
and stream the clip by opening the appropriate media player and requesting the clip through the player’s **File>Open** dialog.
This appendix explains the structure of authorization log entries created by allowance plug-ins.

For More Information: Allowance plug-ins must be configured to perform logging. For configuration details, refer to the section “Setting Basic Allowance Plug-in Parameters” on page 27.

Authorization Log Format

The following syntax illustrates the fields included in each log entry for an allowance plug-in:

log_type [access_time] URL response_code IP_address client_ID [client_start]
client_duration play_time pause_time [allowance_code allowance_message]
[reallowance_code realallowance_message]

The following is an example of a URL request for a streaming clip that the allowance plug-in denied because the secure hash key in the URL did not correspond to the URL components:

token_ttl=120&gtime=1052261021&tokenname=key5&guid=1bc652c062424025
885700003ba1871d&life=0&key=D832AC16CC54615E313723538AF6F278 0
-0800] 0 0 0 [5 Key Mismatch] [0 No Error]

Note: Allowance plug-in log entries for Helix Server, Helix Proxy, a download server, and a download proxy are identical. For download servers and proxies, certain fields applicable to streaming only, such as play_time and pause_time, always use the value 0.
Authorization Log Fields

The following sections explain the fields present in each allowance plug-in log entry.

Log Type

The log_type entry indicates the type of authorization log:

0 Initial authorization entry.

Access Time

The [access_time] field indicates the time at which the plug-in logged the entry according to the server clock. The entry uses the following format in which TZ is the server’s time zone offset from Coordinated Universal Time (Greenwich Time):

[dd/Mmm/yyyy:hh:mm:ss TZ]

For example:


URL

The URL field lists the URL in its encoded format exactly as the browser or media player requested it from the server.

Response Code

The response_code field lists the action that the allowance plug-in took in response to the URL request:

0 request not validated because of bypass rules (see “Setting Up Allowance Bypass” on page 53)
1 request allowed
2 request rejected

Client IP Address

The client_IP field records the IP address of the media player making the request. This address is gathered from the source address field of the request IP packets. If allowance plug-ins verify the client IP address when the client
accesses a secure URL, they may validate the access using an IP address from a
different source, such as the RTSP headers. Therefore, the IP address used for
URL validation may differ from that recorded for the access and used as the
value of this parameter. This may reflect the fact that the RTSP headers were
configured to provide the actual client IP address, whereas the IP request
packets used the address of a NAT firewall.

Client Identification

The client_ID field records the unique integer assigned to each client
connection by Helix Server or Helix Proxy. The numbers start with 1 and
increment by 1 for each new connection. After a server restart, the client_ID
values restart again at 1. Allowance plug-ins for download servers and proxies
do not record this value.

Streaming Times

The [client_start] field records the time according to the server clock that the
allowance plug-in authorized the URL and streaming began. The format is the
same as the [access_time] field:
[dd/MM/yyyy:hh:mm:ss TZ]
The client_duration field records an integer that indicates the total number of
seconds that the client has been connected to the stream, or has been
downloading the clip, since the [client_start] time began. For a streamed clip
or broadcast, this value includes any time that the viewer paused the stream.

Play and Pause Times

The play_time value is an integer that indicates the total number of seconds
that the media player has played the clip. The pause_time value is an integer
that records the number of seconds that the media player has paused the
stream. Together, the play_time and pause_time values equal the client_duration
time. Allowance plug-ins for download servers do not record the play and
pause times.

Note: Time spent rebuffering a stream is recorded in the
play_time value. Excessive stream rebuffering may therefore
result in recorded play times that are larger than actual.
Allowance Codes

The allowance_code and allowance_message fields record one of the codes and messages described in the following table.

<table>
<thead>
<tr>
<th>Code</th>
<th>Default Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
<td>Session started and ended normally.</td>
</tr>
<tr>
<td>1</td>
<td>Invalid URL</td>
<td>URL does not contain a secure hash key and is not granted access through bypass rules described in the section “Setting Up Allowance Bypass” on page 53. The section “Error Handling for Allowance Plug-ins” on page 41 describes the actions the plug-in can take.</td>
</tr>
<tr>
<td>2</td>
<td>URL Timed Out</td>
<td>Lifetime for the secure hash key has expired. The lifetime is determined by the tokeintl query string parameter or the allowance plug-in KeyExpireTime configuration parameter. The section “Error Handling for Allowance Plug-ins” on page 41 describes the actions the plug-in can take.</td>
</tr>
<tr>
<td>3</td>
<td>Internal Error</td>
<td>Plug-in experienced an error processing the request. The plug-in may not have received certain necessary values from the server or may be out of memory. The section “Error Handling for Allowance Plug-ins” on page 41 describes the actions the plug-in can take.</td>
</tr>
<tr>
<td>4</td>
<td>No Default Token</td>
<td>No token name was passed as the value of a tokeintname query parameter, and the allowance plugin is not configured to evaluate the secure hash key using a default token value. Refer to “Setting Basic URL Processor Values” on page 13 for information about default tokens.</td>
</tr>
<tr>
<td>5</td>
<td>Key Mismatch</td>
<td>Secure key value does not match the URL. Either the secure URL was changed or the allowance plug-in is not using the appropriate token value to evaluate the hash key.</td>
</tr>
<tr>
<td>6</td>
<td>Life Expired</td>
<td>Streaming session's life value has expired and the client was disconnected.</td>
</tr>
<tr>
<td>8</td>
<td>IP Address Mismatch</td>
<td>The client’s IP address as identified by the plug-in differed from that encoded into the secure URL by the URL Processor. For more information, refer to “Verifying Client IP Addresses” on page 68. The section “Error Handling for Allowance Plug-ins” on page 41 describes the actions the plug-in can take.</td>
</tr>
</tbody>
</table>
## Reallowance Codes

The `reallowance_code` and `reallowance_message` fields record a code and message describing the action that occurred if the allowance plug-in forwarded the URL to an external allowance server. Codes 100 and higher indicate errors in operation.

### Reallowance Codes and Messages

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>default response</td>
<td>External allowance server denied the request for content. Allowance plug-in responded with its default error action.</td>
</tr>
<tr>
<td>2</td>
<td>allow response</td>
<td>External allowance server requested that the client be given access to the content with no further action. Allowance plug-in granted access to the content.</td>
</tr>
<tr>
<td>3</td>
<td>custom error</td>
<td>External allowance server denied the request and returned a customized error message. Allowance plug-in forwarded the error message to the user.</td>
</tr>
<tr>
<td>4</td>
<td>redirect response</td>
<td>External allowance server requested a URL redirect. Allowance plug-in redirected the user to the URL supplied by the external allowance server.</td>
</tr>
<tr>
<td>101</td>
<td>Connection failed</td>
<td>Connection to the external allowance server was rejected, possibly due to an incorrect IP address or port, or misconfiguration of the external allowance server.</td>
</tr>
<tr>
<td>102</td>
<td>Connection timed out</td>
<td>Not able to connect to the external allowance server before the connection timeout period elapsed. This time is set by the <code>ConnectionTimeout</code> parameter, as described in the section “Configuring Error Web Service Attribute Values” on page 46.</td>
</tr>
<tr>
<td>103</td>
<td>Internal Error</td>
<td>A plug-in error occurred when processing data during an external allowance request.</td>
</tr>
<tr>
<td>104</td>
<td>Cannot read from server</td>
<td>Connection to the external allowance server succeeded, but the plug-in could not read from the socket.</td>
</tr>
</tbody>
</table>
### Reallowance Codes and Messages (continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>105</td>
<td>Reallowance server timed out</td>
<td>External allowance server did not respond before the configured timeout was reached. This maximum response time is set through the ResponseTimeout parameter, as described in the section “Configuring Error Web Service Attribute Values” on page 46.</td>
</tr>
<tr>
<td>300-599</td>
<td>dynamic message</td>
<td>External allowance server returned a non-200 HTTP response, indicating an error in the connection attempt. In this case, the plug-in takes the default action for the user.</td>
</tr>
</tbody>
</table>

**For More Information:** Refer to the section “Error Handling for Allowance Plug-ins” on page 41 for information about external allowance servers. The section “Responding to a Reallowance Request” on page 48 describes the exact HTTP response that the external allowance server uses for reallowance codes 1 through 4.
The authorization log described in Appendix A records actions that the allowance plug-in took on each request for a secure stream. This appendix explains how to create an additional, customized report in Helix Server or Helix Proxy that summarizes the reason each secure stream terminated.

Termination Codes

When a media stream disconnects, Helix Server or Helix Proxy records a reason for termination property (Client.*.ReasonForTermination). This property contains a numeric code that reflects both the initial action taken by the allowance plug-in and any real allowance action. Using the advanced logging feature, you can create a customized report that logs the termination reasons along with other stream information, such as the client IP address and requested URL.

The termination code results from adding the error codes listed in the table “Allowance Codes and Messages” on page 84 to the real allowance codes listed in the table “Reallowance Codes and Messages” on page 85. To ensure that each possible combination of the error code and real allowance code produces a unique result, the allowance code value is multiplied by 65536. Therefore, the formula for generating the termination code is the following:

The following table lists the termination codes, showing which combination of error code and realallowance code each termination code represents.

### Termination Codes

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Error</td>
<td>0</td>
<td>n/a</td>
<td>0</td>
</tr>
<tr>
<td>65536</td>
<td>Invalid URL</td>
<td>1</td>
<td>n/a (reallowance not available)</td>
<td>0</td>
</tr>
<tr>
<td>65537</td>
<td>Invalid URL</td>
<td>1</td>
<td>default response</td>
<td>1</td>
</tr>
<tr>
<td>65538</td>
<td>Invalid URL</td>
<td>1</td>
<td>allow response</td>
<td>2</td>
</tr>
<tr>
<td>65539</td>
<td>Invalid URL</td>
<td>1</td>
<td>custom error</td>
<td>3</td>
</tr>
<tr>
<td>65540</td>
<td>Invalid URL</td>
<td>1</td>
<td>redirect response</td>
<td>4</td>
</tr>
<tr>
<td>65637</td>
<td>Invalid URL</td>
<td>1</td>
<td>Connection failed</td>
<td>101</td>
</tr>
<tr>
<td>65638</td>
<td>Invalid URL</td>
<td>1</td>
<td>Connection timed out</td>
<td>102</td>
</tr>
<tr>
<td>65639</td>
<td>Invalid URL</td>
<td>1</td>
<td>Internal Error</td>
<td>103</td>
</tr>
<tr>
<td>65640</td>
<td>Invalid URL</td>
<td>1</td>
<td>Cannot read from server</td>
<td>104</td>
</tr>
<tr>
<td>65641</td>
<td>Invalid URL</td>
<td>1</td>
<td>Reallowance server timed out</td>
<td>105</td>
</tr>
<tr>
<td>65836-66135</td>
<td>Invalid URL</td>
<td>1</td>
<td>dynamic message</td>
<td>300-599</td>
</tr>
<tr>
<td>131072</td>
<td>URL Timed Out</td>
<td>2</td>
<td>n/a (reallowance not available)</td>
<td>0</td>
</tr>
<tr>
<td>131073</td>
<td>URL Timed Out</td>
<td>2</td>
<td>default response</td>
<td>1</td>
</tr>
<tr>
<td>131074</td>
<td>URL Timed Out</td>
<td>2</td>
<td>allow response</td>
<td>2</td>
</tr>
<tr>
<td>131075</td>
<td>URL Timed Out</td>
<td>2</td>
<td>custom error</td>
<td>3</td>
</tr>
<tr>
<td>131076</td>
<td>URL Timed Out</td>
<td>2</td>
<td>redirect response</td>
<td>4</td>
</tr>
<tr>
<td>131173</td>
<td>URL Timed Out</td>
<td>2</td>
<td>Connection failed</td>
<td>101</td>
</tr>
<tr>
<td>131174</td>
<td>URL Timed Out</td>
<td>2</td>
<td>Connection timed out</td>
<td>102</td>
</tr>
<tr>
<td>131175</td>
<td>URL Timed Out</td>
<td>2</td>
<td>Internal Error</td>
<td>103</td>
</tr>
<tr>
<td>131176</td>
<td>URL Timed Out</td>
<td>2</td>
<td>Cannot read from server</td>
<td>104</td>
</tr>
<tr>
<td>131177</td>
<td>URL Timed Out</td>
<td>2</td>
<td>Reallowance server timed out</td>
<td>105</td>
</tr>
<tr>
<td>131372-131671</td>
<td>URL Timed Out</td>
<td>2</td>
<td>dynamic message</td>
<td>300-599</td>
</tr>
<tr>
<td>196608</td>
<td>Internal Error</td>
<td>3</td>
<td>n/a (reallowance not available)</td>
<td>0</td>
</tr>
<tr>
<td>196609</td>
<td>Internal Error</td>
<td>3</td>
<td>default response</td>
<td>1</td>
</tr>
<tr>
<td>196610</td>
<td>Internal Error</td>
<td>3</td>
<td>allow response</td>
<td>2</td>
</tr>
<tr>
<td>196611</td>
<td>Internal Error</td>
<td>3</td>
<td>custom error</td>
<td>3</td>
</tr>
<tr>
<td>196612</td>
<td>Internal Error</td>
<td>3</td>
<td>redirect response</td>
<td>4</td>
</tr>
</tbody>
</table>

(Table Page 1 of 2)
## Termination Codes (continued)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>196709</td>
<td>Internal Error</td>
<td>3</td>
<td>Connection failed</td>
<td>101</td>
</tr>
<tr>
<td>196710</td>
<td>Internal Error</td>
<td>3</td>
<td>Connection timed out</td>
<td>102</td>
</tr>
<tr>
<td>196711</td>
<td>Internal Error</td>
<td>3</td>
<td>Internal Error</td>
<td>103</td>
</tr>
<tr>
<td>196712</td>
<td>Internal Error</td>
<td>3</td>
<td>Cannot read from server</td>
<td>104</td>
</tr>
<tr>
<td>196713</td>
<td>Internal Error</td>
<td>3</td>
<td>Reallowance server timed out</td>
<td>105</td>
</tr>
<tr>
<td>196908-197207</td>
<td>Internal Error</td>
<td>3</td>
<td><em>dynamic message</em></td>
<td>300 - 599</td>
</tr>
<tr>
<td>262144</td>
<td>No Default Token</td>
<td>4</td>
<td>n/a (reallowance not available)</td>
<td>0</td>
</tr>
<tr>
<td>327680</td>
<td>Key Mismatch</td>
<td>5</td>
<td>n/a (reallowance not available)</td>
<td>0</td>
</tr>
<tr>
<td>393216</td>
<td>Life Expired</td>
<td>6</td>
<td>n/a (reallowance not available)</td>
<td>0</td>
</tr>
<tr>
<td>524288</td>
<td>IP Address Mismatch</td>
<td>8</td>
<td>n/a (reallowance not available)</td>
<td>0</td>
</tr>
<tr>
<td>524289</td>
<td>IP Address Mismatch</td>
<td>8</td>
<td><em>default response</em></td>
<td>1</td>
</tr>
<tr>
<td>524290</td>
<td>IP Address Mismatch</td>
<td>8</td>
<td><em>allow response</em></td>
<td>2</td>
</tr>
<tr>
<td>524291</td>
<td>IP Address Mismatch</td>
<td>8</td>
<td><em>custom error</em></td>
<td>3</td>
</tr>
<tr>
<td>524292</td>
<td>IP Address Mismatch</td>
<td>8</td>
<td><em>redirect response</em></td>
<td>4</td>
</tr>
<tr>
<td>524389</td>
<td>IP Address Mismatch</td>
<td>8</td>
<td>Connection failed</td>
<td>101</td>
</tr>
<tr>
<td>524390</td>
<td>IP Address Mismatch</td>
<td>8</td>
<td>Connection timed out</td>
<td>102</td>
</tr>
<tr>
<td>524391</td>
<td>IP Address Mismatch</td>
<td>8</td>
<td>Internal Error</td>
<td>103</td>
</tr>
<tr>
<td>524392</td>
<td>IP Address Mismatch</td>
<td>8</td>
<td>Cannot read from server</td>
<td>104</td>
</tr>
<tr>
<td>524393</td>
<td>IP Address Mismatch</td>
<td>8</td>
<td>Reallowance server timed out</td>
<td>105</td>
</tr>
<tr>
<td>524588-524887</td>
<td>IP Address Mismatch</td>
<td>8</td>
<td><em>dynamic message</em></td>
<td>300 - 599</td>
</tr>
</tbody>
</table>

*(Table Page 2 of 2)*

## Termination Reports

You create a termination report using the advanced logging feature of Helix Server or Helix Proxy. RealNetworks recommends that you create a client properties report that produces output each time a client disconnects.

### Sample Report Using Helix Administrator

The following illustrations shows a termination report created using the graphical interface of Helix Administrator. To navigate to the advanced logging page, click **Logging & Monitoring > Advanced Logging**.
**Termination Report in Helix Administrator**

**Advanced Logging**

<table>
<thead>
<tr>
<th>Templates</th>
<th>Template Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reason for Term</td>
<td>Reason for Termination Report</td>
</tr>
<tr>
<td>Server Stats</td>
<td></td>
</tr>
</tbody>
</table>

**Add Template**

**Template Description (optional)**

**Outputs**

<table>
<thead>
<tr>
<th>Output Name</th>
<th>Output Type</th>
<th>File Name</th>
<th>Log Rolling Frequency</th>
<th>Log Rolling Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text File Output</td>
<td>File</td>
<td>termination_reason.txt</td>
<td>0 Hours (zero disables)</td>
<td>1 MB</td>
</tr>
</tbody>
</table>

**Add Output Type**

**Output Interval (optional)** :

**Disconnect Output Format**

```
%Date% %Time%
Server IP: %Client%.Session%.InterfaceAddress%
Client IP: %Client%.IPAddress%
Client ID: %Client%.clientID%
URL: %Client%.Session%.RequestedURL%
Termination Reason: %Client%.ReasonForTermination%
```

**Tip:** Although this report writes its output to a text file, you can produce many different kinds of outputs. You might send the results to a Web server using HTTP POST, for example.

**For More Information:** Refer to the advanced logging chapter of Helix Server Administration Guide or Helix Proxy Administration Guide.
Sample Report Syntax in the Configuration File

The termination report shown in the preceding section produces the following syntax in the configuration file. You can create or edit the report directly in the configuration file, which is rmserver.cfg for Helix Server or rmproxy.cfg for Helix Proxy:

```xml
<!--  T E M P L A T E D   L O G G I N G  -->
<List Name="LoggingTemplates">
...other reports...
<List Name="Reason for Termination Report">
  <Var Enabled="1"/>
  <Var Format="Time: %Date% %Time% 
Server IP: %Client.*.Session.*.InterfaceAddress% 
Client IP: %Client.*.IPAddress% 
Termination Reason: %Client.*.ReasonForTermination% 
Disconnect="\nTime: %Date% %Time% 
Server IP: %Client.*.Session.*.InterfaceAddress% 
Client IP: %Client.*.IPAddress% 
Client ID: %Client.*.ClientID% 
URL: %Client.*.Session.*.PlayerRequestedURL% 
Termination Reason: %Client.*.ReasonForTermination%"/>
<List Name="Outputs">
  <List Name="Text File Output">
    <Var Filename="termination_reason.txt"/>
    <Var Type="File"/>
    <Var LogRollSize="1"/>
  </List>
  </List>
  <Var Type="ClientStats"/>
</List>
</List>
```

For More Information: Refer to the logging chapter of Helix Server Configuration and Registry Reference or Helix Proxy Configuration and Registry Reference.

Sample Report Output

Within the report, the `%Client.*.ReasonForTermination%` property records the termination reason. You can add any other client or server properties recorded by Helix Server or Helix Proxy, as well as format the report using tabs and line breaks. The preceding report examples use the following format:
This format produces output such as the following each time a client disconnects:

Time: 04/26/06 15:41:49  
Server IP: 207.188.21.158  
Client IP: 207.188.25.138  
Client ID: WinNT_5.1_6.0.12.1040_RealPlayer_RN30PRM_en-us_686  
URL: rtsp://207.188.21.158/music_intro.rm  
Termination Reason: 131074

**For More Information:** For details about the properties that you can add to a report, as well as for instructions on formatting a report, refer to the registry properties section of Helix Server Configuration and Registry Reference or Helix Proxy Configuration and Registry Reference.
INDEX

A  address validation, see IP address validation
   allowance bypass
      bypass criteria, 56
      expression masks, 55
      overview, 53, 54
      regular expressions, 57
      security holes, 57
   allowance plug-in
      error handlers, 42
      error messages
         customized, 43
         default, 41
      interval log, 82
   log file
      access time, 82
      client ID, 83
      client IP, 82
      error codes, 84
      format, 81
      log type, 82
      play and pause times, 83
      reallocation codes, 85
      response code, 82
      streaming time, 83
      termination codes, 87
   URL requested, 82
   URL redirection, 44
   see also allowance bypass
   see also download server allowance plug-in
   see also external allowance
   see also Helix Server allowance plug-in

Client.*.ReasonForTermination property, 87
Covalent ERS, see download server

D  dauclpin plug-in, 24
   download server allowance plug-in
      configuration, 34
      configuration attributes
         AllowanceDefault, 36
         AuthLogFileName, 36
         AuthLogging, 36
         DefaultToken, 36
         DigestAlgorithmKeyword, 37
         Enabled, 36
         GTimeKeyword, 37
         HashKeyKeyword, 37
         IPAddressKeyword, 37
         KeyExpiryTime, 36
         LicenseFileName, 36
         LifeKeyword, 37
         NetmaskKeyword, 37
         SourceAddressHeaderName, 36
         TokenNameKeyword, 37
         TokenFileName, 36
         TokenTimeoutKeyword, 37
      installation, 33
      license file, 33
      machine requirements, 5
      see also allowance plug-in
   download server documentation, 33

E  external allowance
   byte range, 47
   configuration attributes, 46
   error handler
      download server, 45
      streaming server, 45
   POST messages, 46

C  client address validation, see IP address validation
responses
  allow access, 49
  customized message, 49
  default error, 48
  redirection
    external, 50
    internal, 50
  timeout values, 46

H Helix Server allowance plug-in
configuration parameters
  AllowanceDefault, 31
  AuthLogFileFileName, 30
  AuthLogging, 30
  AuthLogStyle, 30
  DefaultToken, 28
  DigestAlgorithmKeyword, 32
  Enabled, 28
  GTimeKeyword, 32
  HashKeyKeyword, 32
  IPAddressKeyword, 32
  KeyExpireTime, 28
  LifeKeyword, 32
  LogRollSize, 31
  NetmaskKeyword, 32
  SourceAddressHeaderName, 31
  TokenFileName, 28
  TokennameKeyword, 32
  TokenTimeoutKeyword, 32
  UseFullURL, 29
  UserUserAlerts, 31
configuration procedure, 25
daucplin file, 24
enabling, 28
installation
  license, 23
  plug-in, 24
IP address validation source, 31
machine requirements, 5
pdaucplin file, 24
testing, 26
tokens
  default token, 28
token file, 24
token file path, 28
see also allowance plug-in

I
  interval logging
    cumulative play time, 83
    log type indicator, 82
IP address validation
  address verification methods
    explicit, 70
    implicit, 70
  IPAddress parameter, 74
  ipquery string parameter, 70
  netmask values, 69
  nmquery string parameter, 69
  packet headers used, 71
  tips for using, 70

J
  Java environment, 9
  JAVA_HOME environment variable, 10
  JBoss, see URL Processor
  JBOSS_HOME environment variable, 10

L
  Log4j set-up, 11

M
  MD5 encryption algorithm, 15

P
  pdaucplin plug-in, 24

R
  /ramgen/ replacement, 19

S
  /sdpgen/ replacement, 19
  secure URLs
    /asxgen/ replacement, 19
    da parameter, 67
    gtime parameter, 65
    guid parameter, 64
    ip parameter, 70
    key parameter, 64
    keyword replacement, 18
    length limit, 64
    life parameter, 65, 74
    nm parameter, 69
    parameter name abbreviations, 16
    /ramgen/ replacement, 19
    required parameter values, 15
    reserved characters, 62
    /sdpgen/ replacement, 19
supported formats, 61
tokenname parameter, 74
tokenttl parameter, 67
SHA-1 encryption algorithm, 15
shutdown.bat script, 21
shutdown.sh script, 21
Sun JDK and JRE, 9

T
Technical Support, 2
termination codes
creating reports, 89
list of, 88
reason for termination property, 87
tokens
default
download server, 36
streaming server, 28
URL Processor, 13
guidelines, 20
passed to URL Processor, 74
token file
download server, 38
streaming server, 24
URL Processor, 19
tokenname query string parameter, 74

U
URL Processor
Base64 encoding, 14
configuration, 12
license file path, 13
configuration parameters
DefaultToken, 13
DigestAlgorithmKeyword, 17
EncryptionType, 15
GTimeKeyword, 17
HashKeyKeyword, 17
IPAddressKeyword, 17
LifeKeyword, 17
MACAddress, 14
NetmaskKeyword, 17
ReplaceKeywordInURL, 18
RequiredURLAttributes, 15
TokenNameKeyword, 18
TokenTimeOutKeyword, 18
URLKeywords, 16
UseBase64, 14
UseDefaultToken, 13
UseFullURL, 14
encryption algorithm
Base64 encoding, 14
MD5, 15
override, 67
SHA-1, 15
GUID, 64
hex encoding, 61
installation
procedure, 10
JAVA_HOME environment variable, 10
JBoss_HOME environment variable, 10
keyword replacement, 18
Log4j installation, 11
MAC address, 14
machine requirements, 4
media lifetime, 65, 74
method arguments
IPAddress, 74
originalURL, 74
timeWindow, 74
tokenName, 74
overview, 59
port value, 22
reserved characters, 62
resubmitting secure URLs, 51
secure URL format, 63
security key, 64
using full URL, 14
security restrictions for accessing, 59
shutdown, 21
start-up, 21
testing
main test page, 76
tokens
default token, 13
token file, 13, 19
URLs
formats, 61
generation time, 65
length limit, 64
parameter name abbreviations, 16
requiring parameter values, 15
time-to-live, 67
see also secure URLs
URL Processor token choice, 74
URLProcessorBean servlet
  encoding query string parameters, 72
  formatting the response, 72
  IP address validation, 73
  passing arguments, 72
URLProcessorManager class
  generateHashURL method, 75
  getConfig method, 75
  getConfigName method, 75
  getInstance method, 75
  release method, 76
  reload method, 76
  reloadConfig method, 76